

POWER7 and POWER7+ Optimization and Tuning Guide

Discover simple strategies to optimize
your POWER7 environment

Analyze and maximize
performance with solid solutions

Learn about the new
POWER7+ processor



Brian Hall
Mala Anand
Bill Buros
Miso Cilimdzc
Hong Hua
Judy Liu
John MacMillan
Sudhir Maddali
K Madhusudanan
Bruce Mealey

Steve Munroe
Francis P O'Connell
Sergio Reyes
Raul Silvera
Randy Swanberg
Brian Twichell
Brian F Veale
Julian Wang
Yaakov Yaari

Redbooks



International Technical Support Organization

POWER7 and POWER7+ Optimization and Tuning Guide

November 2012

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (November 2012)

This edition pertains to Power Systems servers based on POWER7 and POWER7+ processor-based technology. Specific software levels and firmware levels used are noted throughout the text.

© Copyright International Business Machines Corporation 2012. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team who wrote this book	ix
Now you can become a published author, too!	xiii
Comments welcome	xiv
Stay connected to IBM Redbooks	xiv
Chapter 1. Optimization and tuning on IBM POWER7 and IBM POWER7+	1
1.1 Introduction	2
1.2 Outline of this guide	2
1.3 Conventions that are used in this guide	4
1.4 Background	4
1.5 Optimizing performance on POWER7	5
1.5.1 Lightweight tuning and optimization guidelines	6
1.5.2 Deployment guidelines	13
1.5.3 Deep performance optimization guidelines	17
Chapter 2. The POWER7 processor	21
2.1 Introduction to the POWER7 processor	22
2.1.1 The POWER7+ processor	23
2.2 Multi-core and multi-thread scalability	23
2.3 Using POWER7 features	25
2.3.1 Page sizes (4 KB, 64 KB, 16 MB, and 16 GB)	25
2.3.2 Cache sharing	29
2.3.3 SMT priorities	35
2.3.4 Storage synchronization (sync, lwsync, lwarx, stwcx, and eieio)	37
2.3.5 Vector Scalar eXtension (VSX)	39
2.3.6 Decimal floating point (DFP)	44
2.3.7 Data prefetching using d-cache instructions and the Data Streams Control Register (DSCR)	46
2.4 Related publications	51
Chapter 3. The POWER Hypervisor	55
3.1 Introduction to the POWER7 Hypervisor	56
3.2 POWER7 virtualization	57
3.2.1 Virtual processors	57
3.2.2 Page table sizes for LPARs	61
3.2.3 Placing LPAR resources to attain higher memory affinity	61
3.2.4 Active memory expansion	64
3.2.5 Optimizing Resource Placement – Dynamic Platform Optimizer	65
3.3 Related publications	65
Chapter 4. AIX	67
4.1 AIX and system libraries	68
4.1.1 AIX operating system-specific optimizations	68
4.1.2 Using POWER7+ features under AIX	83
4.2 AIX Active System Optimizer and Dynamic System Optimizer	84

4.2.1	Concepts	84
4.2.2	ASO and DSO optimizations	86
4.2.3	Workloads	87
4.2.4	The asoo command	89
4.2.5	Environment variables	89
4.2.6	Installing DSO	91
4.2.7	Log files	91
4.3	AIX preferred practices	92
4.3.1	AIX preferred practices that are applicable to all Power Systems generations	92
4.3.2	AIX preferred practices that are applicable to POWER7	93
4.3.3	POWER7 mid-range and high-end High Impact or Pervasive advisory	93
4.4	Related publications	94
	Chapter 5. Linux	97
5.1	Linux and system libraries	98
5.1.1	Introduction	98
5.1.2	Linux operating system-specific optimizations	98
5.2	Related publications	106
	Chapter 6. Compilers and optimization tools for C, C++, and Fortran	107
6.1	Compiler versions and optimization levels	108
6.2	Advanced compiler optimization techniques	109
6.2.1	Common prerequisites	109
6.2.2	XL compiler family	110
6.2.3	GCC compiler family	112
6.3	IBM Feedback Directed Program Restructuring	114
6.3.1	Introduction	114
6.3.2	FDPR supported environments	115
6.3.3	Acceptable input formats	116
6.3.4	General operation	116
6.3.5	Instrumentation and profiling	117
6.3.6	Optimization	119
6.4	Related publications	123
	Chapter 7. Java	125
7.1	Java levels	126
7.2	32-bit versus 64-bit Java	126
7.3	Memory and page size considerations	127
7.3.1	Medium and large pages for Java heap and code cache	127
7.3.2	Configuring large pages for Java heap and code cache	128
7.3.3	Prefetching	128
7.3.4	Compressed references	128
7.3.5	JIT code cache	129
7.3.6	Shared classes	130
7.4	Java garbage collection tuning	130
7.4.1	GC strategy: Optthruput	130
7.4.2	GC strategy: Optavgpause	131
7.4.3	GC strategy: Gencon	131
7.4.4	GC strategy: Balanced	131
7.4.5	Optimal heap size	132
7.5	Application scaling	133
7.5.1	Choosing the correct SMT mode	133
7.5.2	Using resource sets	133
7.5.3	Java lock reservation	135

7.5.4	Java GC threads	135
7.5.5	Java concurrent marking	136
7.6	Related publications	136
Chapter 8. DB2	137
8.1	DB2 and the POWER7 processor	138
8.2	Taking advantage of the POWER7 processor	138
8.2.1	Affinitization	138
8.2.2	Page sizes	139
8.2.3	Decimal arithmetics	139
8.2.4	Using SMT priorities for internal lock implementation	140
8.3	Capitalizing on the compilers and optimization tools for POWER7	140
8.3.1	Whole-program analysis and profile-based optimizations	140
8.3.2	Feedback directed program restructuring (FDPR)	140
8.4	Capitalizing on POWER7 virtualization	140
8.4.1	DB2 virtualization	141
8.4.2	DB2 in an AIX workload partition	141
8.5	Capitalizing on the AIX system libraries	142
8.5.1	Using the thread_post_many API	142
8.5.2	File systems	142
8.6	Capitalizing on performance tooling	143
8.6.1	High-level investigation	143
8.6.2	Low-level investigation	144
8.7	Conclusion	144
8.8	Related publications	144
Chapter 9. WebSphere Application Server	147
9.1	IBM WebSphere	148
9.1.1	Installation	148
9.1.2	Deployment	148
9.1.3	Performance	149
9.1.4	Performance analysis, problem determination, and diagnostic tests	150
Appendix A. Analyzing malloc usage under AIX	151
	Introduction	152
	How to collect malloc usage information	152
Appendix B. Performance tooling and empirical performance analysis	155
	Introduction	156
	Performance advisors	156
Expert system advisors	156	
Rational Performance Advisor	161	
AIX	162	
CPU profiling	163	
AIX trace-based analysis tools	165	
Finding emulation issues	170	
hpmstat, hpmcount, and tprof -E	171	
Linux	171	
Empirical performance analysis using the IBM SDK for PowerLinux	172	
Using the IBM SDK for PowerLinux Trace Analyzer	173	
High library usage	173	
Deeper empirical analysis	174	
Java (either AIX or Linux)	176	
32-bit or 64-bit JDK	176	

Java heap size, and garbage collection policies and parameters	176
Hot method or routine analysis	177
Locking analysis	182
Thread state analysis	183
Appendix C. POWER7 optimization and tuning with third-party applications	185
Migrating Oracle to POWER7	186
Oracle 11gR2 preferred practices for AIX V6.1 and AIX V7.1 on Power Systems	188
Migrating Sybase ASE to POWER7	193
Implementing Sybase IQ to POWER7	195
Environment variables	195
Special considerations for Sybase IQ with SMT4 mode	196
Shared versus dedicated processors with Sybase IQ and POWER7	196
Migrating SAS to POWER7	197
Disk storage tuning	200
Migrating SAP BusinessObjects Business Intelligence platform with POWER7	203
Platform support	203
Sizing for optimum performance	204
Landscape design	204

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Active Memory™	IBM®	PowerPC®
AIX 5L™	Micro-Partitioning®	PowerVM®
AIX®	Power Architecture®	POWER®
DB2®	POWER Hypervisor™	PureSystems™
developerWorks®	Power Systems™	Rational®
DS4000®	Power Systems Software™	Redbooks®
DS6000™	POWER6®	Redbooks (logo)®
DS8000®	POWER7 Systems™	System Storage®
Enterprise Storage Server®	POWER7+™	System z®
FDPR®	POWER7®	Tivoli®
IBM PowerLinux™	PowerLinux™	WebSphere®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

LTO, the LTO Logo and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication provides advice and technical information about optimizing and tuning application code to run on systems that are based on the IBM POWER7® and POWER7+™ processors. This advice is drawn from application optimization efforts across many different types of code that runs under the IBM AIX® and Linux operating systems, focusing on the more pervasive performance opportunities that are identified, and how to capitalize on them. The technical information was developed by a set of domain experts at IBM.

The focus of this book is to gather the right technical information, and lay out simple guidance for optimizing code performance on the IBM POWER7 and POWER7+ systems that run the AIX or Linux operating systems. This book contains a large amount of straightforward performance optimization that can be performed with minimal effort and without previous experience or in-depth knowledge. This optimization work can:

- ▶ Improve the performance of the application that is being optimized for the POWER7 system
- ▶ Carry over improvements to systems that are based on related processor chips
- ▶ Improve performance on other platforms

The audience of this book is those personnel who are responsible for performing migration and implementation activities on IBM POWER7-based servers, which includes system administrators, system architects, network administrators, information architects, and database administrators (DBAs).

The team who wrote this book

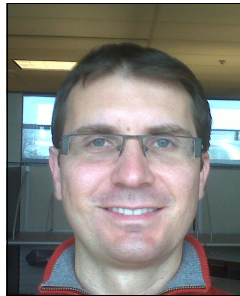
This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Brian Hall is the lead analyst for Power performance improvement efforts with the IBM Software Group Next Generation Systems team. He works with many IBM software products to capitalize on the IBM Power Architecture® and develop performance preferred practices for software development and deployment. After joining IBM in 1987, Brian originally worked on the IBM XL C/C++/Fortran compilers and on the just-in-time compiler for IBM Java on Power. He has a Bachelor's degree in Computer Science from Queen's University at Kingston and a Master's degree in Computer Science from the University of Toronto.

Mala Anand is a Senior Technical Staff Member with the System Performance team in the IBM Systems and Technology Group. She holds a Master's degree in Computer Science from the University of Houston. Her areas of expertise include system performance and architecture, with a focus on hypervisors, operating systems, networks, and middleware on AIX and Linux. She has published papers on network, middleware, and hypervisor performance.



Bill Buros is a Senior Technical Staff Member in the IBM Linux Technology Center. He is a performance lead on STG IBM Power Systems™ for the Linux operating systems. His focus is on the performance tools, workloads, benchmarks, systems, and teams around the world for IBM PowerLinux™. He has a Bachelor's degree in Computer Science from California Polytechnic State University in San Luis Obispo, CA. He is one of the sponsoring leaders of the IBM PowerLinux technical community on the IBM developerWorks® website and has written extensively on the many aspects of performance tuning and optimizations.



Miso Cilindzic has been with IBM since 2000. Over the years, he has worked on a diverse set of projects, with a focus on IBM DB2® in the areas of performance, and the integration with, and exploitation of, hardware and design of workload optimized systems. Currently, Miso manages the DB2 Performance Benchmarking team.



Hong Hua is a Senior Technical Staff Member with IBM STG Systems Solution Development. Her responsibilities include end-to-end system and software design performance, from application, middleware, and operating system to hardware. Her fields of expertise are Java, IBM PowerPC/AIX system performance analysis, and optimization.



Judy Liu is a Performance Analyst for DB2 with the IBM Toronto Lab. She has six years of experience with DB2 family products, and has been working with DB2 and IBM WebSphere® projects for the past five years. Her areas of expertise include DB2 online transaction processing (OLTP) and producing world-record benchmark results for DB2 and WebSphere. Judy holds a degree in Computing from Queen's University.



John MacMillan is the Technical Lead for the IBM Rational® Developer for IBM Power Systems Software™ Performance Advisor product. John has over 20 years of experience with C/C++ Development Tools and performance analysis.



Sudhir Maddali is a Senior Software Engineer working in AIX Storage Device Driver Development in Hyderabad, India. He has 12 years of experience, mainly working with AIX in the storage device drivers area. Sudhir's areas of expertise include storage device drivers and protocols, such as SCSI, SAS, and USB 2.0/3.0.



K Madhusudanan is an IBM Master Inventor and the Technical Chief Engineering Manager (TCEM) for AIX user space architecture with IBM. Madhu has experience with the AIX malloc subsystem, AIX libraries, debugger, and AIX Clustering Technology. He joined IBM in August, 2003, and has 12 years of industry experience. He holds a Bachelor's degree in Computer Science Engineering from Bharathiar University, Coimbatore - India and a Master's degree in Software Systems from Birla Institute of Technology, Pilani-India.



Bruce Mealey is a Distinguished Engineer in the US. He has 24 years of experience in AIX development. Bruce holds a Bachelor's degree in Electrical Engineering from the University of Texas.



Steve Munroe is a Senior Technical Staff Member at the Rochester, Minnesota Lab in IBM US. He has 38 years of experience in the software development field. He holds a Bachelor's degree in Computer Science from Washington State University (1974). His areas of expertise include compilers, POSIX run times, PowerISA, and performance analysis. He has written extensively about IBM POWER® performance and Java performance.

Francis P O'Connell is a member of the IBM Systems and Technology Group in Austin, Texas. He is a Senior Technical Staff Member in Power Systems development, specializing in performance. He joined IBM in 1981 after receiving a Bachelor's degree in Mechanical Engineering from the University of Connecticut and then earned a Master's degree in Engineering-Economic Systems from Stanford University.

Sergio Reyes is a Senior Engineer in the US. He has 12 years of experience in performance, specifically with virtualization technologies. He has a Bachelor's degree in Computer Science from the University of Texas - Pan American. Sergio's areas of expertise include the AIX operating system, disk I/O subsystem, and IBM PowerVM® technology.



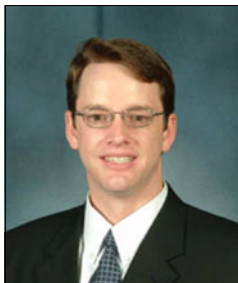
Raul Silvera is a Senior Technical Staff Member (STSM) at the IBM Canada Lab in Toronto. He is a graduate of the Master of Science program at the School of Computer Science of McGill University. He joined IBM in 1997 and has been focused on the development of compilation technology for the IBM Power and System z® platforms, including code analysis, optimization, parallelization, and code generation for C, C++, Fortran, and other static languages.



Randy Swanberg is a Distinguished Engineer in IBM Power Systems Software and is one of the principal AIX architects. After beginning his career working on defense navigation systems for the US Army, he joined IBM in 1989 and worked on several operating system projects, including AIX, OSF, Project Monterey, and Linux. Most of his 23 years with IBM have been working with the AIX core kernel, specializing in processor and hardware bring-up, memory management, and virtualization. Randy has architectural responsibility for AIX support and usage of hardware systems, features, and new technologies. He is also a member of the POWER processor Architecture Control Board.



Brian Twichell is a Senior Technical Staff Member at the IBM development lab in Austin, Texas. He has over 25 years of experience in systems performance, including over 20 years with Power Systems, where he contributed in roles spanning research, development, and pre-sales and post-sales support. He holds Bachelor's and Master's degrees in Computer Science from Purdue University and the University of Texas at Austin, respectively.



Brian F Veale is a Senior Software Engineer in Power Systems Software and an AIX architect. He holds a PhD in Computer Science from the University of Oklahoma, where he was a Graduate Assistance in Areas of National Need (GAANN) Fellow, and a lecturer, teaching courses in Computer Science and Electrical and Computer Engineering. Before obtaining his PhD, Brian worked on training simulators for the US Air Force. At IBM, he works on the AIX core kernel, specializing in support and usage of new processor features and hardware systems. He is a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE) and a member of the POWER processor Architecture Control Board.



Julian Wang is the technical lead of JIT compiler and Java performance on Power Systems, and has been developing compiler and runtime products for the past 17 years. Julian has a passion for making Java perform better on the Power Architecture, and he has acute interests in parallel computing, operating systems, performance analysis, and bit-twiddling.

Yaakov Yaari is a research scientist at Haifa Research Lab, Israel. He has 30 years of experience in program simulation, computer architecture, and in the compiler and program optimization fields. He holds a Ph.D. degree in Mathematics from Bar Ilan University, and Bachelor of Science and Master of Science degrees in Electrical Engineering from Technion, Haifa. His areas of expertise include program optimization, machine learning, and computer architecture. He has published several papers in these areas and holds several patents.

Thanks to the following people for their contributions to this project:

- ▶ Matthew Accapadi, F.A.S.T., AIX performance, Austin, Texas
- ▶ Peter Barnett, Consulting/T Specialist--pSeries Wall Street, New York, New York
- ▶ Scott A Carroll, HMC, Power Linux & FPGA Program Manager, Austin, Texas
- ▶ Grover Davidson, Development Support/Performance Analysis, Austin, Texas
- ▶ Surya Duggirala, Lead WebSphere Performance Architect, Rochester, Minnesota
- ▶ John Finley, IBM STG Power Systems Client Care, Poughkeepsie, New York
- ▶ Jerrold M (Jerry) Heyman, Software Engineer/Technical Consultant - IBM System p/AIX, Research Triangle Park, North Carolina
- ▶ Karen Lawrence, IBM Redbooks Technical Writer, Research Triangle Park, North Carolina
- ▶ Mirco Malessa, SAP on POWER Development - IBM i, IBM PureSystems™, Boeblingen Germany
- ▶ Anbazhagan Mani, Senior Software Engineer, Solutions Management Architecture, Bangalore, India
- ▶ Colin Parris, General Manager, Power Systems, Somers, New York
- ▶ Nashib Qadri, Program Manager - Hardware Acceleration Lab, Ontario, Canada
- ▶ Linda Robinson, IBM Redbooks Graphics Specialist, Research Triangle Park, North Carolina
- ▶ Sanjay Ruprell, IT Specialist, Foster City, California
- ▶ Bruce P Semple, STG Power and z Systems Architecture, Gaithersburg, Maryland
- ▶ Maneesh Sharma, POWER Sales Representative, Englewood Cliffs, New Jersey
- ▶ Wolfgang Tertel, Oracle Onsite Redwood Shores, Redwood City, California
- ▶ Scott Vetter, Executive Project Manager - ITSO IBM Redbooks Systems Team Lead, Austin, Texas
- ▶ Steven W White, Distinguished Engineer, STG Chief Compiler Architect, Austin, Texas
- ▶ The IBM Redbooks editing team, San Jose, California

We also thank the many other people who participated in the preparation of this book.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your s about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Optimization and tuning on IBM POWER7 and IBM POWER7+

This chapter describes the optimization and tuning of the IBM POWER7 system. It covers the the following topics:

- ▶ Introduction
- ▶ Outline of this guide
- ▶ Conventions that are used in this guide
- ▶ Background
- ▶ Optimizing performance on POWER7

1.1 Introduction

The focus of this publication is about gathering the correct technical information, and laying out simple guidance for optimizing code performance on IBM POWER7 and POWER7+ systems that run the AIX or Linux operating systems. There is much straightforward performance optimization that can be performed with a minimum of effort and without extensive previous experience or in-depth knowledge. This optimization work can:

- ▶ Substantially improve the performance of the application that is being optimized for the POWER7 system
- ▶ Typically carry over improvements to systems that are based on related processor chips
- ▶ Improve performance on other platforms

The new POWER7+ processor offers higher clock rates and larger caches than the original POWER7 processor, along with some new features, but is otherwise similar to the POWER7 processor. Most of the technical information, and all of the guidance for optimizing performance on POWER7, also applies to POWER7+. Any differences in POWER7+ and any new features in that chip are specifically noted; you can otherwise presume that any information presented for POWER7 also applies to POWER7+.

This guide strives to focus on optimizations that tend to be positive across a broad set of IBM POWER processor chips and systems. While specific guidance is given for the POWER7 and POWER7+ processors, the general guidance is applicable to the IBM POWER6®, POWER5, and even to earlier processors.

This guide is directed at personnel who are responsible for performing that migration and implementation activities on IBM POWER7-based servers, which includes system administrators, system architects, network administrators, information architects, and database administrators (DBAs).

1.2 Outline of this guide

The first section of this guide lays out simple strategies for optimizing performance (see 1.5, “Optimizing performance on POWER7” on page 5). We describe a set of straightforward steps to set up the environment for performance tuning and optimization, followed by an explanation about how to perform these easy and investigative steps. These steps are the most valuable areas on which to focus.

Next, this guide describes deployment, that is, system setup and configuration choices that ensure good performance on POWER7. Together, these simple optimization strategies and deployment guidance satisfy the requirements for most environment and can deliver substantial improvements.

Finally, this guide describes some of the more advanced investigative techniques that can be used to identify performance bottlenecks in an application. It is here that optimization efforts move into the code internals of an application, and improvements are typically made by modifying source code. Of necessity, coverage in this last area is fairly rudimentary, focusing on general areas of investigation and the tooling that you should use.

Most of the remaining material in this guide is technical information that was developed by domain experts at IBM:

- ▶ We provide hardware information about the POWER7 and POWER7+ processors (see Chapter 2, “The POWER7 processor” on page 21), highlighting the important features from a performance perspective and laying out the basic information that is drawn upon by the material that follows.
- ▶ Next, we describe the system software stack, examining the IBM POWER Hypervisor™ (see Chapter 3, “The POWER Hypervisor” on page 55), the AIX and Linux operating systems and system libraries (see Chapter 4, “AIX” on page 67 and Chapter 5, “Linux” on page 97), and the compilers (see Chapter 6, “Compilers and optimization tools for C, C++, and Fortran” on page 107). Java (see Chapter 7, “Java” on page 125) also receives extensive coverage.
- ▶ In Chapter 4, “AIX” on page 67, we examine a set of operating system-specific optimization opportunities. Then, we highlight some of the areas in which AIX uses some new features of the POWER7+ processor. Next, we provide detailed coverage about the new Dynamic System Optimizer (DSO) feature, building on previous work on the Active System Optimizer (ASO). ASO or DSO can be enabled to autonomously use some of the hardware- or operating system-specific optimizations that are described in this guide. The chapter concludes with a short description of AIX preferred practices regarding system setup and maintenance.
- ▶ In Chapter 5, “Linux” on page 97, we describe the two primary Linux operating systems that are used on POWER7: Red Hat Enterprise Linux (RHEL) for POWER and SUSE Linux Enterprise Server (SLES) for POWER. The minimum supported levels of Linux, including service pack (SP) levels, are SLES11/SP1 and RHEL6/GA, which provide full support and usage of POWER7 technologies and systems. Linux is based on community efforts that are focused not only on the Linux kernel, but also all of the complementary packages, tools, toolchains, and GNU Compiler Collection (GCC) compilers that are needed to effectively use POWER7. IBM provides the expertise for Power Systems by developing, optimizing, and pushing open source changes to the Linux communities.
- ▶ Finally, we cover important information about IBM middleware, DB2 (see Chapter 8, “DB2” on page 137) and IBM WebSphere Application Server (see Chapter 9, “WebSphere Application Server” on page 147). Various applications use middleware, and it is critical that the middleware is correctly tuned and performs well. The middleware chapters cover how these products are optimized for POWER7, including select preferred practices for tuning and deploying these products.

Three appendixes are included:

- ▶ Appendix A, “Analyzing malloc usage under AIX” on page 151 explains some simple techniques for analyzing how an application is using the system memory allocation routines (*malloc* and related functions in the C library). *malloc* is often a bottleneck for application performance, especially under AIX. AIX has an extensive set of optimized *malloc* implementations, and it is easy to switch between them without rebuilding or changing an application. Knowing how an application uses *malloc* is key to choosing the best memory allocation alternatives that AIX offers. Even Java applications often make extensive use of *malloc*, either in Java Native Interface (JNI) code that is part of the application itself or in the Java class libraries, or in binary code that is part of the software development kit (SDK).
- ▶ Appendix B, “Performance tooling and empirical performance analysis” on page 155 describes some of the important performance tools that are available on the IBM Power Architecture under AIX or Linux, and strategies for using them in empirical performance analysis efforts.

These performance tools are most often used as part of the advanced investigative techniques that are described in 1.5, “Optimizing performance on POWER7” on page 5, except for the new performance advisors, which are intended as investigative tools, appropriate for a broader audience of users.

- ▶ Appendix C, “POWER7 optimization and tuning with third-party applications” on page 185 describes preferred practices for migrating third-party products (Oracle, Sybase ASE and Sybase IQ, SAS, and SAP) to Power Systems. We describe implementation tasks and how to meet environment requirements. Links to comprehensive documentation about current migrations are included.

After you review the advice in this guide, and if you would like more information, begin by visiting the IBM Power Systems website at:

<http://www.ibm.com/systems/power/index.html>

1.3 Conventions that are used in this guide

In this guide, our convention for indicating sections of code or command examples is shown in Table 1-1.

Table 1-1 Conventions that are used in this guide

Type of example	Format that is used in this guide	Example of our convention
Commands and command options within text	Monofont, bolded	l<code>dedit</code>
Command lines or code examples outside of text	Monofont	<code>l<code>dedit -btextpsize=64k -bdatapsize=64k -bstackpsize=64k</code></code>
Variables in command lines	Monofont, italicized	<code>l<code>dedit -btextpsize=64k -bdatapsize=64k -bstackpsize=64k <executable></code></code>
Variables that are limited to specific choices	Monofont, italicized	<code>-mcmode1={<i>medium</i> <i>large</i>}</code>

1.4 Background

Some trends in processor design are making it more important than ever to consider analyzing and working to improve application performance. In the past, two of the ways in which newer processor chips delivered higher performance were by:

- ▶ Increasing the clock rate
- ▶ Making microarchitectural improvements that increase the performance of a single thread

Often, upgrading to a new processor chip gave existing applications a 50% or possibly 100% performance improvement, leaving little incentive to spend much effort to get an uncertain amount of additional performance. However, the approach in the industry has shifted, so that the newer processor chips do not substantially increased clock rates, as compared to the previous generation. In some cases, clock rates declined in newer designs. Recent designs also generally offer more modest improvements in the performance of a single execution thread.

Instead, the focus has shifted to delivering multiple cores per processor chip, and to delivering more hardware threads in each core (known as simultaneous multi-threading (SMT), in IBM Power Architecture terminology). This situation means that some of the best opportunities for improving the performance of an application are in delivering scalable code by having an application make effective use of multiple concurrent threads of execution.

Coupled with the trend toward aggressive multi-core and multi-threaded designs, there are changes in the amount of cache and memory bandwidth available to each hardware thread. Cache sizes and chip-level bandwidth are increasing at a slower rate than the growth of hardware threads, meaning that the amount of cache per thread is not growing as rapidly. In some cases, it decreases from one generation to the next. Again, this situation shows where deeper analysis and performance optimization efforts can provide some benefits.

1.5 Optimizing performance on POWER7

This section provides guidance for optimizing code performance on POWER7 when you use the AIX or Linux operating systems. The POWER7+ processor is a superset of the POWER7 processor, so all optimizations described for POWER7 apply equally for POWER7+. We cover the more prominent performance opportunities that are noted in past optimization efforts. The guidance is organized in to three broad categories:

1. Lightweight tuning and optimization guidelines

Lightweight tuning covers simple prescriptive steps for tuning application performance on POWER7. These simple steps can be carried out without detailed knowledge of the internals of the application that is being optimized and usually without modifying the application source code. Simple system utilization and performance tools are used for understanding and improving your application performance. The steps and tools are general guidelines that apply to all types of applications. Although they are simple and straightforward, they often lead to significant performance improvements. It is possible to accomplish these steps in as little as two days or so for a small application, or, at most, two weeks for a large and complex application.

Performance improvement: Consider lightweight tuning to be the starting point for any performance improvement effort.

2. Deployment guidelines

Deployment guidelines cover tuning considerations that are related to the:

- Configuration of a POWER7 system to deliver the best performance
- Associated runtime configuration of the application itself

There are many choices in a deployment, some of which are unrelated to the performance of a particular application, and so, at best, this section can present some guidelines and preferred practices. Understanding logical partitions (LPARs), energy management, I/O configurations, and using multi-threaded cores are examples of typical system considerations that can impact application performance.

Performance improvement: Consider deployment guidelines to be the second required activity for any reasonably extensive performance effort.

3. Deep performance optimization guidelines

Deep performance analysis covers performance tools and general strategies for identifying and fixing application bottlenecks. This type of analysis requires more familiarity with performance tooling and analysis techniques, sometimes requiring a deeper understanding of the application internals, and often requiring a more dedicated and lengthy effort. Often, a simpler analysis is all that is required to identify serious bottlenecks in an application, but a more detailed investigation is required to perform an exhaustive search for all of the opportunities for increasing performance.

Performance improvement: Consider this the last activity that is undertaken, with simpler analysis steps, for a moderately serious performance effort. The more complex iterative analysis is reserved for only the most performance critical applications.

This section provides only minimal background on the guidance provided. Detailed material about these topics is incorporated in the chapters that follow and in the appendixes. The following chapters and appendixes also cover many other performance topics that are not addressed here.

Guidance for POWER7 and POWER7+: The guidance that is provided in this book specifically applies to POWER7 and POWER7+ processor chips and systems. The POWER7+ processor is a superset of the POWER7 processor, so all optimizations described for POWER7 apply equally to POWER7+. The guidance that is provided also generally applies to previous generations of POWER processor chips and systems, including POWER5 and POWER6. When our guidance is not applicable to all generations of Power Systems, we note that for you.

1.5.1 Lightweight tuning and optimization guidelines

This section covers building and performance testing applications on POWER7, and gives a brief introduction to the most important simple performance tuning opportunities that are identified for POWER7. More details about these and other opportunities are presented in the later chapters of this guide.

Performance test beds and workloads

In performance work, when you are tuning and optimizing an application for a particular processor, you must run and measure performance levels on that processor. Although there are some characteristics that are shared among processor chips in the same family, each generation of processor chip has unique performance characteristics. Optimizing code for POWER7 requires that you set up a test bed on a POWER7 system.

The POWER7+ processor has higher clock speeds and larger caches than POWER7, and applications should see higher performance on that new chip. Aside from those differences, the performance characteristics of the two chips are the same. A performance effort specifically targeting POWER7+ should use a POWER7+ system, but otherwise a POWER7 system can be used.

You want to see good performance across a range of newer systems, with a special emphasis on optimizing for the latest design. For Power Systems, the previous POWER6 generation is still commonly used. For this reason, it is best to have multiple test bed environments: a POWER7 system for most optimization work and a POWER6 system for limited testing to ensure that all tuning is beneficial on the previous generation of hardware.

POWER7 and POWER6 processors are dissimilar in some respects, and some simple steps can be taken to ensure good performance of a single binary running on either system. In particular, see the information in “C, C++, and Fortran compiler options” on page 8.

Performance test beds must be sized and configured for performance and scalability testing. Choose your scalability goals based on the requirements that are placed on an application, and the test bed must accommodate at least the minimum requirements. For example, when you target a multi-threaded application to scale up to four cores on POWER7, it is important that the test bed be at least a 4-core system and that tests are configured to run in various configurations (1-core, 2-core, and 4-core). You want to be able to measure performance across the different configurations and the scalability can be computed. Ideally, a 4-core system delivers four times the performance of a 1-core system, but in practice, the scalability is generally less than ideal. Scalability bottlenecks might not be clearly visible if the only testing done for this example were in a 4-core configuration.

With the multi-threaded POWER7 cores (see 2.2, “Multi-core and multi-thread scalability” on page 23), each processor core can be instantiated with one, two, or four logical CPUs within the operating system, so a 4-core server, with SMT4 mode (four hardware threads per core), means that the operating system is running 16 logical CPUs. Also, larger-core servers are becoming more pervasive, with scaling considerations well beyond 4-core servers.

The performance test bed must be a dedicated logical partition (LPAR). You must ensure that there is no other activity on the system (including on other LPARs, if any, configured on the system) when performance tests are run. Performance testing initially should be done in a non-virtualized environment to minimize the factors that affect performance. Ensure that the LPAR is running an up-to-date version of the operating system, at the level that is expected for the typical usage of the application. Keep the test bed in place after any performance effort so that performance can occasionally be monitored, which ensures that later maintenance of an application does not introduce a performance regression.

Choosing the appropriate workloads for performance work is also important. Ideally, a workload has the following characteristics:

- ▶ Be representative of the expected actual usage of the application.
- ▶ Have simple measures of performance that are easily collected and compared, such as run time or transactions/second.
- ▶ Be easy to set up and run in an automated environment, with a fairly short run time for a fast turnaround in performance experiments.
- ▶ Have a low run-to-run variability across duplicated runs, such that extensive tests are not required to obtain a statistically significant measure of performance.
- ▶ Produce a result that is easily tested for correctness.

When an application is being optimized for both the AIX and Linux operating systems, much of the performance work can be undertaken on just one of the operating systems. However, some performance characteristics are operating system-dependent, so some analysis must be performed on both operating systems. In particular, perform profiling and lock analysis separately for both operating systems to account for differences in system libraries and kernels. Each operating system also has unique scalability considerations. More operating system-specific optimizations are detailed in Chapter 4, “AIX” on page 67 and Chapter 5, “Linux” on page 97.

Build environment and build tools

The build environment, if separate from the performance test bed, must be running an up-to-date operating system. Only recent operating system levels include Application Binary Interface (ABI) extensions to use or control newer hardware features.

Critically, all compilers that are used to build an application should be up-to-date versions that offer full support for the target processor chip. Older levels of a compiler might tolerate newer processor chips, but they do not capitalize on the unique features of the latest processor chips. For the IBM XL compilers on AIX or Linux, XLC11 and XLF13 are the first compiler versions that have processor-specific tuning for POWER7. The new XLC12 compiler is generally recommended. For the GCC compiler on Linux, the IBM Advance Toolchain 4.0 and 5.0 versions contain an updated GCC compiler that is recommended for POWER7. The IBM XL Fortran Compiler is generally recommended over gfortran for the most optimized high floating point performance characteristics. Compiler levels do not change for POWER7+.

For the GCC compiler on Linux, the GCC compilers that come with RHEL and SLES recognize and take advantage of the Power Architecture and optimizations. For improved optimizations and newer GCC technology, the IBM Advance Toolchain package provides an updated GCC compiler and optimized toolchain libraries that you should use with POWER7.

The Advance Toolchain is a key performance technology available for Power Systems running Linux. It includes newer, Power optimized versions of compilers (GCC, G++, and gfortran), utilities, and libraries, along with various performance tools. The full Advance Toolchain must be installed in the build environment, and the Advance Toolchain runtime package must be installed in the performance test bed. The Toolchain is designed to coexist with the GCC compilers and toolchain that are provided in the standard Linux distributions. More information is available in “GCC, toolchain, and IBM Advance Toolchain” on page 98.

Along with the compilers for C/C++ and Fortran, there is the separate IBM Feedback Directed Program Restructuring (FDPR®) tool to optimize performance. FDPR takes a post-link executable image (such as one produced by static compilers) and applies additional optimizations. FDPR is another tool that can be considered for optimizing applications that are based on an executable image. More details can be found in 6.3, “IBM Feedback Directed Program Restructuring” on page 114.

Java also contains a dynamic Just-In-Time (JIT) compiler, and only newer versions are tuned for POWER7 or POWER7+. However, Java compilations to binary code take place at application execution time, so a newer Java release must be installed on the performance test bed system. For IBM Java, tuning for POWER7 was introduced in Java 6 SR7, and that is the recommended minimum version. Newer versions of Java contain more improvements, though, as described in 7.1, “Java levels” on page 126. Java 7 is generally the preferred version to use for POWER7 or POWER7+ because of improvements over previous versions.

C, C++, and Fortran compiler options

For the static compilers, the important compilation options to consider are as follows:

- ▶ Basic optimization options: With the IBM XL compilers, the minimum suggested optimization level to use is `-O`, while for GCC it is `-O3`. Higher levels of optimization are better for some types of code, and you should experiment with them. The XL compilers also have optimization options, such as `-qhot`, that can be evaluated. More options are detailed in Chapter 6, “Compilers and optimization tools for C, C++, and Fortran” on page 107. The more aggressive optimization options might not work for all programs and might need to be coupled with the strict options described in this list (see page 9).
- ▶ Target processor chip options: It is possible to build a single executable that runs on various POWER processors. However, that executable does not take advantage of some of the features added to later processor chips, such as new instructions. If only a restricted range of newer processor chips must be supported, consider using the compilation options that enable the usage of newer features. With the XL compilers, for example, if the executable must run only on POWER7 processors, the `-qarch=pwr7` option can be specified. The equivalent GCC option is `-mcpu=power7`.

- ▶ Target processor chip tuning options: The XL compiler `-qtune` option specifies that the code produced must be tuned to run optimally on particular processor chips. The executable that is produced still runs on other processor chips, but might not be tuned for them. Some of the possible options to consider are:
 - `-qarch=ppc64 -qtune=pwr7` for an executable that is optimized to run on POWER7, but that can run on all 64-bit implementations of the Power Architecture (POWER7, POWER6, POWER5, and so on)
 - `-qtune=balanced` for an executable that must run well across both POWER6 and IBM POWER7 Systems™
 - `-mtune=power7` to tune for POWER7 on GCC
- ▶ Strict options: Sometimes the compilers can produce faster code by subtly altering the semantics of the original source code. An example of this scenario is expression reorganization. Especially for floating point code, the effect of expression reorganization can produce different results. For some applications, these optimizations must be prevented to achieve valid results. For the XL compilers, certain semantic-altering transformations are allowed by default at higher optimization levels, such as `-O3`, but those transformations can be disabled by using the `-qstrict` option (for example, `-O3 -qstrict`). For GCC, the default is strict mode, but you can use `-ffast-math` to enable optimizations that are not concerned with Not a Number (NaN), signed zeros, infinities, floating point expression reorganization, or setting the `errno` variable. The new `-Ofast` GCC option includes `-O3` and `-ffast-math`, and might include other options in the future.
- ▶ Source code compatibility options: The XL compilers assume that the C and C++ source code conforms to language rules for aliasing. On occasion, older source code fails when compiled with optimization, because the code violates the language rules. A workaround for this situation is to use the `-qalias=noansi` option. The GCC workaround is the `-fno-strict-aliasing` option.
- ▶ Profile Directed Feedback (PDF): PDF is an advanced optimization feature of the compilers that you should consider for performance critical applications.
- ▶ Interprocedural Analysis (IPA): IPA is an advanced optimization feature of the compilers that you should consider for performance critical applications.

Compiler options: Compiler options do not change for POWER7+.

A simple way to experiment with the C, C++, and Fortran compilation options is to repeatedly build an application with different option combinations, and then to run it and measure performance to see the effect. If higher optimization levels produce invalid results, try adding one or both of the `-qstrict` and `-qalias` options with the XL compilers, or `-fno-strict-aliasing` with GCC.

Not all source files must be compiled with the same set of options, but *all* files must be compiled at the minimum optimization level. There are cases where optimization was not used on just one or two important source files, and that caused an application to suffer from reduced performance.

Java options

Many Java applications are performance sensitive to the configuration of the Java heap and garbage collection (GC). Experimentation with different heap sizes and GC policies is an important first optimization step. For generational GC, consider using the options that specify the split between nursery space (also known as the *new* or *young* space) and tenured space (also known as the *old* space). Most Java applications have modest requirements for long-lived objects in the tenured space, but frequently allocate new objects with a short life span in the nursery space.

If 64-bit Java is used, use the `-Xcompressedrefs` option.

By default, recent releases of Java use 64 KB medium pages for the Java heap, which is the equivalent of explicitly specifying the `-X1p64k` option. If older releases of Java are used, we strongly encourage using the `-X1p64k` option. Otherwise, those releases default to using 4 KB pages. Often, there is some additional performance improvement that is seen in using larger 16 MB large pages by using the `-X1p16m` option. However, using 16 MB pages normally requires explicit configuration by the administrator of the AIX or Linux operating system to reserve a portion of the memory to be used exclusively for large pages. (For more information, see 7.3.2, “Configuring large pages for Java heap and code cache” on page 128.) As such, the medium pages are a better choice for general use, and the large pages can be considered for performance critical applications. The new Dynamic System Optimizer (DSO) facility in AIX (see 4.2, “AIX Active System Optimizer and Dynamic System Optimizer” on page 84) autonomously takes advantage of 16 MB pages without administrator configuration. This facility might be appropriate for cases where a very large Java heap is being used.

On Power Systems, the `-Xcodecache` option often delivers a small improvement in performance, especially in a large Java application. This option specifies the size of each code cache that is allocated by the JIT compiler for the binary code that is generated for Java methods. Ideally, all of the compiled Java method binary code fits into a single code cache, eliminating the small penalty that occurs when one Java method calls another method when the binary code for the two methods is in different code caches. To use this option, determine how much code space is being used, and then set the size of the option correctly. The maximum size of each code cache that is allocated is 32 MB, so the largest value that can be used for this option is `-Xcodecache32m`. For more information, see 7.3.5, “JIT code cache” on page 129.

The JIT compiler automatically uses an appropriate optimization level when it compiles Java methods, and recent Java releases automatically fully utilize all of the new features of the target POWER7 or POWER7+ processor of the system an application is running on.

For more information about Java performance, see Chapter 7, “Java” on page 125.

Optimized libraries

Optimized libraries are important for application performance. This section covers some considerations that are related to standard libraries for AIX or Linux, libraries for Java, or specialized mathematical subroutine libraries that are available for the Power Architecture.

AIX malloc

The AIX operating system offers various memory allocation packages (the standard `malloc()` and related routines in the C library). The default package offers good space efficiency and performance for single-threaded applications, but it is not a good choice for the scalability of multi-threaded applications. Choosing the correct malloc package on AIX is important for performance. Even Java applications can make extensive use of malloc through JNI code or internally in the Java Runtime Environment (JRE).

Fortunately, AIX offers a number of different memory allocation packages that are appropriate for different scenarios. These different packages are chosen by setting environment variables and do not require any code modification or rebuilding of an application.

Choosing the best malloc package requires some understanding of how an application uses the memory allocation routines. Appendix A, “Analyzing malloc usage under AIX” on page 151 shows how to easily collect the required information. Following the data collection, experiment with various alternatives, alone or in combination. Some alternatives that deliver high performance include:

- ▶ Pool malloc: The pool front end to the malloc subsystem optimizes the allocation of memory blocks of 512 bytes or less. It is common for applications to allocate many small blocks, and pools are particularly space- and time-efficient for that allocation pattern. Thread-specific pools are used for multi-threaded applications. The pool malloc is a good choice for both single-threaded and multi-threaded applications.
- ▶ Multiheap malloc: The multiheap malloc package uses up to 32 separate heaps, reducing contention when multiple threads attempt to allocate memory. It is a good choice for multi-threaded applications.

Using the pool front end and multiheap malloc in combination is a good alternative for multi-threaded applications. Small memory block allocations, typically the most common, are handled with high efficiency by the pool front end. Larger allocations are handled with good scalability by the multiheap malloc. A simple example of specifying the pool and multiheap combination is by using the environment variable setting:

```
MALLOCOPTIONS=pool,multiheap
```

For more information malloc alternatives, see Chapter 4, “AIX” on page 67 and “Malloc” on page 68.

Linux Advance Toolchain libraries

The Linux Advance Toolchain contains replacements for various standard system libraries. These replacement libraries are optimized for specific processor chips, including POWER5, POWER6, and POWER7. After you install the Linux Advance Toolchain, the dynamic linker automatically has programs use the library that is optimized for the processor chip type in the system.

The libraries in Linux Advance Toolchain Version 5.0 and later are optimized to use the multi-core facilities in POWER7.

Mathematical Acceleration Subsystem Library and Engineering and Scientific Subroutine Library

The Mathematical Acceleration Subsystem (MASS) library contains both optimized and vectorized versions of some basic mathematical functions and runs on AIX and Linux. The MASS library is included with the XL compilers and is automatically used by the compilers when the `-O3 -qhot=level=1` compilation options are used. The MASS routines can be used automatically with the Advance Toolchain GNU Compiler Collection (GCC) by using the `-mvec1ibabi=mass` option, but the library is not included with the compiler and must be separately installed. Explore the use of MASS for applications that use basic mathematical functions. Good results occur when you use the vector versions of the functions. The MASS routines do not necessarily provide the same precision of results as do standard libraries.

The Engineering and Scientific Subroutine Library (ESSL) contains an extensive set of advanced mathematical functions and runs on AIX and Linux. Avoid having applications write their own versions of functions, such as the Basic Linear Algebra Subprograms (BLAS). Instead, use the Power optimized versions in ESSL.

java/util/concurrent

For Java, all of the standard class libraries are included with the JRE. One package of interest for scalability optimization is `java/util/concurrent`. Some classes in `java/util/concurrent` are more scalable replacements for older classes, such as `java/util/concurrent/ConcurrentHashMap`, which can be used as a replacement for `java/util/Hashtable`. `ConcurrentHashMap` might be slightly less efficient than `Hashtable` when run in smaller system configurations where scalability is not an issue, so there can be trade-offs. Also, switching packages requires a source code change, albeit a simple one.

Tuning to capitalize on hardware performance features

For almost all applications, using 64 KB pages is beneficial for performance. Newer Linux releases (RHEL5, SLES11, and RHEL6) default to 64 KB pages, and AIX defaults to 4 KB pages. Applications on AIX have 64 KB pages that are enabled through one or a combination of the following methods:

1. Using an environment variable setting:

```
LDR_CNTRL=TEXTPSIZE=64K@DATAPSIZE=64K@STACKPSIZE=64K@SHMPSIZE=64K
```

2. Modifying the executable file with:

```
ldedit -btextpsize=64k -bdatapsize=64k -bstacksize=64k <executable>
```

3. Using linker options at build time:

```
cc -btextpsize:64k -bdatapsize:64k -bstacksize:64k ...  
ld -btextpsize:64k -bdatapsize:64k -bstacksize:64k ...
```

All of these mechanisms for enabling 64 KB pages can be safely used when they run on older hardware or operating system levels that do not support 64 KB pages. When the needed support is not in place, the system simply defaults to using 4 KB pages.

As mentioned in “Java options” on page 10, the recent Java releases default to using 64 KB pages. For Java, it is important that the Java heap space uses 64 KB pages, which are enabled by the `-Xlp64k` option in older releases (a minimum Linux level of RHEL5, SLES11, or RHEL6 is required).

Larger 16 MB pages are also supported on the Power Architecture and might provide an additional performance boost when compared to 64 KB pages. However, the usage of 16 MB pages requires explicit configuration by the administrator of the AIX or Linux operating system. The new DSO facility in AIX (see 4.2, “AIX Active System Optimizer and Dynamic System Optimizer” on page 84) autonomously uses 16 MB pages without any administrator configuration, and might be appropriate for cases in which a very large memory space is being used by an application.

For certain types of non-numerical applications, turning off the default POWER7 hardware prefetching improves performance. In specific cases, disabling hardware prefetching is beneficial for Java programs, WebSphere Application Server, and DB2. One way to control hardware prefetching is at the partition level, where prefetching is turned off by running the following commands:

- ▶ AIX: `dscrct1 -n -s 1`
- ▶ Linux: `ppc64_cpu --dscr=1`

Controlling prefetching in this way might not be appropriate if different applications are running in a partition, because some applications might run best with prefetching enabled. There are also mechanisms to control prefetching at the process level.

POWER7 allows not only prefetching to be enabled or disabled, but it also allows the fine-tuning of the prefetch engine. Such fine-tuning is especially beneficial for scientific/engineering and memory-intensive applications.¹ Hardware prefetch tuning is autonomously used when DSO in AIX is enabled.

For more information about hardware prefetching, the DSO facility, and hardware and operating system tuning and usage for optimum performance, see Chapter 2, “The POWER7 processor” on page 21, Chapter 4, “AIX” on page 67, and Chapter 5, “Linux” on page 97.

Scalability considerations

Aside from the scalability considerations already mentioned (such as AIX malloc tuning and java/util/concurrent usage), there is one Linux operating system setting that enhances scalability in some cases: setting `sched_compat_yield` to 1. This task is accomplished by running the following command:

```
sysctl -w kernel.sched_compat_yield=1
```

This setting makes the Completely Fair Scheduler more compatible with earlier versions of Linux. Use this setting for Java environments, such as for WebSphere Application Server. For more information about multiprocessing with the Completely Fair Scheduler, go to:

<http://www.ibm.com/developerworks/linux/library/l-cfs/?ca=dgr-1nxw06CFC4Linux>

1.5.2 Deployment guidelines

This section discusses deployment guidelines as they relate to virtualized and non-virtualized environments, and the effect of partition size and affinity on deployments.

Virtualized versus non-virtualized environments

Virtualization is a powerful technique that is applicable to situations where many applications are consolidated onto a single physical server. This consolidation leads to better usage of hardware and simplified system administration. Virtualization is efficient on the Power Architecture, but it does come with some costs. For example, the Virtual I/O Server (VIOS) partition that is allocated for a virtualized deployment consumes a portion of the hardware resources to support the virtualization. For situations where few business-critical applications must be supported on a server, it might be more appropriate to deploy with non-virtualized resources. This situation is particularly true in cases where the applications have considerable network requirements.

Virtualized environments offer many choices for deployment, such as dedicated or non-dedicated processor cores and memory, IBM Micro-Partitioning® that uses fractions of a physical processor core, and memory compression. These alternatives are explored in Chapter 3, “The POWER Hypervisor” on page 55. When you set up a virtualized deployment, it is important that system administrators have a complete understanding of the trade-offs inherent in the different choices and the performance implications of those choices. Some deployment choices, such as enabling memory compression features, can disable other performance features, such as support for 64 KB memory pages.

¹ <http://dl.acm.org/citation.cfm?id=2370837> (available for purchase or with access to the ACM Digital Library)

The POWER7 processor and affinity performance effects

The IBM POWER7 is the latest processor chip in the IBM Power Systems family. The POWER7 processor chip is available in configurations with four, six, or eight cores per chip, as compared to the POWER5 and POWER6, each of which have two cores per chip. Along with the increased number of cores, the POWER7 processor chip implements SMT4 mode, supporting four hardware threads per core, as compared to the POWER5 and POWER6, which support only two hardware threads per core. Each POWER7 processor core supports running in single-threaded mode with one hardware thread, an SMT2 mode with two hardware threads, or an SMT4 mode with four hardware threads.

Each SMT hardware thread is represented as a logical processor in AIX or Linux. When the operating system runs in SMT4 mode, it has four logical processors for each dedicated POWER7 processor core that is assigned to the partition. To gain full benefit from the throughput improvement of SMT, applications must use all of the SMT threads of the processor cores.

Each POWER7 chip has memory controllers that allow direct access to a portion of the memory DIMMs in the system. Any processor core on any chip in the system can access the memory of the entire system, but it takes longer for an application thread to access the memory that is attached to a remote chip than to access data in the local memory DIMMs.

For more information about the POWER7 hardware, see Chapter 2, “The POWER7 processor” on page 21. This short description provides some background to help understand two important performance issues that are known as *affinity effects*.

Cache affinity

The hardware threads for each core of a POWER7 processor share a core-specific cache space. For multi-threaded applications where different threads are accessing the same data, it can be advantageous to arrange for those threads to run on the same core. By doing so, the shared data remains resident in the core-specific cache space, as opposed to moving between different private cache spaces in the system. This enhanced *cache affinity* can provide more efficient utilization of the cache space in the system and reducing the latency of data references.

Similarly, the multiple cores on a POWER7 processor share a chip-specific cache space. Again, arranging the software threads that are sharing the data to run on the same POWER7 processor (when the partition spans multiple sockets) often allows more efficient utilization of cache space and reduced data reference latencies.

Memory affinity

By default, the POWER Hypervisor attempts to satisfy the memory requirements of a partition using the local memory DIMMs for the processor cores that are allocated to the partition. For larger partitions, however, the partition might contain a mixture of local and remote memory. For an application that is running on a particular core or chip, the application should always use only local memory. This enhanced *memory affinity* reduces the latency of memory accesses.

Partition sizes and affinity

In terms of partition sizes and affinity, this section describes Power dedicated LPARs, shared resource environments, and memory requirements.

Power dedicated LPARs

Dedicated LPAR deployments generally use larger partitions, ranging from just one POWER7 core up to a partition that includes all of the cores and memory in a large symmetric multi-processor (SMP) system. A smaller partition might run a single application, and a larger partition typically runs multiple applications, or multiple instances of a single application. A common example of multiple instances of a single application is in deployments of WebSphere Application Server.

With larger partitions, one of the most important performance considerations is often which cores and memory are allocated to a partition. For partitions of eight cores or less, the POWER Hypervisor attempts to allocate all cores for the partition from a single POWER7 chip and attempts to allocate only memory local to the chip that is used. Those partitions generally and automatically have good cache and memory affinity. However, it might not be possible to obtain resources for each of the LPARs from a single chip.

For example, assume that you have a 32-core system with four chips, each with eight cores. If five partitions are configured, each with six cores, the fifth LPAR would spread across three chips. Start the most important partition first to obtain resources from a single chip. (The order of starting partitions is one consideration in obtaining the best performance for high priority workloads.)

Another example is when the partition sizes are mixed. Here, starting smaller partitions consumes resources that are spread across many chips, resulting in larger partitions that are spread across multiple chips, which might be contained on a chip if the larger partitions are started first. It is a preferred practice to start higher priority partitions first, so that there is a better opportunity for them to obtain good affinity characteristics in their core and memory allocations. The affinity of the cores and memory that is allocated to a partition can be determined by running the AIX `lssrad -va` command. For more information about partition resource allocation and the `lssrad` command, see Chapter 3, “The POWER Hypervisor” on page 55.

For partitions larger than eight cores, the partition always spans more than one chip and has a mixture of local and remote memory. For these larger partitions, it is often useful to manually force good affinity for an application. Manual affinity can be forced by binding applications so that they can run only on particular cores, and by specifying to the operating system that only local memory can be used by the application.

Consider an example where you run four instances of WebSphere Application Server on a partition of 16 cores on a POWER7 system that is running in SMT4 mode. Each instance of WebSphere Application Server would be bound to run on four of the cores of the system. Because each of the cores has four SMT threads, each instance of WebSphere Application Server is bound to 16 logical processors. Good memory and cache affinity on AIX can therefore be ensured by completing the following steps:

1. Set the AIX `MEMORY_AFFINITY` environment variable, typically to the value `MCM`. This setting tells the AIX operating system to use local memory when an application thread requires physical memory to be allocated.
2. Start the four instances of WebSphere Application Server by running the following `execrset` commands, which bind the execution to the specified set of logical processors:
 - `execrset -c 0-15 -m 0 -e <command to start first WebSphere Application Server instance>`
 - `execrset -c 16-31 -m 0 -e <command to start second WebSphere Application Server instance>`

- `execrset -c 32-47 -m 0 -e <command to start third WebSphere Application Server instance>`
- `execrset -c 48-63 -m 0 -e <command to start fourth WebSphere Application Server instance>`

Some important items to understand in this example are:

- ▶ For a particular number of instances and available cores, the most important consideration is that each instance of an application runs only on the cores of one POWER7 processor chip.
- ▶ Memory and logical processor binding is not done independently because doing so can negatively affect performance.
- ▶ The workload must be evenly distributed over WebSphere Application Server processes for the binding to be effective.
- ▶ There is an assumed mapping of logical processors to cores and chips, which is always established at boot time. This mapping can be altered if the SMT mode of the system is changed by running `smtctl -w now`. It is always best to reboot to change the SMT mode of a partition to ensure that the assumed mapping is in place.

Along with the operating system facilities for manually establishing cache and memory affinity, the AIX Active System Optimizer (ASO) can be enabled to autonomously establish enhanced affinity.

For more information about ASO, the `MEMORY_AFFINITY` environment variable, the `execrset` command, and related environment variables and commands, see Chapter 4, “AIX” on page 67.

The same forced affinity can be established on Linux by running `taskset` or `numactl`. For example:

- ▶ `numactl -C 0-15 -l <command to start first WebSphere Application Server instance>`
- ▶ `numactl -C 16-31 -l <command to start second WebSphere Application Server instance>`
- ▶ `numactl -C 32-47 -l <command to start third WebSphere Application Server instance>`
- ▶ `numactl -C 48-63 -l <command to start fourth WebSphere Application Server instance>`

The `-l` option on these `numactl` commands is the equivalent of the AIX `MEMORY_AFFINITY=MCM` environment variable setting.

Even for partitions of eight cores or less, better cache affinity can be established with multiple application instances by using logical processor binding commands. With eight cores or less, the performance effects typically range up to about 10% improvement with binding. For partitions of more than eight cores that span more than one POWER7 processor chip, using manual affinity results in significantly higher performance.

Shared resource environments

Virtualized deployments that share cores among a set of partitions also can use logical processor binding to ensure good affinity within the guest operating system (AIX). However, the real dispatching of physical cores is handled by the underlying host operating system (POWER Hypervisor).

The POWER Hypervisor uses a three-level affinity mechanism in its scheduler to enforce affinity as much as possible. The reason why absolute affinity is not always possible is that partitions can expand and use unused cycles of other LPARs. This process is done using uncapped mode in Power, where the uncapped cycles might not always have affinity. Therefore, binding logical processors that are seen at the operating system level to physical threads seen at the hypervisor level works only in some cases in shared partitions. Achieving a high level of affinity is difficult when multiple partitions share resources from a single pool, especially at high utilization, and when partitions are expanding to use other partition cycles. Therefore, creating large shared processor core pools that span across chips tends to create remote memory accesses. For this reason, it might be less desirable to use larger partitions and large processor core pools where high-level affinity performance is expected.

Virtualized deployments can use Micro-Partitioning, where a partition is allocated a fraction of a core. Micro-Partitioning allow a core allocation as small as 0.1 cores in older firmware levels, and as small as 0.05 cores starting at the 760 firmware level, when coupled with supporting operating system levels. This powerful mechanism provides great flexibility in deployments. However, very small core allocations may be more appropriate for situations in which many virtual machines are often idle. Therefore, active 0.05 core LPARs can use those idle cycles. Also, there is one negative performance effect in deployments with considerably small partitions, in particular with 0.1 or less cores at high system utilization: Java warm-up times can be greatly increased. In a Java execution, the JIT compiler is producing binary code for Java methods dynamically. Steady-state optimal performance is reached after a portion of the Java methods are compiled to binary code. With considerably small partitions, there might be a long warm-up period before reaching steady-state performance, where a 0.05 LPAR cannot get additional cycles from other LPARs because the other LPARs are consuming their cycles. Also, if the workload that is running on this small-size LPAR does not need more than 5% of a processor core capacity, then the performance impact is mitigated.

Memory requirements

For good performance, there should be enough physical memory that is available so that application data does not need to be frequently paged in and out between memory and disk. The physical memory that is allocated to a partition must be enough to satisfy the requirements of the operating system and the applications that are running on the partition.

Java is sensitive to having enough physical memory available to contain the Java heap because Java applications often have frequent GC cycles where large portions of the Java heap are accessed. If portions of the Java heap are paged out to disk by the operating system because of a lack of physical memory, then GC cycles can cause a large amount of disk activity, which is known as *thrashing*.

1.5.3 Deep performance optimization guidelines

Performance tools for AIX and Linux are described in Appendix B, “Performance tooling and empirical performance analysis” on page 155. A deep performance optimization effort typically uses those tools and follows this general strategy:

1. Gather general information about the execution of an application when it is running on a dedicated POWER7 performance system. Important statistics to consider are:
 - The user and system CPU usage of the application: Ideally, a multi-threaded application generates a high overall CPU usage with most of the CPU time in user code. Too high a system CPU usage is generally a sign of a locking bottleneck in the application. Too low an overall usage usually indicates some type of resource bottleneck, such as network or disk. For low CPU usage, look at the number of runnable threads reported by the operating system, and try to ensure that there are as many runnable threads as there are logical processors in the partition.

- The network utilization of the application: Networks can be a bottleneck in execution either because of bandwidth or latency issues. Link aggregation techniques are often used to solve networking issues.
- The disk utilization of the application: High disk I/O issues are increasingly being solved by using solid-state disks (SSDs).

Common operating system tools for gathering this general information include **topas** (AIX), **top** (Linux), **vmstat**, and **netstat**. Detailed CPU usage information is available by running **sar**. This command diagnoses cases where some logical processors are saturated and others are underutilized, an issue that is seen with network interrupt processing on Linux.

2. Collect a time-based profile of the application to see where execution time is concentrated. Some possible areas of concern are:

- Particular user routines or Java methods with a high concentration of execution time. This situation is an indication of a poor coding practice or an inefficient algorithm that is being used in the application itself.
- Particular library routines or Java class library methods with a high concentration of execution time. First, determine whether the hot routine or method is legitimately used to that extent. Look for alternatives or more efficient versions, such as using the optimized libraries in the Linux Advance Toolchain or the vector routines in the MASS library (for more information, see “Mathematical Acceleration Subsystem Library and Engineering and Scientific Subroutine Library” on page 11).
- A concentration of execution time in the pthreads library (see “Java profiling example” on page 178) or in kernel locking (see “AIX kernel locking services” on page 38) routines. This situation is associated with a locking issue. This locking might ultimately arise at the system level (as seen with malloc locking issues on AIX), or at the application level in Java code (associated with synchronized blocks or methods in Java code). The source of locking issues is not always immediately apparent from a profile. For example, with AIX malloc locking issues, the time that is spent in the malloc and free routines might be quite low, with almost all of the impact appearing in kernel locking routines.

The tools for gathering profiles are **tprof** (AIX) and **Oprofile** (Linux) (both tools are described in “Rational Performance Advisor” on page 161). The **curt** tool (see “AIX trace-based analysis tools” on page 165) also provides a breakdown, describing where CPU time is consumed and includes more useful information, such as a system call summary.

3. Where there are indications of a locking issue, collect locking information.

With locking problems, the primary concern is to determine where the locking originates in the application source code. Cases such as AIX malloc locking can be easily solved just by switching to a more scalable memory allocation package through the **MALLOCTYPE** and **MALLOCOPTIONS** environment variables. In this case, you should examine how malloc is used and consider making changes at the source code level. For example, rather than repeatedly allocating many small blocks of memory by calling malloc for each block, the application can allocate an array of blocks and then internally manage the space.

As mentioned in “java/util/concurrent” on page 12, Java locking issues that are associated with some older classes, such as `java/util/Hashtable`, can be easily solved by using `java/util/concurrent/ConcurrentHashMap`.

For Java programs, use *Java Lock Monitor* (see “Java Health Center” on page 177). For non Java programs, use the **sp1at** tool on AIX (see “AIX trace-based analysis tools” on page 165).

4. For Java, the WAIT tool is a powerful, easy-to-use analysis tool that is based on collecting thread state information.

Using the WAIT tool requires installing and running only a data collection shell. The shell collects various information about the Java program execution, the most important of which is a set of javacore files. The javacore files show the state of all of the threads at the time the file was dumped. The collected data is submitted to an online tool using a web browser, and the tool analyzes the data and displays the results with a GUI. The GUI presents information about thread states and has powerful features to drill down to see call chains.

The WAIT tool results combine many of the features of a time-based profile, a lock monitor, and other tools. For Java programs, the WAIT tool might be one of the first analysis tools to consider because of its versatility and ease of use.

For more information about IBM Whole-system Analysis of Idle Time, which is the browser-based (that is, no-install) WAIT tool, go to:

<http://wait.researchlabs.ibm.com>

Guidance about POWER processor chips: The guidance that is provided in this publication generally applies to previous generations of POWER processor chips and systems, including POWER5 and POWER6. When our guidance is not applicable to all generations of Power Systems, we note that for you.



The POWER7 processor

This chapter introduces the POWER7 processor and describes some of the technical details and features of this product. It covers the the following topics:

- ▶ Introduction to the POWER7 processor
- ▶ Multi-core and multi-thread scalability
- ▶ Using POWER7 features

2.1 Introduction to the POWER7 processor

The POWER7 processor is manufactured using the IBM 45 nm Silicon-On-Insulator (SOI) technology. Each chip is 567 mm² and contains 1.2 billion transistors. As shown in Figure 2-1, the chip contains eight cores, each with its own 256 KB L2 and 4 MB L3 (embedded DRAM) cache, two memory controllers, and an interconnection system that connects all components within the chip. The interconnect also extends through module and board technology to other POWER7 processors in addition to DDR3 memory and various I/O devices. The number of memory controllers and cores available for use depends upon the particular POWER7 system.

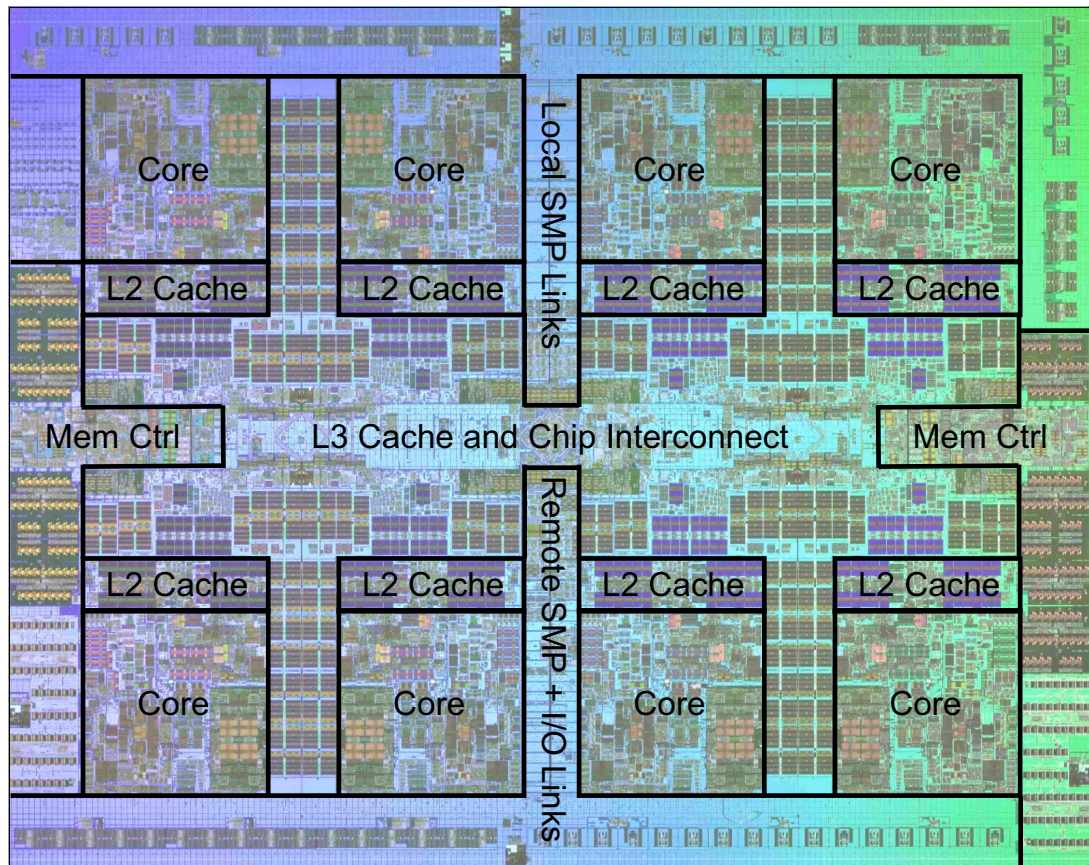


Figure 2-1 The POWER7 processor chip

Each core is a 64-bit implementation of the IBM Power ISA (Version 2.06 Revision B), and has the following features:

- ▶ Multi-threaded design, capable of up to four-way SMT
- ▶ 32 KB, four-way set-associative L1 i-cache
- ▶ 32 KB, eight-way set-associative L1 d-cache
- ▶ 64-entry Effective to Real Address Translation (ERAT) for effective to real address translation for instructions (2-way set associative)
- ▶ 64-entry ERAT for effective to real address translation for data (fully associative)
- ▶ Aggressive branch prediction, using both local and global prediction tables with a selector table to choose the best predictor
- ▶ 15-entry link stack

- ▶ 128-entry count cache
- ▶ 128-entry branch target address cache
- ▶ Aggressive out-of-order execution
- ▶ Two symmetric fixed-point execution units
- ▶ Two symmetric load/store units, which can also run simple fixed-point instructions
- ▶ An integrated, multi-pipeline vector-scalar floating point unit for running both scalar and SIMD-type instructions, including the VMX instruction set and the new Vector Scalar eXtension (VSX) instruction set, and capable of up to eight flops per cycle
- ▶ Hardware data prefetching with 12 independent data streams and software control
- ▶ Hardware DFP capability
- ▶ Adaptive power management

The POWER7 processor is designed for system offerings from 16-core blades to 256-core drawers. It incorporates a dual-scope broadcast coherence protocol over local and global SMP links to provide superior scaling attributes.

For more information about this topic, see 2.4, “Related publications” on page 51.

2.1.1 The POWER7+ processor

The POWER7+ is the same POWER7 processor core with new technology, including more on-chip accelerators and an additional L3 cache. There are no new instructions in POWER7+ over POWER7. The differences in POWER7+ are:

- ▶ Manufactured with 32-nm technology
- ▶ A 10 MB L3 cache per core
- ▶ On-chip encryption accelerators
- ▶ On-chip compression accelerators
- ▶ On-chip random number generators

2.2 Multi-core and multi-thread scalability

POWER7 Systems advancements in multi-core and multi-thread scaling are significant. A significant POWER7 performance opportunity comes from parallelizing workloads to enable the full potential of the Power platform. Application scaling is influenced by both multi-core and multi-thread technology in POWER7 processors. A single POWER7 chip can contain up to eight cores. With SMT, each POWER7 core can present four hardware threads. SMT is the ability of a single physical processor core to simultaneously dispatch instructions from more than one hardware thread context. Because there are multiple hardware threads per physical processor core, additional instructions can run at the same time. SMT is primarily beneficial in commercial environments where the speed of an individual transaction is not as important as the total number of transactions performed. SMT is expected to increase the throughput of workloads with large or frequently changing working sets, such as database servers and web servers.

Additional details about the SMT feature are described in Table 2-1 and Table 2-2.

Table 2-1 Multi-thread per core features by POWER generation

Technology	Cores/system	Maximum SMT mode	Maximum hardware threads per LPAR
IBM POWER4	32	ST	32
IBM POWER5	64	SMT2	128
IBM POWER6	64	SMT2	128
IBM POWER7	256	SMT4	1024

Table 2-2 Multi-thread per core features by single LPAR scaling

Single LPAR scaling	AIX release	Linux
32-core/32-thread	5.3/6.1/7.1	RHEL 5/6 SLES 10/11
64-core/128-thread	5.3/6.1/7.1	RHEL 5/6 SLES 10/11
64-core/256-thread	6.1(TL4)/7.1	RHEL 6 SLES 11sp1
256-core/1024-thread	7.1	RHEL 6 SLES 11sp1

Operating system enablement usage of multi-core and multi-thread technology varies by operating system and release.

Power operating systems present an SMP view of the resources of a partition. Hardware threads are presented as logical CPUs to the application stack. Many applications can use the operating system scheduler to place workloads onto logical processors and maintain the SMP programming model. In some cases, the differentiation between hardware threads per core can be used to improve performance. Placement of a workload on hardware book, drawer and node, socket, core, and thread boundaries can improve application scaling. Details about operating system binding facilities are available in 4.1, “AIX and system libraries” on page 68, and include:

- ▶ Affinity/topology bindings
- ▶ Hybrid thread features

Using multi-core and multi-thread features is a challenging prospect. An overview about this topic and about the application considerations is provided in 4.1, “AIX and system libraries” on page 68. Additionally, the following specific scaling topics are described in 4.1, “AIX and system libraries” on page 68:

- ▶ pthread tuning
- ▶ malloc tuning

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3 Using POWER7 features

This section describes several features of POWER7 that can affect performance, including page sizes, cache sharing, SMT priorities, and others.

2.3.1 Page sizes (4 KB, 64 KB, 16 MB, and 16 GB)

The virtual address space of a program is divided into segments. The size of each segment can be either 256 MB or 1 TB on a Power System. The virtual address space can also consist of a mix of these segment sizes. The segments are again divided into units, called *pages*. Similarly, the physical memory on the system is divided into page size units called *page frames*. The role of the Virtual Memory Manager (VMM) is to manage the allocation of real memory page frames, and to manage virtual memory page references (which is always larger than the available real memory). The VMM must minimize the total processor time, disk bandwidth price, and response time to handle the virtual memory page faults. IBM Power Architecture provides support for multiple virtual memory page sizes, which provides performance benefits to an application because of hardware efficiencies that are associated with larger page sizes.^{1,2}

The POWER5+ and later processor chips support four virtual memory page sizes: 4 KB, 64 KB, 16 MB, and 16 GB. The POWER6 processor also supports using 64 KB pages inside segments along with a base page size of 4 KB.³ The 16 GB pages can be used only within 1 TB segments.

Large pages provide multiple technical advantages:⁴

- ▶ **Reduced Page Faults and Translation Lookaside Buffer (TLB) Misses:** A single large page that is being constantly referenced remains in memory. This feature eliminates the possibility of several small pages often being swapped out.
- ▶ **Unhindered Data Prefetching:** A large page enables unhindered data prefetch (which is constrained by page boundaries).
- ▶ **Increased TLB Reach:** This feature saves space in the TLB by holding one translation entry instead of n entries, which increases the amount of memory that can be accessed by an application without incurring hardware translation delays.
- ▶ **Increased ERAT Reach:** The ERAT on Power is a first level and fully associative translation cache that can go directly from effective to real address. Large pages also improve the efficiency and coverage of this translation cache as well.

Large segments (1 TB) also provide reduced Segment Lookaside Buffer (SLB) misses, and increases the reach of the SLB. The SLB is a cache of the most recently used Effective to Virtual Segment translations.

¹ *Power ISA Version 2.06 Revision B*, available at:

http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

² *What's New in the Server Environment of Power ISA v2.06*, a white paper from Power.org, available at:

<https://www.power.org/documentation/whats-new-in-the-server-environment-of-power-isa-v2-06/>
(registration required)

³ *Multiple page size support*, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

⁴ *What's New in the Server Environment of Power ISA v2.06*, a white paper from Power.org, available at:

<https://www.power.org/documentation/whats-new-in-the-server-environment-of-power-isa-v2-06/>
(registration required)

While 16 MB and 16 GB pages are intended only for particularly high performance environments, 64 KB pages are considered general purpose, and most workloads benefit from using 64 KB pages rather than 4 KB pages.

Multipage size support on Linux

On Power Systems running Linux, the default page size is 64 KB, so most, but not all, applications are expected to see a performance benefit from this default. There are cases in which an application uses many small files, which can mean that each file is loaded into a 64 KB page, resulting in poor memory utilization.

Support for 16 MB pages (*hugepages* in Linux terminology) is available through various mechanisms and is typically used for databases, Java engines, and high-performance computing (HPC) applications. The `libhugetlbfs` package is available in Linux distributions, and using this package gives you the most benefit from 16 MB pages.

Multipage size support on AIX

The `pagesize -a` command on AIX determines all of the page sizes that are supported by AIX on a particular system.

IBM AIX 5L™ Version 5.3 with the 5300-04 Technology Level supports up to four different page sizes, but the actual page sizes that are supported by a particular system vary, based on processor chip type.

AIX V6.1 supports segments with two page sizes: 4 KB and 64 KB. By default, processes use these variable page size segments. This configuration is overridden by the existing page size selection mechanism. Page sizes are an attribute of an individual segment (whether single page size or mixed per segment). A process address space can consist of a mix of segments of varying page sizes. For example, the process text segment can be 4 KB pages, and its stack and heap segments can be 64 KB page size segments.⁵

Because the 64 KB page size is easy to use, and because it is expected that many applications perform better when they use the 64 KB page size rather than the 4 KB page size, AIX has rich support for the 64 KB page size. No system configuration changes are necessary to enable a system to use the 64 KB page size. On systems that support the 64 KB page size, the AIX kernel automatically configures the system to use it. Table 2-3 and Table 2-4 on page 27 list the page size specifications for Power Systems.

Table 2-3 Page size support for Power HW and AIX configuration support⁶

Page size	Required hardware	Requires user configuration	Restricted
4 KB	ALL	No	No
64 KB	POWER5+ or later	No	No
16 MB	POWER4 or later	Yes	Yes
16 GB	POWER5+ or later	Yes	Yes

⁵ Multiple page size support, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

⁶ Ibid

Table 2-4 Supported segment page sizes⁷

Segment base page size	Supported page sizes	Minimum required hardware
4 KB	4 KB/64 KB	POWER6
64 KB	64 KB	POWER5+
16 MB	16 MB	POWER4
16 GB	16 GB	POWER5+

The `vmo` command on AIX allows configuration of the VMM tunable parameters.

The `vmo` tunable `vmm_mpsize_support` toggles the operating system multiple page size support for the extra page sizes that are provided by POWER5+ and later machines.

A value of 1 indicates that the operating system takes advantage of extra page sizes that are supported by a processor chip. A value of 2 indicates that the operating system takes advantage of the capability of using multiple page sizes per segment. When set to 0, the only page size the operating system recognizes are 4 KB and the system large page size.

AIX V6.1 takes advantage of this new hardware capability to combine the conservative memory usage aspects of the 4 KB page size in sparsely referenced memory regions, with the performance benefits of the 64 KB page size in densely referenced memory regions. AIX V6.1 takes advantage of this automatically, without user intervention. This AIX feature is referred to as dynamic Variable Page Size Support (VPSS). Some applications might prefer to use a larger page size, even when a 64 KB region is not fully referenced. The page size promotion aggressiveness factor (PSPA) can be used to reduce the memory-referenced requirement, at which point a group of 4 KB pages is promoted to a 64 KB page size. The PSPA can be set for the whole system by using the `vmm_default_pspa` `vmo` tunable, or for a specific process by using the `vm_pattnr` system call.⁸

In addition to 4 KB and 64 KB page sizes, AIX supports 16 MB pages, also called *large pages*, and 16 GB pages, also called *huge pages*. These page sizes are intended for use only in high-performance environments, and AIX normally does not automatically configure a system to use these page sizes. However, the new Dynamic System Optimizer (DSO) facility in AIX (see 4.2, “AIX Active System Optimizer and Dynamic System Optimizer” on page 84) can autonomously configure and use 16 MB pages when enabled.

Use the `vmo` tunables `lgpg_regions` and `lgpg_size` to configure the number of 16 MB large pages on a system.

The following example allocates 1 GB of 16 MB large pages:

```
vmo -r -o lgpg_regions=64 -o lgpg_size=16777216
```

To use large pages, non-root users must have the `CAP_BYPASS_RAC_VMM` capability in AIX enabled. The system administrator can add this capability by running `chuser`:

```
chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user_id>
```

Huge pages must be configured using the Hardware Management Console (HMC). To do so, complete the following steps:

1. On the managed system, click **Properties** → **Memory** → **Advanced Options** → **Show Details** to change the number of 16 GB pages.
2. Assign 16 GB huge pages to a partition by changing the partition profile.

⁷ Ibid

⁸ Ibid

Application support to use multisize pages on AIX⁹

As described in *Power Instruction Set Architecture Version 2.06*,¹⁰ you can specify page sizes to use for four regions of a 32-bit or 64-bit process address space.

These page sizes can be configured with an environment variable or with settings in an application XCOFF binary with the `ldedit` or `ld` commands, as shown in Table 2-5.

Table 2-5 Page sizes for four regions of a 32-bit or 64-bit process address space

Region	ld or ldedit option	LDR_CNTRL environment variable	Description
Data	<code>bdatapsize</code>	<code>DATAPSIZE</code>	Initialized data, bss, and heap
Stack	<code>bstacksize</code>	<code>STACKSIZE</code>	Initial thread stack
Text	<code>btextpsize</code>	<code>TEXTPSIZE</code>	Main executable text
Shared memory	None	<code>SHMPSIZE</code>	Shared memory that is allocated by the process

You can specify a different page size to use for each of the four regions of a process address space. Only the 4 KB and 64 KB page sizes are supported for all four memory regions. The 16 MB page size is supported only for the process data, process text, and process shared memory regions. The 16 GB page size is supported only for a process shared memory region.

You can set the preferred page sizes for an application in the XCOFF/XCOFF64 binary file by running the `ldedit` or `ld` commands.

The `ld` or `cc` commands can be used to set these page size options when you are linking an executable command:

- ▶ `ld -o mpsize.out -btextpsize:4K -bstacksize:64K sub1.o sub2.o`
- ▶ `cc -o mpsize.out -btextpsize:4K -bstacksize:64K sub1.o sub2.o`

The `ldedit` command can be used to set these page size options in an existing executable command:

```
ldedit -btextpsize=4K -bdatapsize=64K -bstacksize=64K mpsize.out
```

We can set the preferred page sizes of a process with the `LDR_CNTRL` environment variable. As an example, the following command causes the `mpsize.out` process to use 4 KB pages for its data, 64 KB pages for its text, 64 KB pages for its stack, and 64 KB pages for its shared memory on supported hardware:

```
LDR_CNTRL=DATAPSIZE=4K@TEXTPSIZE=64K@SHMPSIZE=64K mpsize.out
```

Page size environment variables override any page size settings in an executable XCOFF header. Also, the `DATAPSIZE` environment variable overrides any `LARGE_PAGE_DATA` environment variable setting.

Rather than using the `LDR_CNTRL` environment variable, consider marking specific executable files to use large pages, because this limits the large page usage to the specific application that benefits from large page usage.

⁹ Ibid

¹⁰ *Power ISA Version 2.06 Revision B*, available at:
http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

Page size and shared memory

To back shared memory segments of an application with large pages, specify the **SHM_LGPAGE** and **SHM_PIN** flags in the **shmget()** function. If large pages are unavailable, the 4 KB pages back the shared memory segment.

Support for specifying the page size to use for the shared memory of a process with the **SHMPSIZE** environment variable is available starting in IBM AIX 5L Version 5.3 with the 5300-08 Technology Level, or later, and AIX Version 6.1 with the 6100-01 Technology Level, or later.

Monitoring page size that is used by an application

Monitoring page size is accomplished by running the following commands:¹¹

- ▶ The **ps** command can be used to monitor the base page sizes that are used for process data, stack, and text.
- ▶ The **vmstat** command has two options available to display memory statistics for a specific page size.
- ▶ The **vmstat -p** command displays global **vmstat** information, along with a breakdown of statistics per page size.
- ▶ The **vmstat -P** command displays per page size statistics.

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3.2 Cache sharing

Power Systems consist of multiple processor cores and multiple processor chips that share caches and memory in the system. The architecture uses a processor and memory layout that you can use to scale the hardware to many nodes of processor chips and memory. One advantage is that systems can be used for multiple workloads and workloads that are large. However, these characteristics must be carefully weighed in the design, implementation, and evaluation of a workload. Aspects of a program, such as the allocation of data across cores and chips and the layout of data within a data structure, play a key role in maximizing performance, especially when scaling across many processor cores and chips.

Power Systems use a cache-coherent SMP design in which all the memory in the system is accessible to all of the processor cores in the system, and all of the cache is coherently maintained:¹²

1. Any processor core on any chip can access the memory of the entire system.
2. Any processor core can access the contents of any core cache, even if it on a different chip.

Processor core access: In both of these cases, the processor core can access only memory or cache that it has authorized access to using normal operating system and Hypervisor memory access permissions and controls.

In POWER7 Systems, each chip consists of eight processor cores, each with on-core L1 instruction and d-caches, an L2 cache, and an L3 cache, as shown in Figure 2-2 on page 31.

¹¹ *Multiple page size support*, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

¹² *Of NUMA on POWER7 in IBM i*, available at:
http://www.ibm.com/systems/resources/pwrsysperf_P7NUMA.pdf

All of these caches are effectively shared. The L2 cache has a longer access latency than L1, and L3 has a longer access latency than L2. Each chip also has memory controllers, allowing direct access to a portion of the memory DIMMs in the system.¹³ Thus, it takes longer for an application thread to access data in cache or memory that is attached to a remote chip than to access data in a local cache or memory. These types of characteristics are often referred to as *affinity performance effects* (see “The POWER7 processor and affinity performance effects” on page 14). In many cases, systems that are built around different processor models have varying characteristics (for example, while L3 is supported, it might not be implemented on some models).

Functionally, it does not matter which core in the system an application thread is running on, or what memory the data it is accessing is on. However, this situation does affect the performance of applications, because accessing a remote memory or cache takes more time than accessing a local memory or cache.¹⁴ This situation becomes even more imperative with the capability of modern systems to support massive scaling and the resulting possibility for remote accesses to occur across a large processor interconnection complex.

The effect of these system properties can be observed by application threads, because they often move, sometimes rather frequently, between processor cores. This situation can happen for various reasons, such as a page fault or lock contention that results in the application thread being preempted while it waits for a condition to be satisfied, and then being resumed on a different core. Any application data that is in the cache local to the original core is no longer in the local cache, because the application thread moved and a remote cache access is required.¹⁵ Although modern operating systems, such as AIX, attempt to ensure that cache and memory affinity is retained, this movement does occur, and can result in a loss in performance. For an introduction to the concepts of cache and memory affinity, see “The POWER7 processor and affinity performance effects” on page 14.

The IBM POWER Hypervisor is responsible for:

- ▶ Virtualization of processor cores and memory that is presented to the operating system
- ▶ Ensuring that the affinity between the processor cores and memory an LPAR is using is maintained as much as possible

However, it is important for application designers to consider affinity issues in the design of applications, and to carefully assess the impact of application thread and data placement on the cores and the memory that is assigned to the LPAR the application is running in.

Various techniques that are employed at the system level can alleviate the effect of cache sharing. One example is to configure the LPAR so that the amount of memory that is requested for the LPAR is satisfied by the memories that are locally available to processor cores in the system (the memory DIMMs that are attached to the memory controllers for each processor core). Here, it is more likely that the POWER Hypervisor is able to maintain affinity between the processor cores and memory that is assigned to the partition, improving performance¹⁶.

For more information about LPAR configuration and running the `lssrad` command to query the affinity characteristics of a partition, see Chapter 3, “The POWER Hypervisor” on page 55.

¹³ Ibid

¹⁴ Ibid

¹⁵ Ibid

¹⁶ Ibid

The rest of this section covers multiple topics that can affect application performance, including the effects of cache geometry, alignment of data, and sensitivity to the scaling of applications to more cores. Tips are provided for using the various functionalities that are provided in Power Systems and AIX.

Cache geometry

Cache geometry refers to the specific layout of the caches in the system, including their location, interconnection, and sizes. These design details change for every processor chip, even within the Power Architecture. Figure 2-2 shows the layout of a POWER7 chip, including the processor cores, caches, and local memory. Table 2-6 shows the cache sizes and related geometry information for POWER7.

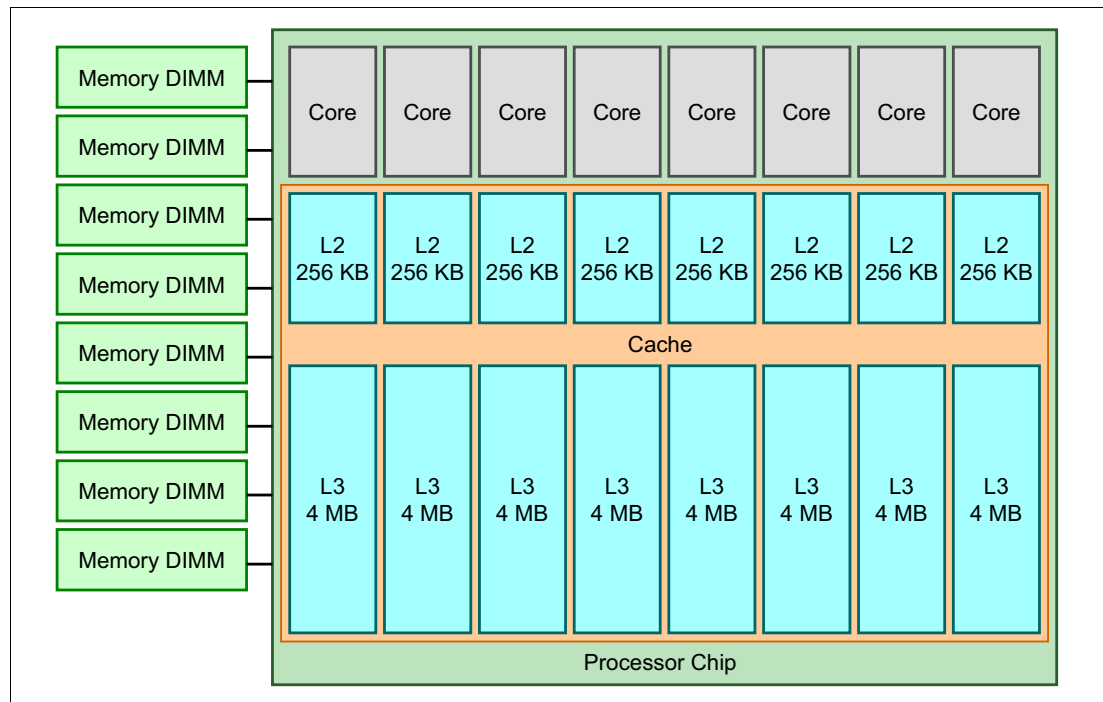


Figure 2-2 POWER7 chip and local memory¹⁷

Table 2-6 POWER7 storage hierarchy¹⁸

Cache	POWER7	POWER7+
L1 i-cache: Capacity/associativity	32 KB, 4-way	32 KB, 4-way
L1 d-cache: Capacity/associativity bandwidth	32 KB, 8-way 2 16 B reads or 1 16 B writes per cycle	32 KB, 8-way 2 16 B reads or 1 16 B writes per cycle
L2 cache: Capacity/associativity bandwidth	256 KB, 8-way Private 32 B reads and 16 B writes per cycle	256 KB, 8-way Private 32 B reads and 16 B writes per cycle

¹⁷ Ibid

¹⁸ Ibid

Cache	POWER7	POWER7+
L3 cache: Capacity/associativity bandwidth	On-Chip 4 MB/core, 8-way 16 B reads and 16 B writes per cycle	On-Chip 10 MB/core, 8-way 16 B reads and 16 B writes per cycle

Optimizing for cache geometry

There are several ways to optimize for cache geometry, as described in this section.

Splitting structures into hot and cold elements

A technique for optimizing applications to take advantage of cache is to lay out data structures so that fields that have a high rate of reference (that is, hot) are grouped, and fields that have a relatively low rate of reference (that is, cold) are grouped.¹⁹ The concept is to place the hot elements into the same *byte* region of memory, so that when they are pulled into the cache, they are co-located into the same cache line or lines. Additionally, because hot elements are referenced often, they are likely to stay in the cache. Likewise, the cold elements are in the same area of memory and result in being in the same cache line, so that being written out to main storage and discarded causes less of a performance degradation. This situation occurs because they have a much lower rate of access. Power Systems use 128-byte length cache lines. Compared to Intel processors (64-byte cache lines), these larger cache lines have the advantage of increasing the reach possible with the same size cache directory, and the efficiency of the cache by covering up to 128-bytes of hot data in a single line. However, it also has the implication of potentially bringing more data into the cache than needed for fine-grained accesses (that is, less than 64 bytes).

As described in *Eliminate False Sharing, Stop your CPU power from invisibly going down the drain*,²⁰ it is also important to carefully assess the impact of this strategy, especially when applied to systems where there are a high number of CPU cores and a phenomenon referred to as *false sharing* can occur. False sharing occurs when multiple data elements are in the same cache line that can otherwise be accessed independently. For example, if two different hardware threads wanted to update (store) two different words in the same cache line, only one of them at a time can gain exclusive access to the cache line to complete the store. This situation results in:

- ▶ Cache line transfers between the processors where those threads are
- ▶ Stalls in other threads that are waiting for the cache line
- ▶ Leaving all but the most recent thread to update the line without a copy in their cache

This effect is compounded as the number of application threads that share the cache line (that is, threads that are using different data in the cache line under contention) is scaled upwards.^{21, 20} The discussion about cache sharing²² in also presents techniques for analyzing false sharing and suggestions for addressing the phenomenon.

¹⁹ *Splitting Data Objects to Increase Cache Utilization (Preliminary Version, 9th October 1998)*. available at: <http://www.ics.uci.edu/%7Efranz/Site/pubs-pdf/ICS-TR-98-34.pdf>

²⁰ *Eliminate False Sharing, Stop your CPU power from invisibly going down the drain*, available at: <http://drdobbs.com/goparallel/article/showArticle.jhtml?articleID=217500206>

²¹ Ibid

²² Ibid

Prefetching to avoid cache miss penalties

Prefetching to avoid cache miss penalties is another technique that is used to improve performance of applications. The concept is to prefetch blocks of data to be placed into the cache a number of cycles before the data is needed. This action hides the penalty of waiting for the data to be read from main storage. Prefetching can be speculative when, based on the conditional path that is taken through the code, the data might end up not actually being required. The benefit of prefetching depends on how often the prefetched data is used. Although prefetching is not strictly related to cache geometry, it is an important technique.

A caveat to prefetching is that, although it is common for the technique to improve performance for single-thread, single core, and low utilization environments, it actually can decrease performance in high thread-count per-socket and high-utilization environments. Most systems today virtualize processors and the memory that is used by the workload. Because of this situation, the application designer must consider that, although an LPAR might be assigned only a few cores, the overall system likely has a large number of cores. Further, if the LPARs are sharing processor cores, the problem becomes compounded.

The `dcbt` and `dcbtst` instructions are commonly used to prefetch data.^{23,24} *Power Architecture ISA 2.06 Stride N Prefetch Engines to boost Application's performance*²⁵ provides an overview about how these instructions can be used to improve application performance. These instructions can be used directly in hand-tuned assembly language code, or they can be accessed through compiler built-ins or directives.

Prefetching is also automatically done by the POWER7 hardware and is configurable, as described in 2.3.7, “Data prefetching using d-cache instructions and the Data Streams Control Register (DSCR)” on page 46.

Alignment of data

Processors are optimized for accessing data elements on their naturally aligned boundaries. Unaligned data accesses might require extra processing time by the processor for individual load or store instructions. They might require a trap and emulation by the host operating system. Ensuring natural data alignment also ensures that individual accesses do not span cache line boundaries.

Similar to the idea of splitting structures into hot and cold elements, the concept of data alignment seeks to optimize cache performance by ensuring that data does not span across multiple cache lines. The cache line size in Power Systems is 128 bytes.

The general technique for alignment is to keep operands (data) on *natural* boundaries, such as a word or doubleword boundary (that is, an int would be aligned to be on a word boundary in memory). This technique might involve padding and reordering data structures to avoid cases such as the interleaving of chars and doubles: `char; double; char; double`. High-level language compilers do automatic data alignment. However, padding must be carefully analyzed to ensure that it does not result in more cache misses or page misses (especially for rarely referenced groupings of data).

²³ `dcbt` (Data Cache Block Touch) Instruction, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_dcbt_insts.htm

²⁴ `dcbtst` (Data Cache Block Touch for Store) Instruction, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_dcbtst_insts.htm

²⁵ *Power Architecture ISA 2.06 Stride N prefetch Engines to boost Application's performance*, available at:
<https://www.power.org/documentation/whitepaper-on-stride-n-prefetch-feature-of-isa-2-06/> (registration required)

Additionally, to achieve optimal performance, floating point and VMX/VSX have different alignment requirements. For example, the preferred VSX alignment is 16 bytes instead of the element size of the data type being used. This situation means that VSX data that is smaller than 16 bytes in length must be padded out to 16 bytes. The compilers introduce padding as necessary to provide optimal alignment for vector data types.

Sensitivity of scaling to more cores

Different processor chip versions and system models provide less or more scaling of LPARs and workloads to cores. Different processor chips and systems might have different bus widths and latencies. All of these factors result in the sensitivity of the performance of an application/workload to the number of cores it is running on to change based on the processor chip version and system model.

In general terms, an application that tends to not access memory without CPU intervention (that are core-centric) scales perfectly across more cores. Performance loss when scaling across multiple cores tends to come from one or more of the following sources:

- ▶ Increased cache misses (often from invalidations of data by other processor cores, especially for locks)
- ▶ The increased cost of cache misses, which in turn drives overall memory and interconnect fabric traffic into the region of bandwidth limitations (saturating the memory busses and interconnect)
- ▶ The additional cores that are being added to the workload in other nodes, resulting in increased latency in reaching memory and caches in those nodes

Briefly, cache miss requests and returning data can end up being routed through busses that connect multiple chips and memory, which have particular bandwidth and latency characteristics. The goal for scaling across multiple cores, then, is to minimize the change in the potential penalties that are associated with cache misses and data requests as the workload size grows.

It is difficult to assess what strategies are effective for scaling to more cores without considering the complex aspects of a specific application. For example, if all of the cores that the application is running across eventually access all of the data, then it might be wise to interleave data across the processor sockets (which are typically a grouping of processor chips) to optimize them from a memory bus utilization point of view. However, if the access pattern to data is more localized so that, for most of the data, separate processor cores are accessing it most of the time, the application might obtain better performance if the data is close to the processor core that is accessing that data the most (maintaining affinity between the application thread and the data it is accessing). For the latter case, where the data ought to be close to the processor core that is accessing the data, the AIX `MEMORY_AFFINITY=MCM` environment variable can be set to achieve this behavior.

When multiple processor cores are accessing the same data and that data is being held by a lock, resulting in the data line in the cache that is invalidated, programs can suffer. This phenomenon is often referred to as *hot locks*, where a lock is holding data that has a high rate of contention. Hot locks result in intervention and can easily limit the ability to scale a workload because all updates to the lock are serialized. Tools such as `sp1at` (see “AIX trace-based analysis tools” on page 165) can be used to identify hot locks.

Hot locks can be caused by the programmer having lock control access to too large an area of data, which is known as *coarse-grained locking*.²⁶ In that case, the strategy to effectively deal with a hot lock is to split the lock into a set of fine-grained locks, such that multiple locks, each managing a smaller portion of the data than the original lock, now manage the data for which access is being serialized. Hot locks can also be caused by trying to scale an application to more cores than the original design intended. In that case, using an even finer grain of locking might be possible, or changes can be made in data structures or algorithms, such that lock contention is reduced.

Additionally, the programmer must spend time considering the layout of locks in the cache to ensure that multiple locks, especially hot locks, are not in the same cache line because any updates to the lock itself results in the cache line being invalidated on other processor cores. When possible, locks should be padded so that they are in their own distinct cache line.

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3.3 SMT priorities

POWER5 introduced the capability for the SMT thread priority level for each hardware thread to be set, controlling the relative priority of the threads within a single core. This relative difference between the priority of each hardware thread determines the number of decode cycles each thread receives during a period.²⁷ Typically, changing the SMT priority level is done by using a special no-op **OR** instruction or by using the `thread_set_smt_priority` system call in AIX. The result can be boosted performance for the sibling SMT threads on the same processor core.

Concepts and benefits

The POWER processor architecture uses SMT to provide multiple streams of hardware execution. POWER7 provides four SMT hardware threads per core and can be configured to run in SMT4, SMT2, or single-threaded mode (SMT1 mode or, as referred to in this publication, ST mode) while POWER6 and POWER5 provide two SMT threads per core and can be run in SMT2 mode or ST mode.

By using multiple SMT threads, a workload can take advantage of more of the hardware features provided in the POWER processor than if a single SMT thread is used per core. By configuring the processor core to run in multi-threaded mode, the operating system can maximize the usage of the hardware capabilities that are provided in the system and the overall workload throughput by correctly balancing software threads across all of the cores and SMT hardware threads in the partition.

The Power Architecture provides an SMT Thread Priority mechanism by which the priority among the SMT threads in the processor core can be adjusted so that an SMT thread can receive more or less favorable performance (in terms of dispatch cycles) than the other threads in the same core. This mechanism can be used in various situations, such as to boost the performance of other threads while the thread with a lowered priority is waiting on a lock, or when waiting on other cooperative threads to reach a synchronization point.

²⁶ *Lock granularity*, available at: <http://www.read.seas.harvard.edu/~kohler/class/05s-osp/notes/notes8.html>

²⁷ *thread_set_smt_priority system call*, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.kerneltechref/doc/ktchrfl/thread_set_smt_priority.htm

SMT thread priority levels

Table 2-7 lists various SMT thread priority levels that are supported in the Power Architecture. The level at which code can set the SMT priority level to is controlled by the privilege level that the code is running at (such as problem-state versus supervisor level). For example, code that is running in problem-state cannot set the SMT priority level to High. However, AIX provides a system call interface that allows the SMT priority level to be set to any level other than the ones restricted to hypervisor code.

Table 2-7 SMT thread priority levels for POWER5, 6, and 7^{28, 29}

SMT thread priority level	Minimum privilege that is required to set level in POWER5, POWER6, and POWER7
Thread shutoff (read only; set by disabling thread)	Hypervisor
Very low	Supervisor
Low	Problem-state
Medium low	Problem-state
Medium	Problem-state
Medium high	Supervisor
High	Supervisor
Very high	Hypervisor

Various methods for setting the SMT priority level are described in “APIs” on page 37.

AIX kernel usage of SMT thread priority and effects

The AIX kernel is optimized to take advantage of SMT thread priority by lowering the SMT thread priority in select code paths, such as when spinning in the wait process. When the kernel modifies the SMT thread priority and execution is returned to a process-thread, the kernel sets the SMT thread priority back to Medium or the level that is specified by the process-thread using an AIX system call that modified the SMT thread priority (see “APIs” on page 37).

Where to use

SMT thread priority can be used to improve the performance of a workload by lowering the SMT thread priority that is being used on an SMT thread that is running a particular process-thread when:

- ▶ The thread is waiting on a lock
- ▶ The thread is waiting on an event, such as the completion of an IO event

Alternatively, process-threads that are performance sensitive can maximize their performance by ensuring that the SMT thread priority level is set to an elevated level.

²⁸ Setting Very Low SMT priority requires only the Problem-State privilege on POWER7+ processors. The required privilege to set a particular SMT thread priority level is associated with the physical processor implementation that the LPAR is running on, and not the processor compatible mode. Therefore, setting Very Low SMT priority only requires user level privilege on POWER7+ processors, even when running in P6-, P6+, or P7-compatible modes.

²⁹ Power ISA Version 2.06 Revision B, available at:
http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

APIs

There are three ways to set the SMT priority when it is running on POWER processors:^{30, 31}

1. Modify the SMT priority directly using the PPR register.³²
2. Modify the SMT priority through the usage of special no-ops.³³
3. Using the AIX `thread_set_smt_priority` system call.³⁴

On POWER7 and earlier, code that is running in problem-state can only set the SMT priority level to Low, Medium-Low, or Medium. On POWER7+, code that is running in problem-state can additionally set the SMT priority to Very-Low.

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3.4 Storage synchronization (sync, lwsync, lwarx, stwcx, and eieio)

The Power Architecture storage model provides for out-of-order storage accesses, providing opportunities for performance enhancement when accesses do not need to be in order. However, when accessing storage shared by multiple processor cores or shared with I/O devices, it is important that accesses occur in the correct order that is required for the sharing mechanisms that is used.

The architecture provides mechanisms for synchronization of such storage accesses and defines an architectural model that ought to be adhered to by software. Several synchronization instructions are provided by the architecture, such as `sync`, `lwsync`, `lwarx`, `stwcx`, and `eieio`. There are also operating system-specific locking services provided that enforce such synchronization. Software must be carefully designed when you use these mechanisms to ensure optimal performance while providing appropriate data consistency because of their inherent heavyweight nature.

Concepts and benefits

The Power Architecture defines a storage model that provides weak ordering of storage accesses. The order in which memory accesses are performed might differ from the program order and the order in which the instructions that cause the accesses are run.³⁵

The Power Architecture provides a set of instructions that enforce storage access synchronization, and the AIX kernel provides a set of kernel services that provide locking mechanisms and associated synchronization support.³⁶ However, such mechanisms come with an inherent cost because of the nature of synchronization. Thus, it is important to intelligently use the correct storage mechanisms for the various types of storage access scenarios to ensure that accesses are performed in program order while minimizing their impact.

³⁰ *Power ISA Version 2.06 Revision B*, available at:

http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

³¹ `thread_set_smt_priority` system call, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.kerneltechref/doc/ktech_chrf1/thread_set_smt_priority.htm

³² *Power ISA Version 2.06 Revision B*, available at:

http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

³³ *Ibid*

³⁴ `thread_set_smt_priority` or `thread_read_smt_priority` System Call, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.kerneltechref/doc/ktech_chrf1/thread_set_smt_priority.htm

³⁵ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, by Lyons, et al, available at:

<http://www.ibm.com/developerworks/systems/articles/powerpc.html>

³⁶ *Ibid*

AIX kernel locking services

AIX provides a set of locking services that enforce synchronization by using mechanisms that are provided by the Power Architecture. These services are documented in online publications.³⁷ The correct use of these locking services allows code to ensure that shared memory is accessed by only one producer or consumer of data at a time.

Associated instructions

The following instructions provide various storage synchronization mechanisms:

- sync** This instruction provides an ordering function, so that all instructions issued before the **sync** complete and no subsequent instructions are issued until after the **sync** completes.³⁸
- lwsync** This instruction provides an ordering function similar to **sync**, but it is only applicable to **load**, **store**, and **dcbz** instructions that are run by the processor (hardware thread) running the **lwsync** instruction, and only for specific combinations of storage control attributes.³⁹
- lwarx** This instruction reserves a storage location for subsequent store using a **stwcx** instruction and notifies the memory coherence mechanism of the reservation.⁴⁰
- stwcx** This instruction performs a store to the target location only if the location specified by a previous **lwarx** instruction is not used for storage by another processor (hardware thread) or mechanism, which invalidates the reservation.⁴¹
- eieio** This instruction creates a memory barrier that provides an order for storage accesses caused by **load**, **store**, **dcbz**, **eciwx**, and **ecowx** instructions.⁴²

Where to use

Care must be taken when you use synchronization mechanisms in any processor architecture because the associated load and store instructions have a heavier weight than normal loads and stores and the barrier operations have a cost that is associated with them. Thus, it is imperative that the programmer carefully consider when and where to use such operations, so that data consistency is ensured without adversely affecting the performance of the software and the overall system.

³⁷ *Locking Kernel Services*, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.kernelext/doc/kernextc/lock_kernsvcs.htm

³⁸ *sync (Synchronize) or dcs (Data Cache Synchronize) Instruction*, available at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.aixassem/doc/alangref/sync.htm>

³⁹ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, Michael Lyons, et al, available at:
<http://www.ibm.com/developerworks/systems/articles/powerpc.html>

⁴⁰ *lwarx (Load Word and Reserve Indexed) Instruction*, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_lwarx_lwri_instrs.htm

⁴¹ *stwcx (Store Word Conditional Indexed) Instruction*, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_stwcx_instrs.htm

⁴² *eieio instruction*, available at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_eieio_instrs.htm

*PowerPC storage model and AIX programming*⁴³ describes where synchronization mechanisms must be used to ensure that the code adheres to the Power Architecture. Although this documentation covers how to write compliant code, it does not cover the performance aspect of using the mechanisms.

Unless the code is hand-tuned assembler code, you should take advantage of the locking services that are provided by AIX because they are tuned and provide the necessary synchronization mechanisms. *Power Instruction Set Architecture Version 2.06*⁴⁴ provides assembler programming examples for sharing storage. For more information, see Appendix B, “Performance tooling and empirical performance analysis” on page 155

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3.5 Vector Scalar eXtension (VSX)

VSX in the Power ISA introduced more support for Vector and Scalar Binary Floating Point Operations conforming to the IEEE-754 Standard for Floating Point Arithmetic. The introduction of VSX in to the Power Architecture increases the parallelism by providing Single Instruction Multiple Data (SIMD) execution functionality for floating point double precision to improve the performance of the HPC applications.

The following VSX features are provided to increase opportunities for vectorization:

- ▶ A unified register file, a set of Vector-Scalar Registers (VSR), supporting both scalar and vector operations is provided, eliminating the impact of vector-scalar data transfer through storage.
- ▶ Support for word-aligned storage accesses for both scalar and vector operations is provided.
- ▶ Robust support for IEEE-754 for both vector and scalar floating point operations is provided.

⁴³ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, Michael Lyons, et al, available at:

<http://www.ibm.com/developerworks/systems/articles/powerpc.html>

⁴⁴ *Power ISA Version 2.06 Revision B*, available at:

http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

A 64-entry Unified Register File is shared across VSX, the Binary floating point unit (BFP), VMX, and the DFP unit. The 32 64-bit Floating Point Registers (FPRs), which are used by the BFP and DFP units, are mapped to registers 0 - 31 of the Vector Scalar Registers. The 32 vector registers (VRs) that are used by the VMX are mapped to registers 32 - 63 of the VSRs,⁴⁵ as shown in Table 2-8.

Table 2-8 The Unified Register File

FPR0		VSR0
FPR1		VSR1
..		
..		
FPR30		
FPR31		
VR0		
VR1		
..		
..		
VR30		VSR62
VR31		VSR63

VSX supports Double Precision Scalar and Vector Operations and Single Precision Vector Operations. VSX instructions numbering 142 are broadly divided into two categories that can operate on 64 vector scalar registers:^{46, 47, 48, 49, 50}

- ▶ Computational instructions: Addition, subtraction, multiplication, division, extracting the square root, rounding, conversion, comparison, and combinations of these operations
- ▶ Non-computational instructions: Loads/stores, moves, select values, and so on

⁴⁵ *What's New in the Server Environment of Power ISA v2.06*, a white paper from Power.org, available at: <https://www.power.org/documentation/whats-new-in-the-server-environment-of-power-isa-v2-06/> (registration required)

⁴⁶ *Support for POWER7 processors*, available at: <http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/gstart/architecture.html>

⁴⁷ *Vector Built-in Functions*, available at: http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/compiler_ref/vec_intrin_cpp.html

⁴⁸ *Vector Initialization*, available at: http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/language_ref/vector_init.html

⁴⁹ *Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL*, available at: <http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.essl.doc/esslbooks.html>

⁵⁰ *What's New in the Server Environment of Power ISA v2.06?*, a white paper from Power.org, available at: <https://www.power.org/documentation/whats-new-in-the-server-environment-of-power-isa-v2-06/> (registration required)

Compiler support for vectors

XLC supports vector processing technologies through language extensions on both AIX and Linux. GCC supports using the VSX engine on Linux. XL and GCC C implement and extend the AltiVec Programming Interface specification. In the extended syntax, type qualifiers and storage class specifiers can precede the keyword vector (or its alternative spelling, `__vector`) in a declaration.

Also, the XL compilers are able to automatically generate VSX instructions from scalar code when they generate code that targets the POWER7 processor. This task is accomplished by using the `-qsimd=auto` option with the `-O3` optimization level or higher.

Table 2-9 lists the supported vector data types and the size and possible values for each type.

Table 2-9 Vector data types

Type	Interpretation of content	Range of values
vector unsigned char	16 unsigned char	0..255
vector signed char	16 signed char	128..127
vector bool char	16 unsigned char	0, 255
vector unsigned short	8 unsigned short	0..65535
vector unsigned short int		
vector signed short	8 signed short	32768..32767
vector signed short int		
vector bool short	8 unsigned short	0, 65535
vector bool short int		
vector unsigned int	4 unsigned int	0..232-1
vector unsigned long		
vector unsigned long int		
vector signed int	4 signed int	231..231-1
vector signed long		
vector signed long int		
vector bool int	4 unsigned int	0, 232-1
vector bool long		
vector bool long int		
vector float	4 float	IEEE-754 single (32 bit) precision floating point values
vector double	2 double	IEEE-754 double (64 bit) precision floating point values
vector pixel	8 unsigned short	1/5/5/5 pixel

Vector types: The *vector double* type requires architectures that support the VSX instruction set extensions, such as POWER7. You must specify the XL `-qarch=pwr7 -qaltivec` compiler options when you use this type, or the GCC `-mcpu=power7` or `-mvsx` options.

The hardware does not have instructions for supporting vector unsigned long long, vector bool long long, or vector signed long long. In GCC, you can declare these types, but the only hardware operation you can use these types for is vector floating point convert. In 64-bit mode, vector long is the same as vector long long. In 32-bit mode, these types are not permitted.

All vector types are aligned on a 16-byte boundary. An aggregate that contains one or more vector types is aligned on a 16-byte boundary, and padded, if necessary, so that each member of vector type is also 16-byte aligned. Vector data types can use some of the unary, binary, and relational operators that are used with primitive data types. All operators require compatible types as operands unless otherwise stated. For more information about the operator's usage, see the XLC online publications^{51, 52, 53}.

Individual elements of vectors can be accessed by using the Vector Multimedia eXtension (VMX) or the VSX built-in functions. For more information about the VMX and the VSX built-in functions, refer to the built-in functions section of *Vector Built-in Functions*.⁵⁴

Vector initialization

A vector type is initialized by a vector literal or any expression that has the same vector type. For example:⁵⁵

```
vector unsigned int v1;
vector unsigned int v2 = (vector unsigned int)(10); // XL only, not GCC
v1 = v2;
```

The number of values in a braced initializer list must be less than or equal to the number of elements of the vector type. Any uninitialized element is initialized to zero.

⁵¹ *Support for POWER7 processors*, available at:
http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/get_start/architecture.html

⁵² *Vector Built-in Functions*, available at:
http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/compiler_ref/vec_intrin_cpp.html

⁵³ *Vector Initialization*, available at:
http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/language_ref/vector_init.html

⁵⁴ *Vector Built-in Functions*, available at:
http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/compiler_ref/vec_intrin_cpp.html

⁵⁵ *Vector types (IBM extension)*, available at:
http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/language_ref/altivec_types.html

Here are examples of vector initialization using initializer lists:

```
vector unsigned int v1 = {1}; // initialize the first 4 bytes of v1 with 1
                          // and the remaining 12 bytes with zeros
vector unsigned int v2 = {1,2}; // initialize the first 8 bytes of v2 with 1 and 2
                          // and the remaining 8 bytes with zeros
vector unsigned int v3 = {1,2,3,4}; // equivalent to the vector literal
                          // (vector unsigned int) (1,2,3,4)
```

How to use vector capability in POWER7

When you target a POWER processor that supports VMX or VSX, you can request the compiler to transform code into VMX or VSX instructions. These machine instructions can run up to 16 operations in parallel. This transformation mostly applies to loops that iterate over contiguous array data and perform calculations on each element. You can use the NOSIMD directive to prevent the transformation of a particular loop.⁵⁶

- ▶ Using a compiler: Compiler versions that recognize the POWER7 architecture are XL C/C++ 11.1 and XLF Fortran 13.1 or recent versions of GCC, including the Advance Toolchain, and the SLES 11SP1 or Red Hat RHEL6 GCC compilers:
 - For C:
 - `xlc -qarch=pwr7 -qtune=pwr7 -O3 -qhot -qsimd`
 - `gcc -mcpu=power7 -mtune=power7 -O3`
 - For Fortran
 - `xlf -qarch=pwr7 -qtune=pwr7 -O3 -qhot -qsimd`
 - `gfortran -mcpu=power7 -mtune=power7 -O3`
- ▶ Using Engineering and Scientific Subroutine (ESSL) libraries with vectorization support:
 - Select routines have vector analogs in the library
 - Key FFT, BLAS routines

Vector capability support in AIX

A program can determine whether a system supports the vector extension by reading the `vmx_version` field of the `_system_configuration` structure. If this field is non-zero, then the system processor chips and operating system contain support for the vector extension. A `__power_vmx()` macro is provided in `/usr/include/sys/systemcfg.h` for performing this test. A value of 2 means that the processor chip is both VMX and VSX capable.

The AIX Application Binary Interface (ABI) is extended to support the addition of vector register state and conventions. AIX supports the AltiVec programming interface specification.

A set of malloc subroutines (`vec_malloc`, `vec_free`, `vec_realloc`, and `vec_calloc`) is provided by AIX that give 16-byte aligned allocations. Vector-enabled compilation, with `_VEC_` implicitly defined by the compiler, result in any calls to older mallocs and callocs being redirected to their vector-safe counterparts, `vec_malloc` and `vec_calloc`. Non-vector code can also be explicitly compiled to pick up these same malloc and calloc redirections by explicitly defining `__AIXVEC`.

⁵⁶ Ibid

The alignment of the default `malloc()`, `realloc()`, and `calloc()` allocations can also be controlled at run time. This task can be done externally to any program by using the `MALLOCALIGN` environment variable, or internally to a program by using the `mallopt()` interface command option.⁵⁷

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3.6 Decimal floating point (DFP)

Decimal (base 10) data is widely used in commercial and financial applications. However, most computer systems have only binary (base two) arithmetic. There are two binary number systems in computers: integer (fixed-point) and floating point. Unfortunately, decimal calculations cannot be directly implemented with binary floating point. For example, the value 0.1 needs an infinitely recurring binary fraction, while a decimal number system can represent it exactly, as one tenth. So, using binary floating point cannot ensure that results are the same as those results using decimal arithmetic.

In general, DFP operations are emulated with binary fixed-point integers. Decimal numbers are traditionally held in a binary-coded decimal (BCD) format. Although BCD provides sufficient accuracy for decimal calculation, it imposes a heavy cost in performance, because it is usually implemented in software.

IBM POWER6 and POWER7 processor-based systems provide hardware support for DFP arithmetic. The POWER6 and POWER7 microprocessor cores include a DFP unit that provides acceleration for the DFP arithmetic. The IBM Power instruction set is expanded: 54 new instructions were added to support the DFP unit architecture. DFP can provide a performance boost for applications that are using BCD calculations.⁵⁸

How to take advantage of DFP unit on POWER

You can take advantage of the DFP unit on POWER with the following features:⁵⁹

- ▶ Native DFP language support with a compiler

The C draft standard includes the following new data types (these are native data types, as are `int`, `long`, `float`, `double`, and so on):

<code>_Decimal32</code>	7 decimal digits of accuracy
<code>_Decimal64</code>	16 decimal digits of accuracy
<code>_Decimal128</code>	34 decimal digits of accuracy

Note: The `printf()` function uses new options to print these new data types:

- ▶ `_Decimal32` uses `%Hf`
- ▶ `_Decimal64` uses `%Df`
- ▶ `_Decimal128` uses `%DDf`

⁵⁷ *AIX Vector Programming*, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.genprogc/doc/genprogc/vector_prog.htm

⁵⁸ *How to Leverage Decimal Floating-Point unit on POWER6 for Linux*, available at:

<http://www.ibm.com/developerworks/wikis/display/hpccentral/How+to+Leverage+Decimal+Floating-Point+unit+on+POWER6+for+Linux>

⁵⁹ *How to compile DFPAL?*, available at: <http://speleotrove.com/decimal/dfpal/compile.html>

- The IBM XL C/C++ Compiler, release 9 or later for AIX and Linux, includes native DFP language support. Here is a list of compiler options for IBM XL compilers that are related to DFP:

- **-qdfp**: Enables DFP support. This option makes the compiler recognize DFP literal suffixes, and the `_Decimal32`, `_Decimal64`, and `_Decimal128` keywords.
- **-qfloat=dfpemulate**: Instructs the compiler to use calls to library functions to handle DFP computation, regardless of the architecture level. You might experience performance degradation when you use software emulation.
- **-qfloat=nodfpemulate** (the default when the **-qarch** flag specifies POWER6 or POWER7): Instructs the compiler to use DFP hardware instructions.
- **-D__STDC_WANT_DEC_FP__**: Enables the referencing of DFP-defined symbols.
- **-ldfp**: Enables the DFP functionality that is provided by the Advance Toolchain on Linux.

For hardware supported DFP, with **-qarch=pwr6** or **-qarch=pwr7**, use the following command:

```
cc -qdfp
```

For software emulation of DFP (on earlier processor chips), use the following command:

```
cc -qdfp -qfloat=dfpemulate
```

- The GCC compilers for Power Systems also include native DFP language support.

As of SLES/11/SP1, and RHEL6, IEEE 754R, DFP is fully integrated with compiler and run time (printf and DFP math) support. For older Linux distribution releases (RHEL5/SLES10 and earlier), you can use the freely available Advance Toolchain compiler and run time. The Advance Toolchain runtime libraries can also be integrated with recent XL (V9+) compilers for DFP exploitation.

The latest Advance Toolchain compiler and run times can be downloaded from the following website:

<http://linuxpatch.ncsa.uiuc.edu/toolchain/at/>

Advance Toolchain is a self-contained toolchain that does not rely on the base system toolchain for operability. In fact, it is designed to coexist with the toolchain shipped with the operating system. You do not have to uninstall the regular GCC compilers that come with your Linux distribution to use the Advance Toolchain.

The latest Enterprise distributions and Advance Toolchain run time use the Linux CPU tune library capability to automatically select hardware DFP or software implementation library variants, which are based on the hardware platform.

Here is a list of GCC compiler options for Advance Toolchain that are related to DFP:

- **-mhard-dfp** (the default when **-mcpu=power6** or **-mcpu=power7** is specified): Instructs the compiler to directly take advantage of DFP hardware instructions for decimal arithmetic.
- **-mno-hard-dfp**: Instructs the compiler to use calls to library functions to handle DFP computation, regardless of the architecture level. If your application is dynamically linked to the **libdfp** variant and running on POWER6 or POWER7 processors, then the run time automatically binds to the **libdfp** variant implemented with hardware DFP instructions. Otherwise, the software DFP library is used. You might experience performance degradation when you use software emulation.

- `-D__STDC_WANT_DEC_FP__`: Enables the reference of DFP defined symbols.
 - `-ldfp`: Enables the DFP functionality that is provided by recent Linux Enterprise Distributions or the Advance Toolchain run time.
- Decimal Floating Point Abstraction Layer (DFPAL), which is a no additional cost, downloadable library from IBM.⁶⁰

Many applications that are using BCD today use a library to perform math functions. Changing to a native data type can be hard work, after which you might have an issue with one code set for AIX on POWER6 and one for other platforms that do not support native DFP. The solution to this problem is DFPAL, which is an alternative to the native support. DFPAL contains a header file to include in your code and the DFPAL library.

The header file is downloadable from General Decimal Arithmetic at <http://speleotrove.com/decimal/> (search for “DFPAL”). Download the complete source code, and compile it on your system.

If you have hardware support for DFP, use the library to access the functions.

If you do not have hardware support (or want to compare the hardware and software emulation), you can force the use of software emulation by setting a shell variable before you run your application by running the following command:

```
export DFPAL_EXE_MODE=DNSW
```

Determining if your applications are using DFP

There are two AIX commands that are used for monitoring:

- **hpmstat** (for monitoring the whole system)
- **hpmcount** (for monitoring a single program)

The `PM_DFU_FIN` (DFU instruction finish) field in the output of the **hpmstat** and **hpmcount** commands verifies that the DFP operations finished.

The `-E PM_MRK_DFU_FIN` option in the **tprof** command uses the AIX trace subsystem, which tells you which functions are using DFP and how often.

For more information about this topic, see 2.4, “Related publications” on page 51.

2.3.7 Data prefetching using d-cache instructions and the Data Streams Control Register (DSCR)

The hardware data prefetch mechanism reduces the performance impact that is caused by the latency in retrieving cache lines from higher level caches and from memory. The data prefetch engine of the processor can recognize sequential data access patterns in addition to certain non-sequential (stride-N) patterns and initiate prefetching of d-cache lines from L2 and L3 cache and memory into the L1 d-cache to improve the performance of these storage reference patterns.

The Power ISA architecture also provides cache instructions to supply a hint to prefetch engines for data prefetching to override the automatic stream detection capability of the data prefetcher. Cache instructions, such as **dcbt** and **dcbtst**, allow applications to specify stream direction, prefetch depth, and number of units. These instructions can avoid the starting cost of the automatic stream detection mechanism.

⁶⁰ Ibid

The d-cache instructions **dcbt** (d-cache block touch) and **dcbtst** (d-cache block touch for store) affect the behavior of the prefetched lines. The syntax for the assembly language instructions is:⁶¹

```
dcbt RA, RB, TH
dcbtst RA, RB, TH
```

- ▶ *RA* specifies a source general-purpose register for Effective Address (EA) computation.
- ▶ *RB* specifies a source general-purpose register for EA computation.
- ▶ *TH* indicates when a sequence of d-cache blocks might be needed.

The block that contains the byte addressed by the EA is fetched into the d-cache before the block is needed by the program. The program can later perform loads and stores from the block and might not experience the added delay that is caused by fetching the block into the cache.

The Touch Hint (TH) field is used to provide a hint that the program probably loads or stores to the storage locations specified by the Effective Address (EA) and the TH field. The hint is ignored for locations that are caching-inhibited or guarded. The encodings of the TH field depend on the target architecture that is selected with the `-m` flag or the `.machine` assembly language pseudo-op.

The range of values for the TH field is 0b01000 - 0b01111.

The **dcbt** and **dcbtst** instructions provide hints about a sequence of accesses to data elements, or indicate the expected use. Such a sequence is called a *data stream*, and a **dcbt** or **dcbtst** instruction in which TH is set to one of these values is said to be a *data stream variant* of **dcbt** or **dcbtst**.

A data stream to which a program can perform *Load* accesses is said to be a *load data stream*, and is described using the data stream variants of the **dcbt** instruction.

A data stream to which a program can perform *Store* accesses is said to be a *store data stream*, and is described using the data stream variants of the **dcbtst** instruction.

The contents of the DSCR, a special purpose register, affects how the data prefetcher responds to hardware-detected and software-defined data streams.

The layout of the DSCR register is:

	URG ^a	LSD	SNSE	SSE	DPFD
--	------------------	-----	------	-----	------

a. POWER7+ only

Where:

- ▶ Bits 58 – LSD – Load Stream Disable
Disables hardware detection and initiation of load streams.
- ▶ Bits 59 – SNSE – Stride-N Stream Enable
Enables hardware detection and initiation of load and store streams that have a stride greater than a single cache block. Such load streams are detected when LSD = 0 and such store streams are detected when SSE=1.
- ▶ Bits 60 – SSE – Store Stream Enable
Enables hardware detection and initiation of Store streams.

⁶¹ *Power ISA Version 2.06 Revision B*, available at:
http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

► Bits 61:63 – DPF – Default Prefetch Depth

Supplies a prefetch depth for hardware-detected streams and for software-defined streams for which a depth of zero is specified, or for which `dcbt` or `dcbtst` with TH=1010 is *not* used in their description.

► Bits 55:57 - URG - Depth Attainment Urgency

This field is a new one added in the POWER7+ processor. This field indicates how quickly the prefetch depth should be reached for hardware-detected streams. Values and their meanings are as follows:

- 0: Default
- 1: Not urgent
- 2: Least urgent
- 3: Less urgent
- 4: Medium
- 5: Urgent
- 6: More urgent
- 7: Most urgent

The ability to enable or disable the three types of streams that the hardware can detect (load streams, store streams, or stride-N streams), or to set the default prefetch depth, allows empirical testing of any application. There are no simple rules for determining which settings are optimum overall for an application: the performance of prefetching depends on many different characteristics of the application in addition to the characteristics of the specific system and its configuration. Data prefetches are purely speculative, meaning they can improve performance greatly when the data that is prefetched is, in fact, referenced by the application later, but can also degrade performance by expending bandwidth on cache lines that are not later referenced, or by displacing cache lines that are later referenced by the program.

Similarly, setting DPF to a deeper depth tends to improve performance for data streams that are predominately sourced from memory because the longer the latency to overcome, the deeper the prefetching must be to maximize performance. But deeper prefetching also increases the possibility of stream overshoot, that is, prefetching lines beyond the end of the stream that are not later referenced. Prefetching in multi-core processor implementations has implications for other threads or processes that are sharing cache (in SMT mode) or the same system bandwidth.

Controlling DSCR under Linux

DSCR settings on Linux are controlled with the `ppc64_cpu` command. Controlling DSCR settings for an application is generally considered advanced and specific tuning.

Currently, setting the DSCR value is a cross-LPAR setting.

Controlling DSCR under AIX

Under AIX, DSCR settings can be controlled both by programming API and from the command line by running the following commands:^{62,63}

`dscr_ctl()` API

```
#include <sys/machine.h>
int dscr_ctl(int op, void *buf_p, int size)
```

⁶² `dscr_ctl` subroutine, available at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.basetechref/doc/basetrf1/dscr_ctl.htm

⁶³ `dscrctl` command, available at:

<http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmds2/dscrctl.htm>

Where:

- op:** Operation. Possible values are **DSCR_WRITE**, **DSCR_READ**, **DSCR_GET_PROPERTIES**, and **DSCR_SET_DEFAULT**.
- Buf_p:** Pointer to an area of memory where the values are copied from (**DSCR_WRITE**) or copied to (**DSCR_READ** and **DSCR_GET_PROPERTIES**). For **DSCR_WRITE**, **DSCR_READ**, and **DSCR_SET_DEFAULT** operations, **buf_p** must be a pointer to a 64-bit data area (long long *). For **DSCR_GET_PROPERTIES**, **buf_p** must be a pointer to a **struct dscr_properties** (defined in <sys/machine.h>).
- Size:** Size in bytes of the area pointed to by **buf_p**.

Function:

The action that is taken depends on the value of the operation parameter that is defined in <sys/machine.h>:

DSCR_WRITE	Stores a new value from the input buffer into the process context and in the DSCR.
DSCR_READ	Reads the current value of DSCR and returns it in the output buffer.
DSCR_GET_PROPERTIES	Reads the number of hardware streams that are supported by the platform, the platform (firmware) default Prefetch Depth and the Operating System default Prefetch Depth from kernel memory, and returns the values in the output buffer (struct dscr_properties defined in <sys/machine.h>).
DSCR_SET_DEFAULT	Sets a 64-bit DSCR value in a buffer pointed to by buf_p as the operating system default. Returns the old default in the buffer pointed to by buf_p . Requires root authority. The new default value is used by all the processes that do not explicitly set a DSCR value using DSCR_WRITE . The new default is not permanent across reboots. For an operating system default prefetch depth that is permanent across reboots, use the dscrctl command, which adds an entry into the inittab to initialize the system-wide prefetch depth default value upon reboot (for a description of this command, see “The dscrctl command” on page 50).

Return values are:

0 if successful

-1 if an error detected. In this case, **errno** is set to indicate the error. Possible values are:

EINVAL	Invalid value for DSCR (DSCR_WRITE , DSCR_SET_DEFAULT).
EFAULT	Invalid address that is passed to function.
EPERM	Operation not permitted (DSCR_SET_DEFAULT by non-root user).
ENOTSUP	Data streams that are not supported by platform hardware.

Symbolic values for the following SSE and DPFID fields are defined in <sys/machine.h>:

DPFD_DEFAULT	0
DPFD_NONE	1
DPFD_SHALLOWEST	2
DPFD_SHALLOW	3

DPFD_MEDIUM	4
DPFD_DEEP	5
DPFD_DEEPER	6
DPFD_DEEPEST	7
DSCR_SSE	8

Here is a description of the **dscr_properties** structure in <sys/machine.h>:

```
struct dscr_properties {
    uintversion;
    uintnumber_of_streams; /* Number of HW streams */
    longlongplatform_default_pd; /* PFW default */
    longlongos_default_pd; /* AIX default */
    longlong dscr_res[5]; /* Reservd for future use */
};
```

Here is an example of this structure:

```
#include <sys/machine.h>
int rc;
long long dscr = DSCR_SSE | DPDFD_DEEPER;
rc = dscr_ctl(DSCR_WRITE, &dscr);
...
```

A new process inherits the DSCR from its parent during a *fork*. This value is reset to the system default during exec.

When a thread is dispatched (starts running on a CPU), the value of the DSCR for the owning process is written in the DCSR. You do not need to save the value of the register in the process context when the thread is *undispatched* because the system call writes the new value both in the process context and in the DCSR.

When a thread runs **dscr_ctl** to change the prefetch depth for the process, the new value is written into the AIX process context and the DCSR register of the thread that is running the system call. If another thread in the process is concurrently running on another CPU, it starts using the new DSCR value only after the new value is reloaded from the process context area after either an interrupt or a redispatch. This action can take as much as 10 ms (a clock tick).

The dscrctl command

The system administrator can use this command to read the current settings for the hardware streams mechanism and set a system wide value for the DSCR. The DSCR is privileged. It can be read or written only by the operating system.

To query the characteristics of the hardware streams on the system, run the following command:

```
dscrctl -q
```

Here is an example of this command:

```
# dscrctl -q
Current DSCR settings:
    number_of_streams = 16
    platform_default_pd = 0x5 (DPFD_DEEP)
    os_default_pd = 0xd (DSCR_SSE | DPDFD_DEEP)
```

To set the operating system default prefetch depth on the system temporarily (that is, for the current session) or permanently (that is, after each reboot), run the following command:

```
dscrctl [-n] [-b] -s <dscr_value>
```

The **dscr_value** is treated as a decimal number unless it starts with 0x, in which case it is treated as hexadecimal.

To cancel a permanent setting of the operating system default prefetch depth at boot time, run the following command:

```
dscrctl -c
```

Applications that have predictable data access patterns, such as numerical applications that process arrays of data in a sequential manner, benefit from aggressive data prefetching. These applications must run with the default operating system prefetch depth, or whichever settings are empirically found to be the most beneficial.

Applications that have considerably unpredictable data access patterns, such as some transactional applications, can be negatively affected by aggressive data prefetching. The data that is prefetched is unlikely to be needed, and the prefetching uses system bandwidth and might displace useful data from the caches. Some WebSphere Application Server and DB2 workloads have this characteristic. Performance can be improved by disabling hardware prefetching in these cases by running the following command:

```
dscrctl -n -s 1
```

This system (partition) wide disabling is only appropriate if it is expected to benefit all of the applications that are running in the partition. However, the same effect can be achieved on an application-specific basis by using the programming API.

For more information about this topic, see 2.4, “Related publications” on page 51.

2.4 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *AIX dscr_ctl API sample code*, found at:
<https://www.power.org/documentation/performance-guide-for-hpc-applications-on-ibm-power-755-system/> (registration required)
- ▶ *AIX Version 7.1 Release Notes*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.nt1/RELNOTES/GI11-9815-00.htm>
Refer to the section, The **dscrctl** command.
- ▶ *Application configuration for large pages*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/config_apps_large_pages.htm
- ▶ *False Sharing*, found at:
<http://msdn.microsoft.com/en-us/magazine/cc872851.aspx>
- ▶ *lwsync instruction*, found at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.assem/doc/alongref/idalangref_sync_dcs_instrs.htm

- ▶ *Multiprocessing*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/intro_multitproc.htm
- ▶ *The Performance of Runtime Data Cache Prefetching in a Dynamic Optimization System*, found at:
<http://www.microarch.org/micro36/html/pdf/1u-PerformanceRuntimeData.pdf>
- ▶ *POWER6 Decimal Floating Point (DFP)*, found at:
<http://www.ibm.com/developerworks/wikis/display/WikiPtype/Decimal+Floating+Point>
- ▶ *POWER7 Processors: The Beat Goes On*, found at:
<http://www.ibm.com/developerworks/wikis/download/attachments/104533501/POWER7+-+The+Beat+Goes+On.pdf>
- ▶ *Power Architecture ISA 2.06 Stride N prefetch Engines to boost Application's performance*, found at:
<https://www.power.org/documentation/whitepaper-on-stride-n-prefetch-feature-of-isa-2-06/> (registration required)
- ▶ *Power ISA Version 2.06 Revision B*, found at:
http://power.org/wp-content/uploads/2012/07/PowerISA_V2.06B_V2_PUBLIC.pdf

Refer to the following sections:

- Section 3.1: Program Priority Registers
 - Section 3.2: “or” Instruction
 - Section 4.3.4: Program Priority Register
 - Section 4.4.3: OR Instruction
 - Section 5.3.4: Program Priority Register
 - Section 5.4.2: OR Instruction
 - Book I – 4 Floating Point Facility
 - Book I – 5 Decimal Floating Point
 - Book I – 6 Vector Facility
 - Book I – 7 Vector-Scalar Floating Point Operations (VSX)
 - Book I – Chapter 5 Decimal Floating-Point.
 - Book II – 4.2 Data Stream Control Register
 - Book II – 4.3.2 Data Cache Instructions
 - Book II – 4.4 Synchronization Instructions
 - Book II – A.2 Load and Reserve Mnemonics
 - Book II – A.3 Synchronize Mnemonics
 - Book II – Appendix B. Programming Examples for Sharing Storage
 - Book III – 5.7 Storage Addressing
- ▶ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, found at:
<http://www.ibm.com/developerworks/systems/articles/powerpc.html>

Refer to the following sections:

- Power Instruction Set Architecture
 - Section 4.4.3 Memory Barrier Instructions – Synchronize
- ▶ *Product documentation for XL C/C++ for AIX, V12.1 (PDF format)*, found at:
<http://www.ibm.com/support/docview.wss?uid=swg27024811>

- ▶ *Simple performance lock analysis tool (splat)*, found at:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_splat.htm
- ▶ *Simultaneous Multithreading*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.genprog/doc/genprog/smt.htm>
- ▶ *splat Command*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/splat.htm>
- ▶ *trace Daemon*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/trace.htm>
- ▶ *What makes Apple's PowerPC memcpy so fast?*, found at:
<http://stackoverflow.com/questions/1990343/what-makes-apples-powerpc-memcpy-so-fast>
- ▶ *What programmers need to know about hardware prefetching?*, found at:
<http://www.futurechips.org/chip-design-for-all/prefetching.html>



The POWER Hypervisor

This chapter introduces the POWER7 Hypervisor and describes some of the technical details for this product. It covers the the following topics:

- ▶ Introduction to the POWER7 Hypervisor
- ▶ POWER7 virtualization

3.1 Introduction to the POWER7 Hypervisor

Power Virtualization was introduced in POWER5 systems, so there are many reference materials that are available that cover all three resources (CPU, memory, and I/O), virtualization, capacity planning, and virtualization management. Some of these documents are shown in the reference section at the end of this section, which focuses on POWER7 Virtualization usually. As for any workload deployments, capacity planning, selecting the correct set of technologies, and appropriate tuning are critical to deploying high performing workloads. However, in deploying workloads in virtualized environments, there are more aspects to consider, such as consolidation ratio, workload resource usage patterns, and the suitability of a workload to run in a shared resource environment (or latency requirements).

The first step in the virtualization deployment process is to understand if the performance of a workload in a shared resource environment meets customer requirements. If the workload requires consistent performance with stringent latency requirements, then such workloads must be deployed on a dedicated partition rather than on a shared LPAR. The exceptions are where shared processor pool is not heavily over committed and overutilized; such workloads could meet stringent requirements in a shared LPAR configuration also.

It is a preferred practice to understand the resource usage of all workloads that are planned for consolidation on a single system, especially when you plan to use a shared resource model, such as shared LPARs, IBM Active Memory™ Sharing, and VIO server technologies. The next step is to use a capacity planning tool that takes virtualization impacts into consideration, such as the IBM Workload Estimator, to estimate capacity for each partition. One of the goals of virtualization is maximizing usage. This usage can be achieved by consolidating workloads that peak at different times (that is, in a non-overlapping manner, so each workload (or partition) does not have to be sized for peak usage but rather for average usage). At the same time, each workload can grow to consume free resources from the shared pool that belong to other partitions on the system. This situation allows the packing of more partitions (workloads) on a single system, producing a higher consolidation ratio or higher density on the deployed system. A higher consolidation ratio is a key metric to achieve in the data center, as it helps to reduce the total cost of ownership (TCO).

Let us look at a list of key attributes that require consideration when deploying workloads on a shared resource model (virtualization):

- ▶ Levels of variation between average and peak usage of workloads:
 - A large difference between average and peak usage
 - A small difference between average and peak usage
- ▶ Workloads and their peak duration, frequency, and estimate when they potentially peak:
Select workloads that peak at different times (non-overlapping).
- ▶ Workload Service Level Agreement SLA requirements (latency requirements and their tolerance levels).
- ▶ Ratio of active to inactive (mostly idle) partitions on a system.
- ▶ Provisioning and de-provisioning frequency.
- ▶ IBM PowerVM has a richer set of technology options than virtualization on other platforms. It supports dedicated, shared, and a mix of dedicated and shared resource models for each of the system resources, such as processor cores, memory, and I/O:
 - Shared LPAR: Capped versus uncapped.
 - Shared LPAR: Resources overcommit levels to meet the peak usage (the ratio of virtual processors to physical processor entitled capacity).

- Shared LPAR: Weight selection to assign a level of priority to get uncapped capacity (excess cycles to address the peak usage).
- Shared LPAR: Multiple shared pools to address software licensing costs, which prevents a set of partitions from exceeding its capacity consumption.
- Active Memory Sharing: The size of a shared pool is based on active workload memory consumption:
 - Inactive workload memory is used for active workloads, which reduces the memory capacity of the pool.
 - The Active Memory De-duplication option can reduce memory capacity further.
 - AIX file system cache memory is loaned to address memory demands that lead to memory savings.
 - Workload load variation changes active memory consumption, which leads to opportunity for sharing.
- Active Memory Sharing: A shared pool size determines the levels of memory overcommit. Starts without overcommit and is based on workload consumption that reduces the pool.
- Active Memory Expansion: AIX working set memory is compressed.
- Active Memory Sharing and Active Memory Expansion can be deployed on the same workload.
- Active Memory Sharing: VIO server sizing is critical for CPU and memory.
- Virtual Ethernet: An inter-partition communication VLANs option that is used for higher network performance.
- Shared Ethernet versus host Ethernet.
- Virtual disk I/O: Virtual small computer system interface (vSCSI), N_Port ID Virtualization (NPIV), file-backed storage, and storage pool.
- Dynamic resource movement (DLPAR) to adopt to growth.

3.2 POWER7 virtualization

PowerVM hypervisor and the AIX operating system (AIX V6.1 TL 5 and later versions) on POWER7 implement enhanced affinity in a number of areas to achieve optimized performance for workloads that are running in a virtualized shared processor logical partition (SPLPAR) environment. By using the preferred practices that are described in this guide, customers can attain optimum application performance in a shared resource environment. This guide covers preferred practices in the context of POWER7 Systems, so this section can be used as an addendum to other PowerVM preferred practice documents.

3.2.1 Virtual processors

A virtual processor is a unit of a virtual processor resource that is allocated to a partition or virtual machine. PowerVM hypervisor can map a whole physical processor core, or it can create a time slice of a physical processor core.

PowerVM hypervisor create time slices of Micro-Partitioning on physical CPUs by dispatching and undischatching the various virtual processors for the partitions that are running in the shared pool.

If a partition has multiple virtual processors, they might or might not be scheduled to run simultaneously on the physical processor cores.

Partition entitlement is the guaranteed resource available to a partition. A partition that is defined as capped can consume only the processors units that are explicitly assigned as its entitled capacity. An uncapped partition can consume more than its entitlement, but is limited by many factors:

- ▶ Uncapped partitions can exceed their entitlement if there is unused capacity in the shared pool, dedicated partitions that share their physical processor cores while active or inactive, unassigned physical processors, and Capacity on Demand (CoD) utility processors.
- ▶ If the partition is assigned to a virtual shared processor pool, the capacity for all of the partitions in the virtual shared processor pool might be limited.
- ▶ The number of virtual processors in an uncapped partition is throttled depending on how much CPU it can consume. For example:
 - An uncapped partition with one virtual CPU can consume only one physical processor core of CPU resources under any circumstances.
 - An uncapped partition with four virtual CPUs can consume only four physical processor cores of CPU.
- ▶ Virtual processors can be added or removed from a partition using HMC actions.

Sizing and configuring virtual processors

The number of virtual processors in each LPAR in the system ought not to *exceed* the number of cores available in the system (central electronic complex (CEC)/framework). Or, if the partition is defined to run in a specific virtual shared processor pool, the number of virtual processors ought not to exceed the maximum that is defined for the specific virtual shared processor pool. Having more virtual processors that are configured than can be running at a single point in time does not provide any additional performance benefit and can actually cause more context switches of the virtual processors, which reduces performance.

If there are sustained periods during which there is sufficient demand for all the shared processing resources in the system or a virtual shared processor pool, it is prudent to configure the number of virtual processors to match the capacity of the system or virtual shared processor pool.

A single virtual processor can consume a whole physical core under two conditions:

1. SPLPAR has an entitlement of 1.0 or more processors.
2. The partition is uncapped and there is idle capacity in the system.

Therefore, there is no need to configure more than one virtual processor to get one physical core.

For example, a shared pool is configured with 16 physical cores. Four SPLPARs are configured, each with entitlement 4.0 cores. To configure virtual processors, consider the sustained peak demand capacity of the workload. If two of the four SPLPARs would peak to use 16 cores (the maximum available in the pool), then those two SPLPARs would need 16 virtual CPUs. If the other two SPLPARs peak only up to eight cores, those two would be configured with eight virtual CPUs.

Entitlement versus virtual processors

Entitlement is the capacity that an SPLPAR is ensured to get as its share from the shared pool. Uncapped mode allows a partition to receive excess cycles when there are free (unused) cycles in the system.

Entitlement also determines the number of SPLPARs that can be configured for a shared processor pool. The sum of the entitlement of all the SPLPARs cannot exceed the number of physical cores that are configured in a shared pool.

For example, a shared pool has eight cores and 16 SPLPARs are created, each with 0.1 core entitlement and one virtual CPU. We configured the partitions with 0.1 core entitlement because these partitions are not running that frequently. In this example, the sum of the entitlement of all the 16 SPLPARs comes to 1.6 cores. The rest of the 6.4 cores and any unused cycles from the 1.6 entitlement can be dispatched as uncapped cycles.

At the same time, keeping entitlement low when there is capacity in the shared pool is not always a preferred practice. Unless the partitions are frequently idle or there is plan to add more partitions, the preferred practice is that the sum of the entitlement of all the SPLPARs configured should be close to the capacity in the shared pool. Entitlement cycles are guaranteed, so while a partition is using its entitlement cycles, the partition is not preempted, while a partition can be preempted when it is dispatched to use excess cycles. Following this preferred practice allows the hypervisor to optimize the affinity of the partition's memory and processor cores and also reduces unnecessary preemptions of the virtual processors.

Matching entitlement of an LPAR close to its average usage for better performance

The aggregate entitlement (minimum or wanted processor) capacity of all LPARs in a system is a factor in the number of LPARs that can be allocated. The minimum entitlement is what is needed to boot the LPARs, but the wanted entitlement is what an LPAR gets if there are enough resources available in the system. The preferred practice for LPAR entitlement is to match the entitlement capacity to average usage and let the peak be addressed by more uncapped capacity.

When to add more virtual processors

When there is sustained need for a shared LPAR to use more resources in the system in uncapped mode, increase the virtual processors.

How to estimate the number of virtual processors per uncapped shared LPAR

The first step is to monitor the usage of each partition and for any partition where the average utilization is about 100%, and then add one virtual processor, that is, use the capacity of the configured virtual processors before you add more. Additional virtual processors run concurrently if there are enough free processor cores available in the shared pool.

If the peak usage is below the 50% mark, then there is no need for more virtual processors. In this case, look at the ratio of virtual processors to configured entitlement and if the ratio is greater than 1, then consider reducing the ratio. If there are too many virtual processors that are configured, AIX can “fold” those virtual processors so that the workload would run on fewer virtual processors to optimize virtual processor performance.

For example, if an SPLPAR is given a CPU entitlement of 2.0 cores and four virtual processors in an uncapped mode, then the hypervisor can dispatch the virtual processors to four physical cores concurrently if there are free cores available in the system. The SPLPAR uses unused cores and the applications can scale up to four cores. However, if the system does not have free cores, then the hypervisor dispatches four virtual processors on two cores so that the concurrency is limited to two cores. In this situation, each virtual processor is dispatched for a reduced time slice as two cores are shared across four virtual processors. This situation can impact performance, so AIX operating system processor folding support might be able to reduce to number of virtual processors that are dispatched so that only two or three virtual processors are dispatched across the two physical cores.

Virtual processor management: Processor folding

The AIX operating system monitors the usage of each virtual processor and aggregate usage of an SPLPAR, and if the aggregate usage goes below 49%, AIX starts folding down the virtual CPUs so that fewer virtual CPUs are dispatched. This action has the benefit of virtual CPUs running longer before it is preempted, which helps improve performance. If a virtual CPU gets a shorter dispatch time slice, then more workloads are cut into time slices on the processor core, which can cause higher cache misses.

If the aggregate usage of an SPLPAR goes above 49%, AIX starts unfolding virtual CPUs so that additional processor capacity can be given to the SPLPAR. Virtual processor management dynamically adopts number of virtual processors to match the load on an SPLPAR. This threshold (`vpm_fold_threshold`) of 49% represents the SMT thread usage starting with AIX V6.1 TL6; before that version, `vpm_fold_threshold` (which was set to 70%) represents the core utilization.

With a `vpm_fold_threshold` value of 49%, the primary thread of a core is used before unfolding another virtual processor to consume another core from the shared pool on POWER7 Systems. If free cores are available in the shared processor pool, then unfolding another virtual processor results in the LPAR getting another core along with its associated caches. Now the SPLPAR can run on two primary threads of two cores instead of two threads (primary and secondary) on the same core. A workload that is running on two primary threads of two cores can achieve higher performance if there is less sharing of data than the workload that is running on primary and secondary threads of the same core. The AIX virtual processor management default policy aims at using the primary thread of each virtual processor first; therefore, it unfolds the next virtual processor without using the SMT threads of the first virtual processor. After it unfolds all the virtual processors and consumes the primary thread of all the virtual processors, it starts using the secondary and tertiary threads of the virtual processors.

If the system is highly used and there are no free cycles in the shared pool, when all the SPLPARs in the system try to get more cores by unfolding more virtual processors and use only the primary of thread of each core, the hypervisor creates time slices on the physical cores across multiple virtual processors. This action impacts the performance of all the SPLPARs, as time slicing increases cache misses and context switch cost

However, this alternative policy of making each virtual processor use all of the four threads (SMT4 mode) of a physical core can be achieved by changing the values of a number of restricted tunables. Do not use this change in normal conditions, as most of the systems do not consistently run at high usage. You decide if such a change is needed based on the workloads and system usage levels. For example, critical database SPLPAR needs more cores even in a highly contended situation to achieve the best performance; however, the production and test SPLPARs can be sacrificed by running on fewer virtual processors and using all the SMT4 threads of a core.

Processor bindings in a shared LPAR

In AIX V6.1 TL5 and AIX V7.1, binding virtual processors is available to an application that is running in a shared LPAR. An application process can be bound to a virtual processor in a shared LPAR. In a shared LPAR, a virtual processor is dispatched by the PowerVM hypervisor. The PowerVM hypervisor maintains three levels of affinity for dispatching, such as core, chip, and node level affinity in eFW7.3 and later firmware versions. By maintaining affinity at the hypervisor level and in AIX, applications can achieve higher level affinity through virtual processor bindings.

3.2.2 Page table sizes for LPARs

The hardware page table of an LPAR is sized based on the maximum memory size of an LPAR and not what is assigned (or wanted) to the LPAR. There are some performance considerations if the maximum size is set higher than the wanted memory:

- ▶ A larger page table tends to help performance of the workload, as the hardware page table can hold more pages. This larger table reduces translation page faults. Therefore, if there is enough memory in the system and you want to improve translation page faults, set your max memory to a higher value than the LPAR wanted memory.
- ▶ On the downside, more memory is used for hardware page table, which not only wastes memory, but also makes the table become sparse, which results in the following situations:
 - A dense page table tends to help with better cache affinity because of reloads.
 - Less memory that is consumed by the hypervisor for the hardware page table means that more memory is made available to the applications.
 - There is less page walk time as page tables are small.

3.2.3 Placing LPAR resources to attain higher memory affinity

POWER7 PowerVM optimizes the allocation of resources for both dedicated and shared partitions as each LPAR is activated. Correct planning of the LPAR configuration enhances the possibility of getting both CPU and memory in the same domain in relation to the topology of a system.

PowerVM hypervisor selects the required processor cores and memory that is configured for an LPAR from the system free resource pool. During this selection process, hypervisor takes the topology of the system into consideration and allocates processor cores and memory where both resources are close. This situation ensures that the workload on an LPAR has lower latency in accessing its memory.

When you power on partitions for the first time, power on the partitions of highest importance first. By doing so, the partitions have first access to deallocated memory and processing resources.

Partition powering on: Even though a partition is dependent on a VIOS, it is safe to power on the partition before the VIOS; the partition does not fully power on because of its dependency on the VIOS, but claims its memory and processing resources.

What the SPPL option does on a Power 795 system

A new option named Shared Partition Processor Limit (SPPL) is added to give hints to the hypervisor about whether to contain partitions to minimum domains or to spread partitions across multiple domains. On Power 795, a book can host four chips that total up to 32 cores. If SPPL is set to 32, then the maximum size of an LPAR that can be supported is 32 cores. This hint enables hypervisor to allocate both physical cores and memory of an LPAR within a single domain as much as possible. For example, in a three-book configuration, if the wanted configuration is four LPARs, each with 24 cores, three of those LPARs are contained in each of the three books, and the fourth LPAR is spread across three books.

If SPPL is set to MAX, then a partition size can exceed 32 cores. This hint helps hypervisor to maximize the interconnect bandwidth allocation by spreading LPARs across more domains for larger size LPARs.

SPPL value: The SPPL value can be set on only Power 795 systems that contain four or more processing books. If there are three or less processor books, the SPPL setting is controlled by the system and is set to 32 or 24, based on the number of processors per book.

On a Power 795 system, where the SPPL value is set to MAX, there is a way to configure individual partitions so they are still packed into a minimum number of books. This setup is achieved by using the HMC command-line interface (CLI) through the `lpar_placement` profile attribute on the `chsyscfg` command. Specifying a value of `lpar_placement=1` indicates that the hypervisor attempts to minimize the number of domains that are assigned to the LPAR. Setting `lpar_placement=0` is the default setting, and follows the existing rules when SPPL is set to MAX.

How to determine if an LPAR is contained within a domain

From an AIX LPAR, run `lssrad` to display the number of domains across which an LPAR is spread.

The `lssrad` syntax is:

```
lssrad -av
```

If all the cores and memory are in a single domain, you should receive the following output with only one entry under REF1:

REF1	SRAD	MEM	CPU
0	0	31806.31	0-31
	1	31553.75	32-63

REF1 represents a domain, and domains vary by platform. SRAD always references a chip. However, `lssrad` does not report the actual physical domain or chip location of the partition: it is a relative value whose purpose is to inform if the resources of the partition are within the same domain or chip. The output of this `lssrad` example indicates that the LPAR is allocated with 16 cores from two chips within the same domain. Note that the `lssrad` command output was taken from an SMT4 platform, and those CPU 0-31 actually represents 8 cores.

When all the resources are free (an initial machine state or reboot of the CEC), the PowerVM allocates memory and cores as optimally as possible. At partition boot time, PowerVM is aware of all of the LPAR configurations, so placement of processors and memory are made regardless of the order of activation of the LPARs.

However, after the initial configuration, the setup might not stay static. Numerous operations take place, such as:

- ▶ Reconfiguration of existing LPARs with new profiles
- ▶ Reactivating existing LPARs and replacing them with new LPARs
- ▶ Adding and removing resources to LPARs dynamically (DLPAR operations)

Any of these changes might result in memory fragmentation, causing LPARs to be spread across multiple domains. There are ways to minimize or even eliminate the spread. For the first two operations, the spread can be minimized by releasing the resources that are currently assigned to the deactivated LPARs.

Resources of an LPAR can be released by running the following commands:

- ▶ `chhwres -r mem -m <system_name> -o r -q <num_of_Mbytes> --id <lp_id>`
- ▶ `chhwres -r proc -m <system_name> -o r --procunits <number> --id <lp_id>`

The first command frees the memory and the second command frees cores.

When all of the partitions are inactive, there is another way to clear the resources of all of the existing configurations before you create a configuration. In this situation, all the resources of all the partitions can be cleared from the HMC by completing the following steps:

1. Shut down all partitions.
2. Create the all-resources partition.
3. Activate the all-resources partition.
4. Shut down the all-resources partition.
5. Delete the all-resources partition.

Fragmentation because frequent movement of memory or processor cores between partitions is avoidable with correct planning. DLPAR actions can be done in a controlled way so that the performance impact of resource addition or deletion is minimal. Planning for growth helps alleviate the fragmentation that is caused by DLPAR operations. Knowing the LPARs that must grow or shrink dynamically, and placing them with LPARs that can tolerate nodal crossing latency (less critical LPARs), is one approach to handling the changes of critical LPARs dynamically. In such a configuration, when growth is needed for the critical LPAR, the resources that are assigned to the non-critical LPAR can be reduced so that the critical LPAR can grow.

Affinity groups (introduced in the 730 firmware level)

PowerVM firmware 730 and later has support for affinity groups that can be used to group multiple LPARs to place (allocate resources) within a single or a few domains. On the Power 795 with 32 cores in a book, the total physical core resources of an affinity group are not to exceed 24/32 cores or the physical memory that is contained within a book.

This affinity group feature can be used in multiple situations:

- ▶ LPARs that are dependent or related, such as server and client, and application server and database server, can be grouped so they are in the same book.
- ▶ Affinity groups can be created that are large enough such that they force the assignment of LPARs to be in different books. For example, if you have a two-book system and the total resources (memory and processor cores) assigned to the two groups exceeds the capacity of a single book, these two groups are forced to be in separate books. A simple example is if there is a group of partitions that totals 14 cores and a second group that totals 20 cores. Because these groups exceed the 32 cores in a 795 book, the groups are placed in different books.

If a pair of LPARs is created with the intent of one being a failover to another partition, and one partition fails, the other partition (which is placed in the same node, if both are in the same affinity group) uses all of the resources that were freed up from the failed LPAR.

The following HMC CLI command adds or removes a partition from an affinity group:

```
chsyscfg -r prof -m <system_name> -i name=<profile_name>  
lpar_name=<partition_name>,affinity_group_id=<group_id>
```

group_id is a number 1 - 255 (255 groups can be defined), and **affinity_group_id=none** removes a partition from the group.

When the hypervisor places resources at frame reboot, it first places all the LPARs in group 255, then the LPARs in group 254, and so on. Place the most important partitions regarding affinity in the highest configured group.

PowerVM resource consumption for capacity planning considerations

PowerVM hypervisor consumes a portion of the memory resources in the system. During your planning stage, consider the layout of LPARs. Factors that affect the amount of memory that is consumed are the size of the hardware page tables in the partitions, HEA resources, HCA resources, the number of I/O devices, hypervisor memory mirroring, and other factors. Use the *IBM System Planning Tool* (available at <http://www.ibm.com/systems/support/tools/systemplanningtool/>) to estimate the amount of memory that is reserved by the hypervisor.

Licensing resources and Capacity Upgrade on Demand (CUoD)

Power Systems support capacity on demand so that customers can license capacity on demand as business needs for compute capacity grows. Therefore, a Power System might not have all resources that are licensed, which poses a challenge to allocate both cores and memory from a local domain. PowerVM (eFW 7.3 level firmware) correlates customer configurations and licensed resources to allocated cores and memory from the local domain to each of the LPARs. During a CEC reboot, the hypervisor places all defined partitions as optimally as possible and then unlicenses the unused resources.

For more information about this topic, see 3.3, “Related publications” on page 65.

3.2.4 Active memory expansion

Active memory expansion (AME) is a capability that is supported on POWER7 and later servers that employs memory compression technology to expand the effective memory capacity of an LPAR. The operating system identifies the least frequently used memory pages and compresses them. The result is that more memory capacity within the LPAR is available to sustain more load, or the ability to remove memory from the LPAR to be used to deploy more LPARs. The POWER7+ processor provides enhanced support of AME with the inclusion of on-chip accelerators onto which the work of compression and decompression is offloaded.

AME is deployed by first using the **amepat** tool to model the projected expansion factor and CPU usage of a workload. This modeling looks at the compressibility of the data, the memory reference patterns, and current CPU usage of the workload. AME can then be enabled for the LPAR by setting the expansion factor. The operating system then reports the physical memory available to applications as *actual memory* times the *expansion factor*. Then, transparently, the operating system locates and compresses cold pages to maintain the appearance of expanded memory.

Applications do not need to change, and they are not aware that AME is active. However, not all applications or workloads have suitable characteristics for AME. Here is a partial list of guidelines for the workload characteristics that can be a good fit for AME:

- ▶ The memory footprint is dominated by application working storage (such as heap, stack, and shared memory).
- ▶ Workload data is compressible.
- ▶ Memory access patterns are concentrated to a subset of the overall memory footprint.
- ▶ Workload performance is acceptable without the use of larger page sizes, such as 64 KB pages. AME disables the usage of large pages and uses only 4 KB pages.
- ▶ The average CPU usage of the workload is below 60%.
- ▶ Users of the application and workload are relatively insensitive to response time increases.

For more information about AME usage, see *Active Memory Expansion: Overview and Usage Guide*, available at:

<ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/pow03037usen/POW03037USEN.PDF>

3.2.5 Optimizing Resource Placement – Dynamic Platform Optimizer

In firmware 760 level and later, on select Power Systems servers, a feature is available called the Dynamic Platform Optimizer. This optimizer automates the manual steps to improve resource placement. For more information visit the following Web site and select the Doc-type **Word document P7 Virtualization Best Practice**.

<https://www.ibm.com/developerworks/wikis/display/WikiPtype/Performance+Monitoring+Documentation>

Comment: This document is intended to address POWER7 processor technology based PowerVM best practices to attain the best LPAR performance. This document should be used in conjunction with other PowerVM documents.

3.3 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *Active Memory Expansion: Overview and Usage Guide*, found at:
<ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/pow03037usen/POW03037USEN.PDF>
- ▶ *IBM PowerVM Active Memory Sharing Performance*, found at:
<http://public.dhe.ibm.com/common/ssi/ecm/en/pow03017usen/POW03017USEN.PDF>
- ▶ *IBM PowerVM Virtualization Introduction and Configuration*, SG24-7940
- ▶ *IBM PowerVM Virtualization Managing and Monitoring*, SG24-7590
- ▶ *POWER7 Virtualization Best Practice Guide*, found at:
https://www.ibm.com/developerworks/wikis/download/attachments/53871915/P7_virtualization_bestpractice.doc?version=1
- ▶ *PowerVM Migration from Physical to Virtual Storage*, SG24-7825

- ▶ *Virtual I/O (VIO) and Virtualization*, found at:
<http://www.ibm.com/developerworks/wikis/display/virtualization/VIO>
- ▶ *Virtualization Best Practice*, found at:
<http://www.ibm.com/developerworks/wikis/display/virtualization/Virtualization+Best+Practice>



AIX

This chapter describes the optimization and tuning of a POWER7 processor-based server running the AIX operating system. It covers the the following topics:

- ▶ AIX and system libraries
- ▶ AIX Active System Optimizer and Dynamic System Optimizer
- ▶ AIX preferred practices

4.1 AIX and system libraries

Here we present information about AIX and system libraries.

4.1.1 AIX operating system-specific optimizations

This section describes optimization methods specific to AIX.

Malloc

Every application needs a fast, scalable, and memory efficient allocator. However, each application's memory request patterns are different. It is difficult to provide one common allocator or tunable that can satisfy the needs of all applications. AIX provides different memory allocators and suboptions within the allocator, so that a system administrator or developer can choose more suitable settings for their application. This chapter explains the available choices and when to choose them.

Memory allocators

AIX provides three different allocators, and each of them uses a different memory management algorithm and data structures. These allocators work independently, so the application developer must choose one of them by exporting the **MALLOCTYPE** environment variable. The allocators are:

► Default allocator

The default allocator is selected when the **MALLOCTYPE** environment variable is unset. This setting maintains a consistent performance, even in a worst case scenario, but might not be as memory efficient as a Watson allocator. This allocator is ideal for 32-bit applications, which do not make frequent calls to `malloc()`.

► Watson allocator

This allocator is selected when `MALLOCTYPE=watson` is set. This allocator is designed for 64-bit applications. It is memory efficient, scalable, and provides good performance. This allocator has a built-in bucket component for allocation requests up to 512 bytes. Table 4-1 provides the mapping for the allocation requests to bucket size.

Table 4-1 Mapping for allocation requests to bucket size

Request size	Bucket size	Request size	Bucket size	Request size	Bucket size	Request size	Bucket size
1 - 4		33-40	40	129-144	144	257-288	288
5 - 8		41 - 48	48	145 - 160	160	289 - 320	320
9 - 12	12	49 - 56	56	161 - 176	176	321 - 352	352
13 - 16	16	57 - 64	64	177 - 192	192	353 - 384	384
17 - 20	20	65 - 80	80	193 - 208	208	385 - 416	416
21 - 24	24	81 - 96	96	209 - 224	224	417 - 448	448
25 - 28	28	97 - 112	112	224 - 240	240	449 - 480	480
29 - 32	32	113 - 128	128	241 - 256	256	481 - 512	512

This allocator is ideal for 64-bit memory-intensive applications.

► Malloc 3.1 allocator

This allocator is selected when `MALLOCTYPE=3.1` is set. This is a bucket allocator that divides the heap into 28 hash buckets, each with a size of $2^{\text{pow}(x+4)}$, where x stands for bucket index. This allocator provides the best performance at the cost of memory. In most cases, this algorithm can use as much as twice the amount of memory that is actually requested by the application. In addition, an extra page is required for buckets larger than 4096 bytes because objects of a page in size or larger are page-aligned. Interestingly, some earlier customer applications still use this allocator, as it is more tolerant for application memory overwrite bugs.

Memory allocator suboptions

There are many suboptions available that can be selected by exporting the `MALLOCOPTIONS` environment variable. This chapter covers a few of the suboptions that are more relevant to performance tuning. For a complete list of options, see *System Memory Allocation Using the malloc Subsystem*, available at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.genprog/doc/genprog/sys_mem_alloc.htm

► Multiheap

By default, the malloc subsystem uses a single heap, which causes lock contention for internal locks that are used by malloc in case of multi-threaded applications. By enabling this option, you can configure the number of parallel heaps to be used by allocators. You can set the multiheap by exporting `MALLOCOPTIONS=multiheap[:n]`, where n can vary between 1- 32 and 32 is the default if n is not specified.

Use this option for multi-threaded applications, as it can improve performance.

► Buckets

This suboption is similar to the built-in bucket allocator of the Watson allocator. However, with this option, you can have fine-grained control over the number of buckets, number of blocks per bucket, and the size of each bucket. This option also provides a way to view the usage statistics of each bucket, which be used to refine the bucket settings.

In case the application has many requests of the same size, then the bucket allocator can be configured to preallocate the required size by correctly specifying the bucket options. The block size can go beyond 512 bytes, compared to the Watson allocator or malloc pool options.

You can enable the buckets allocator by exporting `MALLOCOPTIONS=buckets`. Complete details about the buckets options for fine-grained control are available¹. Enabling the buckets allocator turns off the built-in bucket component if the Watson allocator is used.

► malloc pools

This option enables a high performance front end to malloc subsystem for managing storage objects smaller than 513 bytes. This suboption is similar to the built-in bucket allocator of the Watson allocator. However, this suboptions maintains the bucket for each thread, providing lock-free allocation and deallocation for blocks smaller than 513 bytes. This suboption improves the performance for multi-threaded applications, as the time spent on locking is avoided for blocks smaller than 513 bytes.

The pool option makes small memory block allocations fast (no locking) and memory efficient (no header on each allocation object). The pool malloc both speeds up single threaded applications and improves the scalability of multi-threaded applications.

¹ *System Memory Allocation Using the malloc Subsystem*, available at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.genprog/doc/genprog/sys_mem_alloc.htm

► **malloc disclaim**

By enabling this option, **free()** automatically disclaims memory. This suboption is useful for reducing the paging space requirement. This option can be set by exporting `MALLOCOPTIONS=disclaim`.

Use cases

Here are some uses cases that you can use to set up your environment:

1. For a 32-bit single-threaded application, use the default allocator.
2. For a 64-bit application, use the Watson allocator.
3. Multi-threaded applications use the **multiheap** option. Set the number of heaps proportional to the number of threads in the application.
4. For single-threaded or multi-threaded applications that make frequent allocation and deallocation of memory blocks smaller than 513, use the **malloc pool** option.
5. For a memory usage pattern of the application that shows high usage of memory blocks of the same size (or sizes that can fall to common block size in bucket option) and sizes greater than 512 bytes, use the **configure malloc bucket** option.
6. For older applications that require high performance and do not have memory fragmentation issues, use **malloc 3.1**.
7. Ideally, the Watson allocator, along with the **multiheap** and **malloc pool** options, is good for most multi-threaded applications; the pool front end is fast and is scalable for small allocations, while **multiheap** ensures scalability for larger and less frequent allocations.
8. If you notice high memory usage in the application process even after you run **free()**, the **disclaim** option can help.

For more information about this topic, see 4.4, “Related publications” on page 94.

Pthread tunables

The AIX pthread library can be customized with a set of environment variables. Specific variables that improve scaling and CPU usage are listed here. A full description is provided in the following documentation settings:

► **AIXTHREAD_SCOPE={P|S}**

The **P** option signifies a process-wide contention scope (M:N) while the **S** option signifies a system-wide contention scope (1:1). Use system scope (1:1) for AIX. Although process scope (M:N) continues to be supported, it is no longer being enhanced in AIX.

► **SPINLOOPTIME=*n***

The **SPINLOOPTIME** variable controls the number of times the system tries to get a busy mutex or spin lock without taking a secondary action, such as calling the kernel to yield the process. This control is intended for MP systems, where it is hoped that the lock that is held by another actively running pthread is released. The parameter works only within libpthreads (user threads). If locks are usually available within a short period, you might want to increase the spin time by setting this environment variable. The number of times to try a busy lock before yielding to another pthread is *n*. The default is 40 and *n* must be a positive value.

► **YIELDLOOPTIME=*n***

The **YIELDLOOPTIME** variable controls the number of times that the system yields the logical processor when it tries to acquire a busy mutex or spin lock before it goes to sleep on the lock. The logical processor is yielded to another kernel thread, assuming that there is another executable thread with sufficient priority. This variable is effective in complex applications, where multiple locks are in use. The number of times to yield the logical processor before blocking on a busy lock is *n*. The default is 0 and *n* must be a positive value.

For more information about this topic, see 4.4, “Related publications” on page 94.

pollset

AIX 5L V5.3 introduced the pollset APIs. Pollsets are an AIX replacement for UNIX **select()** and **poll()**. **Pollset**, **select()**, and **poll()** all allow an application to efficiently query the status of file descriptors. This action is typically done to allow a single application to multiplex I/O across many file descriptors. Pollset APIs can be more efficient when the number of file descriptors that are queried becomes large.

Efficient I/O event polling through the pollset interface on AIX contains a pollset summary and outlines the most advantageous use of Java. To see this topic, go to:

<http://www.ibm.com/developerworks/aix/library/au-pollset/index.html>

For more information about this topic, see 4.4, “Related publications” on page 94.

File system performance benefits

AIX Enhanced Journaled File System (JFS2) is the default file system for 64-bit kernel environments. Applications can capitalize on the features of JFS2 for better performance.

Direct I/O

The AIX read-ahead and write-behind JFS2 feature might not be suitable for applications that perform large sized I/O operations, as the cache hit ratio is low. In those cases, an application developer must evaluate Direct I/O for I/O intensive applications.

Programs that are good candidates for direct I/O are typically CPU-limited and perform much disk I/O. Technical applications that have large sequential I/Os are good candidates. Applications that benefit from striping are also good candidates.

The direct I/O access method bypasses the file cache and transfers data directly from disk into the user space buffer, as opposed to using the normal cache policy of placing pages in kernel memory.

At the user level, file systems can be mounted using the **dio** option on the **mount** command.

At the programming level, applications enable direct I/O access to a file by passing the **O_DIRECT** flag to the open subroutine. This flag is defined in the `fcntl.h` file. Applications must be compiled with **_ALL_SOURCE** enabled to see the definition of **O_DIRECT**.

For more information, see *Working with File I/O*, available at:

<http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.genprog/doc/genprog/fileio.htm>

Concurrent I/O

An AIX JFS2 inode lock imposes write serialization at the file level. Serializing write accesses prevents data inconsistency because of overlapping writes. Serializing reads regarding writes ensures that the application does not read stale data.

However, some applications can choose to implement their own data serialization, usually at a finer level of granularity than the file. Therefore, they do not need the file system to implement this serialization for them. The inode lock hinders performance in such cases, by unnecessarily serializing non-competing data accesses. For such applications, AIX offers the concurrent I/O (CIO) option. Under concurrent I/O, multiple threads can simultaneously perform reads and writes on a shared file. Applications that do not enforce serialization for accesses to shared files should not use concurrent I/O, as it can result in data corruption because of competing accesses.

Enhanced JFS supports concurrent file access to files. Similar to direct I/O, this access method bypasses the file cache and transfers data directly from disk into the user space buffer.

Concurrent I/O can be specified for a file either by running `mount -o cio` or by using the `open()` system call (by using `O_CIO` as the `OFlag` parameter).

Asynchronous I/O

If an application does a synchronous I/O operation, it must wait for the I/O to complete. In contrast, asynchronous I/O operations run in the background and do not block user applications, which improves performance, because I/O operations and applications processing can run simultaneously. Many applications, such as databases and file servers, take advantage of the ability to overlap processing and I/O.

Applications can use the `aio_read()`, `aio_write()`, or `lio_listio()` subroutines (or their 64-bit counterparts) to perform asynchronous disk I/O. Control returns to the application from the subroutine when the request is queued. The application can then continue processing while the disk operation is being performed.

I/O completion ports

A limitation of the AIO interface that is used in a threaded environment is that `aio_nwait()` collects completed I/O requests for *all* threads in the same process. One thread collects completed I/O requests that are submitted by another thread.

Another limitation is that multiple threads cannot invoke the collection routines (such as `aio_nwait()`) at the same time. If one thread issues `aio_nwait()` while another thread is calling it, the second `aio_nwait()` returns `EBUSY`. This limitation can affect I/O performance when many I/Os must run at the same time and a single thread cannot run fast enough to collect all the completed I/Os.

On AIX, using I/O completion ports with AIO requests provides the capability for an application to capture the results of various AIO operations on a per-thread basis in a multi-threaded environment. This functionality provides threads with a method of receiving a completion status for only the AIO requests initiated by the thread.

You can enable IOCP on AIX by running `smitty iocp`. Verify that IOCP is enabled by running the following command:

```
lsdev -Cc iocp
```

The resulting output should match the following example:

```
iocp0 Available I/O Completion Ports
```


shmat versus mmap

Memory mapped files provide a mechanism for a process to access files by directly incorporating file data into the process address space. The use of mapped files can reduce I/O data movement because the file data does not have to be copied into process data buffers, as is done by the read and write subroutines. When more than one process maps the same file, its contents are shared among them, providing a low-impact mechanism by which processes can synchronize and communicate.

AIX provides two methods for mapping files and anonymous memory regions. The first set of services, which are known collectively as the *shmat* services, are typically used to create and use shared memory segments from a program. The second set of services, which are known collectively as the *mmap* services, is typically used for mapping files, although it can be used for creating shared memory segments as well.

Both the *mmap* and *shmat* services provide the capability for multiple processes to map the same region of an object so that they share addressability to that object. However, the *mmap* subroutine extends this capability beyond that provided by the *shmat* subroutine by allowing a relatively unlimited number of such mappings to be established. Although this capability increases the number of mappings that are supported per file object or memory segment, it can prove inefficient for applications in which many processes map the same file data into their address space. The *mmap* subroutine provides a unique object address for each process that maps to an object. The software accomplishes this task by providing each process with a unique virtual address, which is known as an *alias*. The *shmat* subroutine allows processes to share the addresses of the mapped objects.

shmat can be used to share memory segments in a way that is similar to how it creates and uses files. An *extended shmat* capability is available for 32-bit applications with their limited address spaces. If you define the **EXTSHM=ON** environment variable, then processes running in that environment can create and attach more than 11 shared memory segments.

Use the *shmat* services under the following circumstances:

- ▶ When mapping files larger than 256 MB
- ▶ When mapping shared memory regions that must be shared among unrelated processes (no parent-child relationship)
- ▶ When mapping entire files

In general, *shmat* is more efficient but less flexible.

Use *mmap* under the following circumstances:

- ▶ Many files are mapped simultaneously.
- ▶ Only a portion of a file must be mapped.
- ▶ Page-level protection must be set on the mapping (allows a 4K boundary).

For more information, see *General Programming Concepts: Writing and Debugging Programs*, available at:

http://publib16.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixprggd/genprog/understanding_mem_mapping.htm

For more information about this topic, see 4.4, “Related publications” on page 94.

Large segment tunable (AIX V6.1 only)

AIX V6.1 TL5 and AIX V7.1 introduce the 1 TB Segment Aliasing. 1 TB segments can improve the performance of 64-bit large memory applications. The optimization is specific to large shared memory (*shmat()* and *mmap()*) regions.

1 TB segments are a feature present in POWER5+, POWER6, and POWER7 processors. They can be used to reduce the hardware virtual to real translation impact. Applications that are 64-bit and that have large shared memory regions can benefit from incorporating 1 TB segments.

An overview of 1 TB segment usage can be found in the *IBM AIX Version 7.1 Differences Guide*, SG24-7910.

For more information about this topic, see 4.4, “Related publications” on page 94.

64-bit versus 32-bit Application Binary Interfaces

AIX provides complete support for both 32-bit and 64-bit Application Binary Interface (ABIs). Applications can be developed using either ABI with some performance trade-offs. The 64-bit ABI provides more scaling benefits. With both ABIs, there are performance trade-offs to be considered.

Overview of 64-bit/32-bit Application Binary Interface

All current POWER processors support a 32-bit and 64-bit execution mode. The 32-bit execution mode is a subset of the 64-bit execution mode. The modes are similar, where the most significant difference is addresses in address generation (effective addresses are truncated to 32 bits) and computation of some fixed-point status registers (carry, overflow, and so on). Although hardware 32-bit/64-bit mode does not affect performance, the 32-bit/64-bit ABIs provided by AIX do have performance implications and tradeoffs.

The 32-bit ABI provides an ILP32 model (32-bit integers, longs, and pointers) while the 64-bit ABI provides an LP64 model (32-bit integer and 64-bit longs/pointers). Although current POWER CPUs have 64-bit fixed-point registers, they are treated as 32-bit fixed-point registers by the ABI (the high 32 bits of all fixed-point registers are treated as volatile or undefined by the ABI). The 32-bit ABI preserves only 32-bit fixed-point context across subroutine linkage, non-local goto (`longjmp()`), or signal delivery. 32-bit programs cannot attempt to use 64-bit registers when they run in 32-bit mode (32-bit ABI). In general, other registers (floating point, vector, and status registers) are the same size in both 32-bit/64-bit ABIs.

Starting with AIX V6.1 all supervisor code (kernel, kernel extensions, and device drivers) uses the 64-bit ABI. In general, a unified system call interface is provided to applications that provides efficient system call linkage to both 32-bit and 64-bit applications. Because the AIX V 6.1 kernel is 64-bit, it implies that all systems supported by AIX V 6.1 support the 64-bit ABI. Some older IBM PowerPC® CPUs supported on AIX 5L V 5.3 cannot run the 64-bit ABI.

Operating system libraries provide both 32-bit and 64-bit objects, allowing full support for either ABI. Development tools (assembler, linker, and debuggers) support both ABIs.

Trade-offs

The primary motivation to choose the 64-bit ABI is to go beyond the 4 GB directly memory addressability barrier. A second reason is to improve scalability by extending some 32-bit data type limits that are in the 32-bit ABI (`time_t`, `pid_t`, and `offset_t`). Lastly, 64-bit mode provides access to 64-bit fixed-point registers and instructions that can improve the performance of specific fixed-point operations (long long arithmetic and 64-bit memory copies).

The 64-bit ABI does have some performance drawbacks, such as the 64-bit fixed-point registers and the LP64 model grow stack usage and data structures. These items can cause a performance drawback for some applications. Also, 64-bit text is generally larger for most compiles, producing a larger i-cache footprint.

The most significant issue is typically the porting effort (for existing applications), as changing between ILP32 and LP64 normally requires a port. Large memory addressability and scalability are normally the deciding factor when you chose an application execution model.

For more information about this topic, see 4.4, “Related publications” on page 94.

Affinity APIs

Most applications must be bound to logical processors to get a performance benefit from memory affinity to prevent the AIX dispatcher from moving the application to processor cores in different Multi-chip Modules (MCMs) while the application runs.

The most likely way to obtain a benefit from memory affinity is to limit the application to running only on the processor cores that are contained in a single MCM. You can accomplish this task by running the **bindprocessor** command and the **bindprocessor()** function. It can also be done with the resource set affinity commands (**rset**) and service applications. Often, affinity is provided as an administrator option that can be optionally enabled on large systems.

When the application requires more processor cores than contained in a single MCM, the performance benefit through memory affinity depends on the memory allocation and access patterns of the various threads in the application. Applications with threads that individually allocate and reference unique data areas might see improved performance.

The AIX Active System Optimizer (ASO) facility is capable of autonomously establishing enhanced affinity (see 4.2, “AIX Active System Optimizer and Dynamic System Optimizer” on page 84). This situation is in contrast to the manual usage of the affinity APIs documented in this section.

Processor affinity (bindprocessor)

Processor affinity is the probability of dispatching of a thread to the logical processor that was previously running it. If a thread is interrupted and later redispached to the same logical processor, the processor's cache might still contain lines that belong to the thread. If the thread is dispatched to a different logical processor, it probably experiences a series of cache misses until its cache working set is retrieved from RAM or the other logical processor's cache. If a dispatchable thread must wait until the logical processor that it was previously running on is available, the thread might experience an even longer delay.

The highest possible degree of processor affinity is to bind a thread to a specific logical processor. Binding means that the thread is dispatched to that logical processor only, regardless of the availability of other logical processors.

The **bindprocessor** command and the **bindprocessor()** subroutine bind the thread (or threads) of a specified process to a particular logical processor. Explicit binding is inherited through **fork()** and **exec()** system calls. The **bindprocessor** command requires the process identifier of the process whose threads are to be bound or unbound, and the bind CPU identifier of the logical processor to be used.

While CPU binding is useful for CPU-intensive applications, it can sometimes be counter productive for I/O-intensive applications.

RSETS

Every process and kernel thread can have an RSET attached to it. The CPUs on which a thread can be dispatched are controlled by a hierarchy of resource sets. RSETS are mandatory bindings and are honored by the AIX kernel always. Also, RSETS can affect Dynamic Reconfiguration (DR) activities.

Resource sets

These resource sets are:

Thread effective RSET	Created by <code>ra_attachrset()</code> . Must be a subset (improper or proper) of “Other RSETS” on page 76.
Thread partition RSET	Used by WLM. Partition RSETS allow WLM to limit where a thread can run.
Process effective RSET	Created by <code>ra_attachrset()</code> , <code>ra_exec()</code> , and <code>ra_fork()</code> . Must be a subset (improper or proper) of the process partition RSET.
Process partition RSET	Used by WLM to limit where processes in a WLM class are allowed to run. Can also be created by root users using the <code>rs_setpartition()</code> service.

Other RSETS

Another type of RSET is the exclusive RSET. Exclusive use processor resource sets (XRSETS) allow an installation to limit the usage of the processors in XRSETS; they are used only by work that is attached to those XRSETS. They can be created by running the `mkrset` command in the 'sysxrset' namespace.

RSET data types and operations

The public shipped header file `rset.h` contains declarations for the public RSET data types and function prototypes.

An RSET is an opaque data type. Applications allocate an RSET by calling `rs_alloc()`. Applications receive a *handle* to the RSET. The RSET handle (datatype `rsethandle_t` in `sys/rset.h`) is then used in RSET APIs to manipulate or attach the RSET.

Summary of RSET commands

Here is a summary of the RSET commands:

- ▶ **lsrset**: Displays RSETS stored in the system registry or RSETS attached to a process. For example:

<code>lsrset -av</code>	Displays all RSETS in the system registry.
<code>lsrset -p 28026</code>	Displays the effective RSET attached to PID 28026.
- ▶ **mkrset**: Makes a named RSET containing specific CPU and memory pools and place the RSET in the system registry. For example, `mkrset -c 6 10-12 test/lotsofcpus` creates an RSET named `test/lotsofcpus` that contains the specified CPUs.
- ▶ **rmrset**: Removes an RSET from the system registry. For example:

```
rmrset test/lotsofcpus
```
- ▶ **attachrset**: Attaches an RSET to a specified PID. The RSET can either be in the system registry or CPUs or mempools that are specified in the command. For example:

<code>attachrset test/lotsofcpus 28026</code>	Attaches an RSET in a register to a process.
<code>attachrset -c 4-8 28026</code>	Attaches an RSET with CPUs 4 - 8 to a process as an effective RSET.
<code>attachrset -P -c 4-8 28026</code>	Attaches an RSET with CPUs 4 - 8 to process as a partition rset.
- ▶ **detachrset**: Detaches an RSET from a specified PID. For example:

<code>detachrset 28026</code>	Detaches an effective RSET from a PID.
<code>detachrset -P 20828</code>	Detaches a partition RSET from a PID.

- ▶ **execrset:** Runs a specific program or command with a specified RSET. For example:
execrset sys/node.04.00000 -e test Runs a program test with an effective RSET from the system registry.
- execrset -c 0-1 -e test2** Runs program test2 with an effective RSET that contains logical CPU IDs 0 and 1.
- execrset -P -c 0-1 -e test3** Runs program test3 with a partition RSET that contains logical CPU IDs 0 and 1.

RSET manipulation and information services

This list contains only user space APIs. There are also similar kernel extension APIs. For example, **krs_alloc()** is the kernel extension equivalent to **rs_alloc()**.

- rs_alloc()** Allocates and initializes an RSET and returns an RSET handle to a caller.
- rs_free()** Frees a resource set. The input is an RSET handle.
- rs_init()** Initializes a previously allocated RSET. The initialization options are the same as for **rs_alloc()**.
- rs_op()** Performs one of a set of operations against one or two RSETS.
- rs_getinfo()** Get information about an RSET.
- rs_getrad()** Get resource allocation domain information from an input RSET.
- rs_numrads()** Returns the number of system resource allocation domains at the specified system detail level that have available or online resources.
- rs_getpartition()** Gets a process's partition RSET.
- rs_setpartition()** Sets a process's partition RSET.
- rs_discardname()**
- rs_getnameattr()**
- rs_getnamedrset()**
- rs_setnameattr()**
- rs_registername()** These are services that are used to manage the RSET system registry. There are services to create, obtain, and delete RSETS in the registry.

Attachment services

Here are the RSET attachment services:

- ra_attachrset()** A service to attach a work component to an RSET. The service uses the **rstype_t** and **rsid_t** parameters to identify the work component to attach to the input RSET (specified by an **rsethandle_t**).
- ra_detachrset()** Detaches an RSET from the work unit that is specified by the **rstype_t/rsid_t** parameters.
- ra_exec()** Runs a program that is attached to a specific work component. The service uses **rstype_t** and **rsid_t** to specify the work component. However, the only supported **rstype_t** is **R_RSET**. All of the various versions of **exec()** are supported.

<code>ra_fork()</code>	Forks a process that is attached to a specific work component. The service uses <code>rstype_t</code> and <code>rsid_t</code> to specify the work component. However, the only supported <code>rstype_t</code> is <code>R_RSET</code> .
<code>ra_get_attachinfo()</code> <code>ra_free_attachinfo()</code>	These services retrieve the RSET attachments that are attached to a memory range. The <code>ra_attachrset()</code> allows RSET attachments to ranges of memory in a file or shared memory segment. These services allow those attachments to be queried.
<code>ra_getrset()</code>	Retrieves the RSET attachment to a process or thread. The return code indicates where the returned RSET is attached.
<code>ra_mmap()</code> and <code>ra_mmapv()</code>	Maps a file or memory region into a process and attaches it to the resource set that is specified by the <code>rstype_t</code> and <code>rsid_t</code> parameters. A memory allocation policy similar to <code>ra_attachrset()</code> allows a caller to specify how memory is preferentially allocated when the area is accessed.
<code>ra_shmget()</code> and <code>ra_shmgetv()</code>	Gets a shared memory segment with an attachment to a resource set. The RSET is specified by the <code>rstype_t</code> and <code>rsid_t</code> parameters. A memory allocation policy similar to <code>ra_attachrset()</code> allows a caller to specify how memory is preferentially allocated when the area is accessed.

AIX Enhanced Affinity (Scheduler Resource Allocation Domain)

AIX Enhanced Affinity is a collection of AIX internal system changes and API extensions to improve performance on POWER7 Systems. Enhanced Affinity improves performance by increasing CPU and memory locality on POWER7 Systems. Enhanced Affinity extends the AIX existing memory affinity support. AIX V6.1 technology level 6100-05 contains AIX Enhanced Affinity support.

Enhanced Affinity status is determined during system boot and remains unchanged for the life of the system. A reboot is required to change the Enhanced Affinity status. In AIX V6.1.0 technology level 6100-05, Enhanced Affinity is enabled by default on POWER7 machines. Enhanced Affinity is available only on POWER7 machines. Enhanced Affinity is disabled by default on POWER6 and earlier machines. A `vmo` command tunable (`enhanced_memory_affinity`) is available to disable Enhanced Affinity support on POWER7 machines.

Here are two concepts that are related to Enhanced Affinity:

- ▶ Scheduler Resource Allocation Domain (SRAD): SRAD is the collection of logical CPUs and physical memory resources that are close from a hardware affinity perspective. An AIX system (partition) can consist of one or more SRADs. An SRAD represents the same collection of system resources as an existing MCM. A specific SRAD in a partition is identified by a number. It is an `sradit_t` data type and is often referred to as an SRADID.
- ▶ SRADID: The numeric identifier of a specific SRAD. It is a short integer data type. An SRADID value is the index of the resource allocation domain at the `R_SRADSDL` system detail level in the system's resource set topology.

Power Systems before POWER7 Systems provided only system topology information to dedicated CPU logical partitions. This setup limited the usefulness of RSET attachments for CPU and memory locality purposes to dedicated CPU partitions. POWER7 Systems provide system topology information for shared CPU logical partitions (SPLPAR).

You can use the AIX Enhanced Affinity services to attach SRADs to threads and memory ranges so that the application preferentially identifies the logical CPUs or physical memory to use to run the application. AIX continues to support RSET attachments to identify resources for an application.

RSET versus SRADs

When you compare RSET with SRADIDs:

1. SRADIDs can be attached to threads, shared memory segments, memory map regions, and process memory subranges. SRADIDs may not be attached at the process level (R_PROCESS). SRADIDs may not be attached to files (R_FILDES).
2. SRADID attachments are considered advisory. There are no mandatory SRADID attachments. AIX might ignore advisory SRADID attachments.
3. Process and thread RSET attachments continue to be mandatory. The process and thread resource set hierarchy continues to be enforced. Memory RSET attachments (shared memory, file, and process subrange) continue to be advisory. This situation is unchanged from previous affinity support.

API support

SRADIDs can be attached to threads and memory by using the following functions:

- ▶ `ra_attach()` (new)
- ▶ `ra_fork()`
- ▶ `ra_exec()`
- ▶ `ra_mmap()` and `ra_mmapv()`
- ▶ `ra_shmget()` and `ra_shmgetv()`

SRADIDs can be detached from thread and memory by using the `sra_detach()` function (new).

Hybrid thread and core

AIX provides facilities to customize simultaneous multi-threading (SMT) characteristics of CPUs running within a partition. The features require some partition-wide CPU configuration options, so their use is limited to specific workloads.

Background

SMT is a feature that is introduced in POWER5 and capitalized on by AIX. It allows a single physical processor core to simultaneously dispatch instructions from more than one hardware thread context. An overview of SMT is provided in the AIX SMT Overview.²

SMT does include some performance tradeoffs:

- ▶ SMT can provide a significant throughput and capacity improvement on POWER processors. When you are in SMT mode, there is a trade-off between overall CPU throughput and performance of each hardware thread. SMT allows multiple instruction streams to be run simultaneously, but the concurrency can cause some resource conflict between the instruction streams. This conflict can result in a decrease in performance for an individual thread, but an increase in overall throughput.
- ▶ Some workloads do not run well with the SMT feature. This situation is not typical for commercial workloads, but has been observed with scientific (floating point intensive) workloads.

² *Simultaneous Multithreading*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.genprog/doc/genprog/smt.htm>

AIX provides options to allow SMT customization. The `smtctl` option allows the SMT feature to be enabled, disabled, or capped (SMT2 versus SMT4 mode on POWER7). The partition-wide tuning option, `smtctl`, changes the SMT mode of all processor cores in the partition. It is built on the AIX DR (dynamic reconfiguration) framework to allow hardware threads (logical processors) to be added and removed in a running partition. Because of this option's global nature, it is normally set by system administrators. Most AIX systems (commercial) use the default SMT settings enabled (that is, SMT2 mode on POWER5 and POWER6, and SMT4 mode on POWER7).

When SMT is enabled (SMT2 or SMT4 mode), the AIX kernel takes advantage of the platform feature to dynamically change SMT modes. These mode switches are done based on system load (the number of running or waiting to run software threads) to choose the optimal SMT mode for the CPUs in the partition. The mode switching policies optimize overall workload throughput, but do not attempt to optimize individual software threads.

Hybrid thread features

AIX provides some basic features that allow more control in SMT mode. With these features, specific software threads can be bound to hardware threads assigned to ST mode CPUs. This configuration allows for an asymmetric SMT configuration where some CPUs are in high SMT mode, while others have SMT mode disabled. This configuration allows critical software threads within a workload to receive an ST performance boost, while it allows the remaining threads to benefit from SMT mode. Typical reasons to take advantage of this hybrid mode are:

- ▶ Asymmetric workload, where the performance of one thread serializes an entire workload. For example, one master thread dispatches work to many subordinate threads.
- ▶ Software threads that are critical to a system administrator.

The ability to create hybrid SMT configurations is limited under current AIX releases and does require administrator or privileged configuration changes. CPUs that provide ST mode hardware threads must be placed into exclusive processor resource sets (XRSETs). XRSETs contain logical CPUs that are segregated from the general kernel dispatching pool. Software threads must be explicitly bound to CPUs in an XRSET. The only way to create an XRSET is by running the `mkrset` command. All of the hardware threads for logical CPUs must be contained in the XRSET created RSET. To accomplish this task, run the following commands:

lsrset -av	Displays the RSET topology. The system CPU topology is broken down into a hierarchy that has the form <code>sys/node.XX.YYYYY</code> . The largest XX value is the CPU (core) level. This command provides logical processor groups by core.
mkrset -c 4-7 sysxrset/set1	Creates an XRSET <code>sysxrset/set1</code> containing logical CPUs 4 - 7.

An XRSET alone can be used to ensure that only specific work uses a CPU set. There is also the ability to restrict work execution to primary threads in an XRSET. This ability is known as an STRSET. STRSETs allow software threads to use ST execution mode independently of the load on the other CPUs in the system. Work can be placed onto STRSETs by running the following commands:

execrset -S	This command allows external programs to start and be bound to an exclusive RSET.
ra_attach(R_STRSET)	This API allows a thread to be bound to an STRSET.

For more information about this topic, see 4.4, "Related publications" on page 94.

Sleep and wake-up primitives (`thread_wait` and `thread_post`)

AIX provides proprietary `thread_wait()` and `thread_post()` APIs that can be used to optimize thread synchronization and communication (IPC) operations. AIX also provides several standard APIs that can be used for thread synchronization and communication. These APIs include `pthread_cond_wait()`, `pthread_cond_signal()`, and `semop()`. Although many applications use these standard APIs, the low-level primitives are available to optimize these operations. `thread_wait()` and `thread_post()` can be used to optimize critical applications services, such as user-mode locking or message passing. They are more efficient than the portable/standard APIs.

Here is more information about the associated subroutines:

► `thread_wait`

The `thread_wait` subroutine allows a thread to wait or block until another thread posts it with the `thread_post` or the `thread_post_many` subroutine or until the time limit specified by the timeout value expires.

If the event for which the thread is waiting and for which it is posted occurs only in the future, the `thread_wait` subroutine can be called with a timeout value of 0 to clear any pending posts by running the following command:

```
thread_wait (timeout)
```

► `thread_post`

The `thread_post` subroutine posts the thread whose thread ID is indicated by the value of the `tid` parameter, of the occurrence of an event. If the posted thread is waiting in `thread_wait`, it is awakened immediately. If it is not waiting in `thread_wait`, the next call to `thread_wait` is not blocked, but returns with success immediately.

Multiple posts to the same thread without an intervening wait by the specified thread counts only as a single post. The posting remains in effect until the indicated thread calls the `thread_wait` subroutine, upon which the posting is cleared.

► `thread_post_many`

The `thread_post_many` subroutine posts one or more threads of the occurrence of the event. The number of threads to be posted is specified by the value of the `nthreads` parameter, while the `tidp` parameter points to an array of thread IDs of threads that must be posted. The subroutine works just like the `thread_post` subroutine, but can be used to post to multiple threads at the same time. A maximum of 512 threads can be posted in one call to the `thread_post_many` subroutine.

For more information about this topic, see 4.4, “Related publications” on page 94.

Shared versus private loads

You can use AIX to share text for libraries and dynamically loaded modules. File permissions can be used to enable and disable sharing of loaded text.

Documentation

AIX provides optimizations that enable sharing of loaded text (libraries and dynamically loaded modules). Sharing text among processes often improves performance because it reduces resource usage (memory and disk space). It also allows unrelated software-threads to share cache space when they run concurrently. Lastly, it can reduce load times when the code is already loaded by a previous program.

Applications can control if private or shared loads are performed to shared text regions. Shared loads require that execute permissions be set for group/other on the text files. As a preferred practice, you should enable sharing.

For more information about this topic, see 4.4, “Related publications” on page 94.

Workload partitions (WPAR shared LPP installs)

Starting with AIX V6.1, the WPAR feature gives the system administrator the ability to easily create an isolated AIX operating system that can run services and applications. WPAR provides a secure and isolated environment for enterprise applications in terms of process, signal, and file system space. Any software that is running within the context of a workload partition appears to have its own separate instance of AIX.

The usage of multiple virtual operating systems within a single global operating environment can have multiple advantages. It increases administrative efficiency by reducing the number of AIX instances that must be maintained.

Applications can be installed in a shared environment or a non-shared environment. When an application is installed in a shared environment, it means that it is installed in the global environment and then the application is shared with one or more WPARs. When an application is installed in a non-shared environment, it means that it is installed in the WPAR only. Other WPARs do not have access to that application.

Shared WPAR installation

A shared installation is straightforward because installing software in the global environment is accomplished in the normal manner. What must be considered is whether the system WPARs that share a single installation will or will not interfere with each other’s operation.

For software to function correctly in a shared-installation environment, the software package must be split into shareable and non-shareable files:

- ▶ Shareable files (such as executable code and message catalogs) must be installed into the shared global file systems that are read-only to all system WPARs.
- ▶ Non-shareable files (such as configuration and runtime-modifiable files) must be installed into the file systems that are writable to individual WPARs. This configuration allows multiple WPARs to share a single installation, yet still have unique configuration and runtime data.

In addition to splitting the software package, the software installation process must include a synchronization step to install non-shareable files into system WPARs. To accomplish this task, the application must provide a means to encapsulate the non-shareable files within the shared global file systems so that the non-shared files can be extracted into the WPAR by some means. For example, if a vendor creates a custom-installation system that delivers files into `/usr` and `/`, then the files that are delivered into `/` must be archived within `/usr` and then extracted into `/` using some vendor-provided mechanism. This action can occur automatically the first time that the application is started or configured.

Finally, the software update process must work so that the shareable and non-shareable files stay synchronized. If the shared files in the global AIX instance are updated to a certain fix level, then the non-shared files in individual WPARs also must be updated to the same level. Either the update process discovers all the system WPARs that must be updated or, at start time, the application detects the out-of-synchronization condition and applies the update. Some software products manage to never change their non-shareable files in their update process, so they do not need any special handling for updates.

This type of installation sometimes takes a little effort on the part of the application, but it allows you to get the most value from using WPARs. If there is a need to run the same version of the software in several WPARs, this type of installation provides the following benefits:

- ▶ It increases administrative efficiency by reducing the number of application instances that users must maintain. The administrator saves time in application-maintenance tasks, such as applying fixes and performing backups and migrations.
- ▶ It allows users to quickly deploy multiple instances of the same application, each in its own secure and isolated environment. It can take only a matter of minutes to create and start a WPAR to run a shared installation of the application.
- ▶ By sharing one AIX or application image among multiple WPARs, the memory resource usage is reduced because only one copy of the application image is in real memory.

For more information about WPAR, see *WPAR concepts*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.wpar/wpar-overview.htm>

4.1.2 Using POWER7+ features under AIX

When the AIX operating system runs on POWER7+ processors, it transparently uses the POWER7+ on-chip encryption accelerators. For each of the uses that are described in this section, there are no application visible changes or awareness required.

AIX encrypted file system (EFS)

Integrated with the AIX Journaled File System (JFS2) is the ability to create an encrypted file system (EFS) where all data at rest in the file system is encrypted. When AIX EFS runs on POWER7+, it uses the encryption accelerators, which can show up to a 40% advantage in file system I/O-intensive operations. Applications do not need to be aware of this situation, but application and workload deployments might be able to take advantage of higher levels of security by using AIX EFS for sensitive data.

AIX Internet Protocol Security (IPSec)

When IPSec is enabled on AIX running on POWER7+, AIX transparently uses the POWER7+ encryption accelerators for all data in transit. The advantage that is provided by the accelerators is more pronounced when jumbo frames (a maximum transmission unit (MTU) of 9000 bytes) are used. Applications do not need to be aware of this situation, but application and workload deployments might be able to take advantage of higher levels of security by enabling IPSec.

AIX /dev/random (random number generation)

AIX capitalizes on the on-chip random number generator on POWER7+. Applications that use the AIX special files `/dev/random` or `/dev/urandom` transparently get the advantages of stronger hardware-based random numbers. If an application is making high frequency usage of random number generation, there may also be a performance advantage.

AIX PKCS11 Library

On POWER7+ systems, the AIX operating system PKCS11 library transparently uses the POWER7+ on-chip encryption accelerators. For an application using the PKCS11 APIs, no change or awareness by the application is required. The AIX library interfaces dynamically decides, based on the algorithm and data size, when to use the accelerators. Because of the cost of setup and programming of the on-chip accelerators, the advantage is limited to operations on large blocks of data (tens to hundreds of kilobytes).

4.2 AIX Active System Optimizer and Dynamic System Optimizer

Workloads are becoming increasingly complex. Typically, they involve a mix of single-threaded and multi-threaded applications with interactions that are complex and vary over time. The servers that host these workloads are also continuously evolving to support an ever-increasing demand for processing capacity and flexibility. Tuning such an environment for optimal performance is not trivial. It often requires excessive amounts of time and highly specialized skills. Apart from resources, manual tuning also has the drawback that it is static in nature, and systems must be retuned when new workloads are introduced or when the characteristics of existing ones change in time. The Active System Optimizer (ASO) and Dynamic System Optimizer (DSO) attempt to address the optimization of both the operating system and server autonomously.

4.2.1 Concepts

DSO is built on the Active System Optimizer (ASO) framework, and expands on two of the ASO optimization strategies.

Active System Optimizer

DSO is built on the ASO framework that is introduced in AIX V7.1 TL1 SP1. The ASO framework includes a user-space daemon, advanced instrumentation, and two optimization strategies. The DSO package extends ASO to provide more optimizations. When the DSO product is installed and enabled, base ASO features and the extended DSO optimizations are all activated.

ASO contains a user-level daemon that autonomously tunes the allocation of system resources to achieve an improvement in system performance. ASO is available on the POWER7 platform in AIX V7.1 TL1 SP1 (4Q 2011) and AIX V6.1 TL8 SP1 (4Q 2012). DSO extensions are available in 4Q 2012, and require AIX V7.1 TL2 SP1 or AIX V6.1 TL8 SP1 (on the POWER7 platform).

The ASO framework works by continuously monitoring and analyzing how current workloads impact the system, and then using this information to dynamically configure the system to optimize for current workload requirements.

The ASO framework is transparent, and the administrator is not required to continuously monitor its operations. In fact, the only required tunable that ASO provides is to turn it on or off. Once turned on, ASO automatically identifies opportunities to improve performance and applies the appropriate system changes.

The ASO daemon has a maximum system usage impact of 3%. ASO also monitors the system for situations that are not suitable for certain types of performance optimization. In such situations, ASO *hibernates* those optimization strategies, waking up occasionally to check for a change in environment (for more information about this topic, see 4.2.3, “Workloads” on page 87 and “System requirements” on page 90).

The primary design goal of ASO/DSO is to act only when it is reasonably certain that the result is an improvement in workload performance.

Figure 4-1 illustrates the basic ASO framework. ASO uses information from the AIX kernel and the POWER7 Performance Monitoring Unit (PMU)^{3, 4, 5} to perform long-term runtime analysis with the aim of improving workload performance.

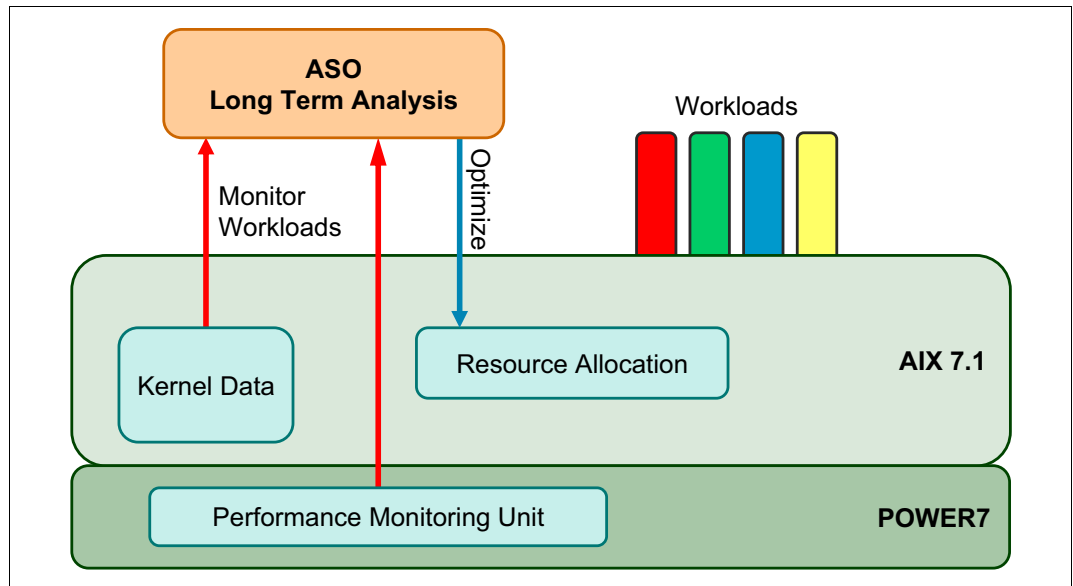


Figure 4-1 Basic ASO architecture that shows an optimization flow on a POWER7 system

Optimization strategies

Two optimization strategies are provided with ASO:

- ▶ Cache affinity optimization
- ▶ Memory affinity optimization

DSO adds two more optimizations to the ASO framework:

- ▶ Large page optimization
- ▶ Memory prefetch optimization

The first version that was released in 4Q 2011 included the cache and memory affinity optimizations. The 4Q 2012 version introduces the large page and data stream prefetch optimization types.

³ *Commonly Used Metrics For Performance Analysis*, available at: <http://www.power.org/documentation/commonly-used-metrics-for-performance-analysis/> (registration required)

⁴ *Comprehensive PMU Event Reference - POWER7*, available at: <http://www.power.org/documentation/comprehensive-pmu-event-reference-power7/> (registration required)

⁵ *Hardware performance monitor APIs and tools*, available at: http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_monitor.htm

ASO and DSO optimizations: ASO cache and memory affinity optimizations are bundled with all AIX editions: AIX Express, AIX Standard, and AIX Enterprise. DSO large page and memory prefetch optimizations are available as a separately chargeable premium feature or bundled with AIX Enterprise.

4.2.2 ASO and DSO optimizations

The ASO framework allows multiple optimizations to be managed. Two optimizations are included with the framework. Two more optimizations are added with the DSO package.

Cache and memory affinity optimization

Power Systems are continually increasing in processing capacity in terms of the number of cores and the number of SMT threads per core. The latest Power Systems support up to 256 cores and four SMT threads per core, which allows a single logical partition (LPAR) to have 1024 logical CPUs (hardware threads). This increase in processing units has led to a hierarchy of affinity domains.

Each core forms the smallest affinity domain. Multiple cores in a chip, multiple chips in a package, and the system itself form other higher affinity domains. System performance is close to optimal when the data that is crossing between these domains is minimal. The need for cross-domain interactions arises because of either:

- ▶ Software threads in different affinity domains must communicate
- ▶ Data being accessed that is in a memory bank that is not in the same affinity domain as the one of the requesting software thread

Apart from the general eligibility requirements that are listed in “System requirements” on page 90, a workload must also be multi-threaded to be considered for cache and memory affinity optimization.

Cache affinity

ASO analyzes the cache access patterns that are based on information from the kernel and the PMU to identify potential improvements in cache affinity by moving threads of workloads closer together. If such a benefit is predicted, ASO uses proprietary algorithms to estimate the optimal size of the affinity domain for the workload, and it uses kernel services (see “Affinity APIs ” on page 75) to restrict the workload to that domain. After acting, ASO continues to monitor the workload to ensure that it performs as predicted. If the results are not as expected, ASO reverses its actions immediately.

Memory affinity

After a workload is identified and optimized for cache affinity, ASO begins monitoring the memory access patterns of the workload process private memory. If it is found that the workload can benefit from moving process private memory closer to the current affinity domain, then hot pages are identified and migrated closer using software instrumentation. Single-threaded processes are not considered for this optimization, because their process private data is already affinityized by the kernel when the thread is moved to a new affinity domain. Also, in the current version, only workloads that fit within a single Scheduler Resource Affinity Domain (SRAD, a chip/socket in POWER7) are considered.

Large page optimization

AIX allows translations of multiple memory page sizes within the same segment. Although 4 KB and 64 KB translations are allowed in the current version of AIX (Version 6.1 and greater), Version 6.1 TL8 and Version 7.2 TL2 (4Q 2012) include dynamic 16 MB translation. For workloads that use large chunks of data, using pages larger than the default size is useful because the number of TLB/ERAT misses is reduced (For information about general page size information and TLB/ERAT, see 2.3.1, “Page sizes (4 KB, 64 KB, 16 MB, and 16 GB)” on page 25). DSO uses this new AIX feature to promote heavily used regions of memory to 16 MB pages dynamically, potentially improving the performance of workloads that use those regions.

System V shared memory: In the current version, only System V shared memory is eligible for dynamic 16 MB page optimization.

Memory prefetch optimization

Power Architecture provides a special purpose register (the DCSR) to control the enablement, depth, and stride for hardware data stream prefetching (for more information, see 2.3.7, “Data prefetching using d-cache instructions and the Data Streams Control Register (DCSR)” on page 46). Setting this register appropriately can potentially benefit workloads, depending on the workload data access patterns. DSO collects information from the AIX kernel and PMU to dynamically determine the optimal setting of this register for a specific period.

4.2.3 Workloads

For ASO/DSO to consider a workload for optimization, the workload must pass certain minimum criteria, as described in this section.

Ideal workloads

Workload characteristics for each optimization are:

- ▶ Cache affinity optimization and memory affinity optimization: Workloads must be long-lived (the minimum lifetime varies with the type of optimization), multi-threaded, and have stable CPU usage. The performance gain is higher for workloads that have a high amount of communication between the threads in the workload.
- ▶ Large page optimization: Workloads must use large System V memory regions, for example, a database with a large shared memory region. Workloads can be either multi-threaded or a group of single-threaded processes. DSO must be active when a workload is started for this optimization to be applied.
- ▶ Memory prefetch optimization: Workloads must have a large System V memory footprint, high CPU usage, and a high context switch rate. Workloads can be either multi-threaded or a group of single-threaded processes. This optimization is disabled if the DCSR register is set manually at the system level (through the `dscrct1` command).

Eligible workloads

For ASO/DSO to consider a workload for optimization, it must pass certain minimum criteria. These criteria are:

- ▶ General requirements
 - Fixed priority. The ASO daemon runs with a fixed scheduler priority of 32. ASO does not optimize a workload if it or any of its threads has a fixed priority more favorable (numerically lower) than itself.

► Cache and memory affinity optimization

– Multi-threaded

Workloads must be multi-threaded to be considered for optimization.

– Workload Manager

Workloads that are classified by the Workload Manager (WLM) with tiers or minimum limits set are not optimized. Furthermore, if the system CPU capacity is fully used, ASO does not optimize processes that belong to classes with specific shares.

WPAR workloads: In general, WPAR workloads (which implicitly use WLM) can be optimized by ASO if minimum CPU and memory limits are not specified.

– User-specified placement

Workloads for which placement is explicitly set by the user, such as with `bindprocessor`, `RSET` attachments (real, partition, or exclusive `RSETs`), and `SRAD` attachments, are not eligible for ASO optimization. Although ASO does not affect these workloads, AIX continues to enforce the resource constraints as normal. Furthermore, if the user attempts to place such a restriction on a workload that has been or is being optimized by ASO, then ASO undoes its optimization and lets the user restriction be placed normally.

– CPU usage

The CPU usage of the workload should be above 0.1 cores.

– Workload age

Workloads must be at least 10 seconds of age to be considered for cache affinity and 5 minutes of age for memory affinity optimization.

► Large page optimization

– Fully populated segments

The shared memory segments should be fully populated to be considered for page size promotion.

– Memory footprint

The workload memory footprint should have at least 16 GB of System V shared memory.

– CPU usage

CPU usage of the workload should be above two cores. A workload may be either a multi-threaded process or a collection of single-threaded processes.

– Workload age

Workloads must be at least 60 minutes of age to be considered.

► Memory prefetch optimization

– Memory footprint

The workload memory footprint should have at least 16 GB of System V shared memory.

– CPU usage

CPU usage of the workload should be above eight cores. A workload may be either a multi-threaded process or a collection of single-threaded processes.

- Workload age

Workloads must be at least 10 minutes of age to be considered.

Optimization time

When you test the effect of DSO on applications, it is important to run the tests for enough time. The duration depends on the type of optimization that is being measured. For example, in the case of large page optimization, there is a small increase in system usage (less than 2%) when the pages are being promoted. So, to see the benefit of large page optimization, the test must be run for longer than the time taken to complete the promotions.

Here is the list of minimum test durations for each type of optimization. All of these times are approximate, and only after the workload is stable.

Cache affinity optimization	30 minutes
Memory affinity optimization	1 hour
Large page optimization	12 hours for promoting 100 GB of shared memory
Memory prefetch optimization	30 minutes

4.2.4 The asoo command

The ASO framework is off by default in an AIX installation. The **asoo** command must be used to enable the ASO framework. The command syntax is as follows:

```
asoo -po aso_active=1
```

4.2.5 Environment variables

Two environment variables are provided that control the behavior of ASO: **ASO_ENABLED** and **ASO_OPTIONS**. These variables are examined at start time.

ASO_ENABLED

ASO_ENABLED provides an administrator with the ability to alter the default behavior of ASO when you evaluate a workload for optimization.

Permitted values for this environment variable are:

- ▶ **ALWAYS**: ASO skips some of the primary eligibility checks for optimization eligibility, such as age of workload and minimum CPU usage.
- ▶ **NEVER**: ASO excludes this process from any optimization under all circumstances.
- ▶ **Unsupported value**: ASO optimizes the process as normal.

ASO_OPTIONS

ASO_OPTIONS provides an administrator with the ability to individually enable and disable cache, memory affinity, large page, and memory prefetch optimization.

Permitted values for this environment variable are shown in Table 4-2

Table 4-2 DSO ASO_OPTIONS environment variable

Option	Values	Effect
ALL	ON, OFF	Enables or disables all ASO optimization. Options are processed left to right; redundant or conflicting options use the rightmost setting.
CACHE_AFFINITY	ON, OFF	Enables or disables the cache affinity optimization.
MEMORY_AFFINITY	ON, OFF	Enables or disables the memory affinity optimization. The memory affinity optimization is applied only if the cache affinity optimization is also applied.
LARGE_PAGE	ON, OFF	Enables or disables 16 MB MPSS optimization.
MEMORY_PREFETCH	ON, OFF	Enables or disables data stream prefetch optimization.
<unset>	ON, OFF	All optimization is enabled.
<any other values>	ON, OFF	An undefined variable.

Multiple options can be combined, as in the following example:

```
$ export ASO_OPTIONS="ALL=OFF,CACHE_AFFINITY=ON"
```

System requirements

ASO/DSO optimizations can be limited based on system configuration. Here are the significant requirements and limitations:

- ▶ ASO and DSO takes advantage of a number of hardware features, and is supported only on POWER7 running in native mode.
- ▶ ASO and DSO is integrated tightly with the AIX kernel, so various optimizations are supported at different operating system levels. The minimum levels for cache and memory affinity optimization are AIX V7.1 TL1 SP1 or AIX V6.1 TL8. Memory prefetch and large page optimization require the installation of the DSO package that is supported on AIX V7.1 TL2 SP1 and AIX V6.1 TL8 SP1.
- ▶ If it is running in a dedicated processor environment, virtual processor management (core folding) must be disabled, which is the default. Enabling PowerSaver mode on the HMC causes virtual processor management in a dedicated environment to be enabled (forcing cache and memory affinity optimizations to be disabled).
- ▶ Enabling active memory sharing disables all optimizations except memory prefetch.
- ▶ Enabling active memory expansion disables memory affinity and large page optimization.
- ▶ In an SPLPAR environment, when CPU resources are capped, the system entitlement must be a minimum of two cores for all but memory prefetch. Memory prefetch requires eight cores.
- ▶ For large page and memory prefetch optimization, the system should have a minimum of 20 GB system memory.

4.2.6 Installing DSO

The AIX DSO is available as a separately chargeable premium package that includes the two new types of optimizations: Large page optimization and memory prefetch optimization. The package name is `dso.aso` and is installable using `installp` or `smitty`, as with any AIX package.

ASO detects the installation of this package automatically and enables memory prefetch and large page optimization. After installation, no restart of the operating system or ASO is required.

4.2.7 Log files

Information about the functioning of ASO and its various optimizations are logged in the following files:

- ▶ `/var/log/aso/aso.log`

This file lists major ASO events, such as when it is enabled or disabled or when it hibernates. It also contains a basic audit trail of optimizations that are performed to workloads. Example 4-1 shows a sample `aso.log` file.

Example 4-1 Sample aso.log file that lists major ASO events

```
# ASO log configuration
aso.notice /var/log/aso/aso.log rotate size 128k time 7d
aso.info /var/log/aso/aso_process.log rotate size 1024k
```

Example 4-2 shows a sample `aso.log` file that results from insufficient CPU entitlement on an SPLPAR.

Example 4-2 Sample aso.log file that shows insufficient CPU entitlement on an SPLPAR

```
Oct 20 02:15:04 p7e04 aso:notice aso[13238402]: [HIB] Current number of system
virtual cpus too low (1 cpus)
Oct 20 02:15:04 p7e04 aso:notice aso[13238402]: [HIB] Increase system virtual
cpus
to at least 3 cpus to run ASO. Hibernating.
```

- ▶ For virtualized environments, you can also use the IBM PowerVM Virtualization Performance Advisor. Instructions for the IBM PowerVM Virtualization Performance Advisor are available at:
<https://www.ibm.com/developerworks/wikis/display/WikiPtype/PowerVM+Virtualization+performance+advisor>
For more information, see “Virtualization Performance Advisor” on page 158.
- ▶ The number of online virtual CPUs of a single LPAR cannot exceed the number of active CPUs in a pool. See the output of `lparstat -i` from the LPAR to see the values for online virtual CPUs and active CPUs in pool.

4.3.2 AIX preferred practices that are applicable to POWER7

Here are the AIX preferred practices that are applicable to POWER7.

Preferred practices for installation and configuration

Preferred practices for installation and configuration are:

- ▶ To ensure that your system conforms to the minimum requirements, see Chapter 3, “The POWER Hypervisor” on page 55 and the references that are provided for that chapter (see 4.4, “Related publications” on page 94).
- ▶ Review the *POWER7 Virtualization Best Practice Guide*, available at:
https://www.ibm.com/developerworks/wikis/download/attachments/53871915/P7_virtualization_bestpractice.doc?version=1

For more information about this topic, see 4.4, “Related publications” on page 94.

4.3.3 POWER7 mid-range and high-end High Impact or Pervasive advisory

IBM maintains a strong focus on the quality and reliability of Power System servers. To maintain that reliability, the currency of microcode levels on your systems is critical. Therefore, apply the latest POWER7 system firmware and management console levels for your systems as soon as possible. These service pack updates contain a collective number of High Impact or Pervasive (HIPER) fixes that continue to provide you with the system availability you expect from IBM Power Systems.

Before you migrate to POWER7, you see more benefits if your AIX level contains the performance bundle set of APARS. Visit IBM Fix Central (<http://www.ibm.com/support/fixcentral/>) to download the latest service pack (SP) for your POWER7 Systems.

Furthermore, you should:

- ▶ Define an upgrade or update plan to bring your servers to the latest fix levels.
- ▶ Concurrently update servers that are running older firmware levels to the latest level.
- ▶ See the associated SP readme files for information about the specific fixes included in each service pack. Sections of readme files that describe more tips and considerations are a helpful resource as well.
- ▶ When you install firmware from the HMC, avoid the do not auto accept option. Selecting this advanced option can cause firmware installation problems.

You should subscribe to My Notifications⁶ to provide you with customizable communications that contain important news, new or updated support content, such as publications, hints, and tips, technical notes, product flashes (alerts), and downloads and drivers.

4.4 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *1 TB Segment Aliasing*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/1TB_segment_aliasing.htm
- ▶ *AIX 64-bit Performance in Focus*, SG24-5103
- ▶ *AIX Linking and Loading Mechanisms*, found at:
http://download.boulder.ibm.com/ibmdl/pub/software/dw/aix/es-aix_ll.pdf
- ▶ *Efficient I/O event polling through the pollset interface on AIX*, found at:
<http://www.ibm.com/developerworks/aix/library/au-pollset/index.html>
- ▶ *Exclusive use processor resource sets*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.baseadm/doc/baseadmndita/excluseprocreset.htm>
- ▶ *execrset command*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmds2/execrset.htm>
- ▶ *General Programming Concepts: Writing and Debugging Programs*, found at:
http://publib16.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixprggd/genprog/understanding_mem_mapping.htm
- ▶ *IBM AIX Version 7.1 Differences Guide*, SG24-7910
Refer to section 1.2, “Improved performance using 1 TB segments”
- ▶ *load and loadAndInIt Subroutines*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf1/load.htm>
- ▶ *mkrset command*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmds3/mkrset.htm>
- ▶ *Oracle Database and 1 TB Segment Aliasing*, found at:
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD105761>
- ▶ *pollset_create, pollset_ctl, pollset_destroy, pollset_poll, and pollset_query Subroutines*, found at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.basetechref/doc/basetrf1/pollset.htm>

⁶ For more information, go to
<ftp://ftp.software.ibm.com/systems/support/tools/mynotifications/overview.pdf>.

- ▶ *POWER7 Virtualization Best Practice Guide*, found at:
https://www.ibm.com/developerworks/wikis/download/attachments/53871915/P7_virtualization_bestpractice.doc?version=1
- ▶ *ra_attach Subroutine*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.basetechref/doc/basetrf2/ra_attach_new.htm
- ▶ *Shared library memory footprints on AIX 5L*, found at:
http://www.ibm.com/developerworks/aix/library/au-slib_memory/index.html
- ▶ *thread_post Subroutine*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/thread_post.htm
- ▶ *thread_post_many Subroutine*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/thread_post_many.htm
- ▶ *thread_wait Subroutine*, found at:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/thread_wait.htm
- ▶ *Thread environment variables*, found at:
https://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/thread_env_vars.htm



Linux

This chapter describes the optimization and tuning of the POWER7 processor-based server running the Linux operating system. It covers the following topics:

- ▶ Linux and system libraries
- ▶ Linux operating system-specific optimizations

5.1 Linux and system libraries

This section contains information about Linux and system libraries.

5.1.1 Introduction

When you work with IBM POWER7 processor-based servers, systems, and solutions, a solid choice for running enterprise-level workloads is Linux. Red Hat Enterprise Linux (RHEL) and SUSE Linux Enterprise Server (SLES) provide operating systems that are optimized and targeted for the Power Architecture. These operating systems run natively on the Power Architecture and are designed to take full advantage of the specialized features of Power Systems.

Both RHEL and SLES provide the tools, kernel support, optimized compilers, and tuned libraries for POWER7 Systems. The Linux distributions provide for excellent performance, and more application and customer-specific tuning approaches are available. IBM provides a number of value-add packages, tools, and extensions that provide for more tunings, optimizations, and products for the best possible performance on POWER7. The typical Linux open source performance tools that Linux users are comfortable with are available on the PowerLinux systems.

The Linux distributions are enabled to run on small Power Micro-Partitioning partitions through the broad range of IBM Power offerings, from low-cost PowerLinux servers and Flex System nodes, up through the largest IBM Power 770 and Power 795 servers.

IBM premier products, such as IBM XL compilers, IBM Java products, IBM WebSphere, and IBM DB2 database products, all provide Power optimized support with the RHEL and SUSE operating systems.

For more information about this topic, see 5.2, “Related publications” on page 106.

5.1.2 Linux operating system-specific optimizations

This section describes optimization methods specific to Linux.

GCC, toolchain, and IBM Advance Toolchain

This section describes 32-bit and 64-bit modes and CPU-tuned libraries.

Linux support for 32-bit and 64-bit modes

The compiler and runtime are fully capable of supporting either 32-bit or 64-bit mode applications simultaneously. The compilers can select the target mode through the `-m32` or `-m64` compiler options.

For the SUSE Linux Enterprise Server and Red Hat Enterprise Linux distributions, the shared libraries have both 32-bit and 64-bit versions. The toolchain (compiler, assembler, linker, and dynamic linker) selects the correct libraries based on the `-ms32` or `-m64` option or the mode of the application program.

The Advance Toolchain defaults to 64-bit, as do SLES 11 and RHEL 6. Older distribution compilers defaulted to 32-bit.

Applications can use 32-bit and 64-bit execution modes, depending on their specific requirements, if their dependent libraries are available for the wanted mode.

The 32-bit mode is lighter with a simpler function call sequence and smaller footprint for stack and C++ objects, which can be important for some dynamic language interpreters and applications with many small functions.

The 64-bit mode has a larger footprint because of the larger pointer and general register size, which can be an asset when you handle large data structures or text data, where larger (64-bit) general registers are used for high bandwidth in the memory and string functions.

The handling of floating point and vector data is the same (registers size and format and instructions) for 32-bit and 64-bit modes. Therefore, for these applications, the key decision depends on the address space requirements. For 32-bit Power applications (32-bit mode applications that are running on 64-bit Power hardware with a 64-bit kernel), the address space is limited to 4 GB, which is the limit of a 32-bit address. 64-bit applications are currently limited to 16 TB of application program or data per process. This limitation is not a hardware one, but is a restriction of the shared Linux virtual memory manager implementation. For applications with low latency response requirements, using the larger, 64-bit addressing to avoid I/O latencies using memory mapped files or large local caches is a good trade-off.

CPU-tuned libraries

If an application must support only one Power hardware platform (such as POWER7 and newer), then compiling the entire application with the appropriate `-mcpu=` and `-mtune=` compiler flags might be the best option.

For example, `-mcpu=power7` allows the compiler to use all the new instructions, such as the Vector Scalar Extended category. The `-mcpu=power7` option also implies `-mtune=power7` if it is not explicitly set.

`mcpu` focuses on the instruction mix that the compiler generates. `mtune` focuses on optimizing the order of the instructions

Most applications do need to run on more than one platform, for example, in POWER6 mode and POWER7 mode. For applications composed of a main program and a set of shared libraries or applications that spend significant execution time in other (from the Linux run time or extra package) shared libraries, you can create packages that automatically select the best optimization for each platform.

Linux also supports automatic CPU tuned library selection. There are a number of implementation options for CPU tuned library implementers as described here. For more information, see *Optimized Libraries*, available at:

<http://www.ibm.com/developerworks/wikis/display/LinuxP/Optimized%20Libraries>

The Linux Technology Center works with the SUSE and Red Hat Linux Distribution Partners to provide some automatic CPU-tuned libraries for the C/POSIX runtime libraries. However, these libraries might not be supported for all platforms or have the latest optimization.

One advantage of the Advance Toolchain is that the runtime RPMs for the current release do include CPU-tuned libraries for all the currently supported POWER processors and the latest processor-specific optimization and capabilities, which are constantly updated. Additional libraries are added as they are identified. The Advance Toolchain run time can be used with either Advance Toolchain GCC or XL compilers and includes configuration files to simplify linking XL compiled programs with the Advance Toolchain runtime libraries.

These techniques are not restricted to systems libraries, and can be easily applied to application shared library components. The dynamic code path and processor tuned libraries are good starting points. With this method, the compiler and dynamic linker do most of the work. You need only some additional build time and extra media for the multiple library images.

In this example, the following conditions apply:

- ▶ Your product is implemented in your own shared library, such as `libmyapp.so`.
- ▶ You want to support Linux running on POWER5, POWER6, and POWER7 Systems.
- ▶ DFP and Vector considerations:
 - Your oldest supported platform is POWER5, which does not have a DFP or the Vector unit.
 - POWER6 has DFP and a Vector Unit implementing the older VMX (vector float but no vector double) instructions.
 - POWER7 has DFP and the new Vector Scalar Extended (VSX) Unit (the original VMX instructions plus Vector Double and more).
 - Your application benefits greatly from both Hardware Decimal and high performance vector, but if you compile your application with `-mcpu=power7 -03`, it does not run on POWER5 (no hardware DFP instructions) or POWER6 (no vector double instructions) machines.

You can optimize all three Power platforms if you build and install your application and libraries correctly by completing the following steps:

1. Build the main application binary file and the default version of `libmyapp.so` for the oldest supported platform (in this case, use `-mcpu=power5 -03`). You can still use decimal data because the Advance Toolchain and the newest SLES11 and RHEL6 include a DFP emulation library and run time.
2. Install the application (`myapp`) into the appropriate `./bin` directory and `libmyapp.so` into the appropriate `./lib64` directory. The following paths provide the application main and default run time for your product:
 - `/opt/ibm/myapp1.0/bin/myapp`
 - `/opt/ibm/myapp1.0/lib64/libmyapp.so`
3. Compile and link `libmyapp.so` with `-mcpu=power6 -03`, which enables the compiler to generate DFP and VMX instructions for POWER6 machines.
4. Install this version of `libmyapp.so` into the appropriate `./lib64/power6` directory. For example:
`/opt/ibm/myapp1.0/lib64/power6/libmyapp.so`
5. Compile and link the fully optimized version of `libmyapp.so` for POWER7 with `-mcpu=power7 -03`, which enables the compiler to generate DFP and all the VSX instructions. Install this version of `libmyapp.so` into the appropriate `./lib64/power7` directory. For example:
`/opt/ibm/myapp1.0/lib64/power7/libmyapp.so`

By simply running some extra builds, your myapp1.0 is fully optimized for the current and N-1/N-2 Power hardware releases. When you start your application with the appropriate LD_LIBRARY_PATH (including /opt/ibm/myapp1.0/lib64), the dynamic linker automatically searches the subdirectories under the library path for names that match the current platform (POWER5, POWER6, or POWER7). If the dynamics linker finds the shared library in the subdirectory with the matching platform name, it loads that version; otherwise, the dynamic linker looks in the base lib64 directory and use the default implementation. This process continues for all directories in the library path and recursively for any dependent libraries.

Using the Advance Toolchain

The latest Advance Toolchain compilers and run time can be downloaded from:

<http://linuxpatch.ncsa.uiuc.edu/toolchain/at/>

The latest Advance Toolchain releases (starting with Advance Toolchain 5.0) add multi-core runtime libraries to enable you to take advantage of application level multi-cores. The toolchain currently includes a Power port of the open source version of Intel Thread Building Blocks, the Concurrent Building Blocks software transactional memory library, and the UserRCU library (the application level version of the Linux kernel's Read-Copy-Update concurrent programming technique). Additional libraries are added to the Advance Toolchain run time as needed and if resources allow it.

Linux on Power Enterprise Distributions default to 64 KB pages, so most applications automatically benefit from large pages. Larger (16 MB) segments can be best used with the libhugetlbf API. Large segments can be used to back shared memory, malloc storage, and (main) program text and data segments (incorporating large pages for shared library text or data is not supported currently).

Tuning and optimizing malloc

Methods for tuning and optimizing malloc are described in this section.

Linux malloc

Generally, tuning malloc invocations on Linux systems is an application-specific focus.

Improving malloc performance

Linux is flexible regarding the system and application tuning of malloc usage.

By default, Linux manages malloc memory to balance the ability to reuse the memory pool against the range of default sizes of memory allocation requests. Small chunks of memory are managed on the **sbrk** heap. This **sbrk** heap is labeled as [heap] in /proc/self/maps.

When you work with Linux memory allocation, there are a number of tunables available to users. These tunables are coded and used in the Linux malloc.c program. Our examples (“Malloc environment variables” on page 101 and “Linux malloc considerations” on page 102) show two of the key tunables, which force the large sized memory allocations away from using mmap, to using the memory on the program stack by using the **sbrk** system directive.

When you control memory for applications, the Linux operating system automatically makes a choice between using the stack for mallocs with the **sbrk** command, or mmap regions. Mmap regions are typically used for larger memory chunks. When you use mmap for large mallocs, the kernel must zero the newly mmaped chunk of memory.

Malloc environment variables

Users can define environment variables to control the tunables for a program. The environment variables that are shown in the following examples caused a significant performance improvement across several real-life workloads.

To disable the usage of mmap for mallocs (which includes Fortran allocates), set the max value to zero:

```
MALLOC_MMAP_MAX=0
```

To disable the trim threshold, set the value to negative one:

```
MALLOC_TRIM_THRESHOLD=-1
```

Trimming and using mmap are two different ways of releasing unused memory back to the system. When used together, they change the normal behavior of malloc across C and Fortran programs, which in some cases can change the performance characteristics of the program. You can run one of the following commands to use both actions:

- ▶ # ./my_program
- ▶ # MALLOC_MMAP_MAX=0 MALLOC_TRIM_THRESHOLD=-1 ./my_program

Depending on your application's behavior regarding memory and data locality, this change might do nothing, or might result in performance improvement.

Linux malloc considerations

The Linux GNU C run time includes a default malloc implementation that is optimized for multi-threading and medium sized allocations. For smaller allocations (less than the MMAP_THRESHOLD), the default malloc implementation allocates blocks of storage with **sbrk()** called arenas, which are then suballocated for smaller malloc requests. Larger allocations (greater than MMAP_THRESHOLD) are allocated by an anonymous mmap, one per request.

The default values are listed here:

DEFAULT_MXFAST	64 (for 32-bit) or 128 (for 64-bit)
DEFAULT_TRIM_THRESHOLD	128 * 1024
DEFAULT_TOP_PAD	0
DEFAULT_MMAP_THRESHOLD	128 * 1024
DEFAULT_MMAP_MAX	65536

Storage within arenas can be reused without kernel intervention. The default malloc implementation uses trylock techniques to detect contentions between POSIX threads, and then tries to assign each thread its own arena. This action works well when the same thread frees storage that it allocates, but it does result in more contention when malloc storage is passed between producer and consumer threads. The default malloc implementation also tries to use atomic operations and more granular and critical sections (lock and unlock) to enhance parallel thread execution, which is a trade-off for better multi-thread execution at the expense of a longer malloc path length with multiple atomic operations per call.

Large allocations (greater than MMAP_THRESHOLD) require a kernel syscall for each **malloc()** and **free()**. The Linux Virtual Memory Management (VMM) policy does not allocate any real memory pages to an anonymous **mmap()** until the application touches those pages. The benefit of this policy is that real memory is not allocated until it is needed. The downside is that, as the application begins to populate the new allocation with data, the application experiences multiple page faults, on first touch to allocate and zero fill the page. This situation means that on the initial touching of memory, there is more processing then, as opposed to the earlier timing when the original mmap is done. In addition, this first touch timing can impact the NUMA placement of each memory page.

Such storage is unmapped by **free()**, so each new large malloc allocation starts with a flurry of page faults. This situation is partially mitigated by the larger (64 KB) default page size of the Red Hat Enterprise Linux and SUSE Linux Enterprise Server on Power Systems; there are fewer page faults than with 4 KB pages.

Malloc tuning parameters

The default malloc implementation provides a `mallot()` API to allow applications to adjust some tuning parameters. For some applications, it might be useful to adjust the `MMAP_THRESHOLD`, `TOP_PAD`, and `MMAP_MAX` limits. Increasing `MMAP_THRESHOLD` so that most (application) allocations fall below that threshold reduces `syscall` and page fault impact, and improves application start time. However, this situation can increase fragmentation within the arenas and `sbrk()` storage. Fragmentation can be mitigated to some extent by also increasing `TOP_PAD`, which is the extra memory that is allocated for each `sbrk()`.

Reducing `MMAP_MAX`, which is the maximum number of chunks to allocate with `mmap()`, can also limit the use of `mmap()` when `MMAP_MAX` is set to 0. Reducing `MMAP_MAX` does not always solve the problem. The run time reverts to `mmap()` allocations if `sbrk()` storage, which is the gap between the end of program static data (bss) and the first shared library, is exhausted.

Linux malloc and memory tools

There are several readily available tools in the Linux open source community:

- ▶ A website that describes the heap profiler that is used at Google to explore how C++ programs manage memory, found at:

<http://gperftools.googlecode.com/svn/trunk/doc/heapprofile.html>

- ▶ *Massif: a heap profiler*, available at:

<http://valgrind.org/docs/manual/ms-manual.html>

For more details about memory management tools, see “Empirical performance analysis using the IBM SDK for PowerLinux” on page 172.

For more information about tuning malloc parameters, see *Malloc Tunable Parameters*, available at:

<http://www.gnu.org/software/libtool/manual/libc/Malloc-Tunable-Parameters.html>

Thread-caching malloc (TCMalloc)

Under some circumstances, an alternative malloc implementation can prove beneficial for improving application performance. Packaged as part of Google's Perftools package (<http://code.google.com/p/gperftools/?redir=1>), and in the Advance Toolchain 5.0.4 release, this specialized malloc implementation can improve performance across a number of C and C++ applications.

TCMalloc uses a thread-local cache for each thread and moves objects from the memory heap into the local cache as needed. Small objects with less than 32 KB are mapped into allocatable size-classes. A thread cache contains a singly linked list of free objects per size-class. Large objects are rounded up to a page size (4 KB) and handled by a central page heap, which is an array of linked lists.

For more information about how TCMalloc works, see *TCMalloc: Thread-Caching Malloc*, available at:

<http://gperftools.googlecode.com/svn/trunk/doc/tcmalloc.html>

The TCMalloc implementation is part of the gperftools project. For more information about this topic, go to:

<http://code.google.com/p/gperftools/>

Usage

To use TCMalloc, link TCMalloc into your application by using the `-ltcmalloc` linker flag by running the following command:

```
$ gcc [...] -ltcmalloc
```

You can also use TCMalloc in applications that you did not compile yourself by using `LD_PRELOAD` as follows:

```
$ LD_PRELOAD="/usr/lib/libtcmalloc.so"
```

These examples assume that the TCMalloc library is in `/usr/lib`. With the Advance Toolchain 5.0.4, the 32-bit and 64-bit libraries are in `/opt/at5.0/lib` and `/opt/at5.0/lib64`.

Using TCMalloc with hugepages

To use large pages with TCMalloc, complete the following steps:

1. Set the environment variables for `libhugetlbfs`.
2. Allocate the number of large pages from the system.
3. Set up the `libhugetlbfs` mount point.
4. Monitor large pages usage.

TCMalloc backs up the heap allocation on the large pages only.

Here are a more detailed version of these steps:

1. Set the environment variables for `libhugetlbfs` by running the following commands:

```
- # export TCMALLOC_MEMFS_MALLOCS_PATH=/libhugetlbfs/  
- # export HUGETLB_ELFMAP=RW  
- # export HUGETLB_MORECORE=yes
```

Where:

- `TCMALLOC_MEMFS_MALLOCS_PATH=/libhugetlbfs/` defines the `libhugetlbfs` mount point.
- `HUGETLB_ELFMAP=RW` allocates both RSS and BSS (text/code and data) segments on the large pages, which is useful for codes that have large static arrays, such as Fortran programs.
- `HUGETLB_MORECORE=yes` makes heap usage on the large pages.

2. Allocate the number of large pages from the system by running one of the following commands:

```
- # echo N > /proc/sys/vm/nr_hugepages  
- # echo N > /proc/sys/vm/nr_overcommit_hugepages
```

Where:

- `N` is the number of large pages to be reserved. A peak usage of 4 GB by your program requires 256 large pages (4096/16).
- `nr_hugepages` is the static pool. The kernel reserves $N * 16$ MB of memory from the static pool to be used exclusively by the large pages allocation.
- `nr_overcommit_hugepages` is the dynamic pool. The kernel sets a maximum usage of N large pages and dynamically allocates or deallocates these large pages.

3. Set up the `libhugetlbfs` mount point by running the following commands:

```
- # mkdir -p /libhugetlbfs  
- # mount -t hugetlbfs hugetlbfs /libhugetlbfs
```


4. Monitor large pages usage by running the following command:

```
# cat /proc/meminfo | grep Huge
```

This command produces the following output:

```
HugePages_Total:  
HugePages_Free:  
HugePages_Rsvd:  
HugePages_Surp:  
Hugepagesize:
```

Where:

- HugePages_Total is the total pages that are allocated on the system for LP usage.
- HugePages_Free is the total free memory available.
- HugePages_Rsvd is the total of large pages that are reserved but not used.
- Hugepagesize is the size of a single LP.

You can monitor large pages by NUMA nodes by running the following command:

```
# watch -d grep Huge /sys/devices/system/node/node*/meminfo
```

MicroQuill SmartHeap

MicroQuill SmartHeap is an optimized malloc that is used for SPECcpu2006 publishes for optimizing performance on selected benchmark components. For more information, see *SmartHeap for SMP: Does your app not scale because of heap contention?*, available at:

<http://www.microquill.com/smartheapsmp/index.html>

Large TOC -mmodel=medium optimization

The Linux Application Binary Interface (ABI) on the Power Architecture is enhanced to optimize larger programs. This ABI both simplifies an application build and improves overall performance.

Previously, the TOC (**-mfull-toc**) defaulted to a single instruction access form that restricts the total size of the TOC to 64 KB. This configuration can cause large programs to fail at compile or link time. Previously, the only effective workaround was to compile with the **-mmimal-toc** option (which provides a private TOC for each source file). The minimal TOC strategy adds a level of indirection that can adversely impact performance.

The **-mmodel=medium** option extends the range of the TOC addressing to +/-2 GB. This setup eliminates most TOC-related build issues. Also, as the Linux ABI TOC includes Global Offset Table (GOT) and local data, you can enable a number of compiler- and linker-based optimizations, including TOC pointer relative addressing for local static and constant data. This setup eliminates a level of indirection and improves the performance of large programs.

Currently, this optimization is only available with Advance Toolchain 4.0 and later.

The medium and large code models are 64-bit only. Advance Toolchain users must remove old **-mmimal-toc** options.

Selecting different SMT modes

Linux enables Power SMT capabilities. By default, the system runs at the highest SMT level. The **ppc64_cpu** command can be used to force the system kernel to use lower SMT levels (ST or SMT2 mode).

For example:

```
ppc64_cpu --smt=1    Sets the SMT mode to ST.  
ppc64_cpu --smt      Shows the current SMT mode.
```

When you run in these modes, the logical processor numbering does not change. However, in SMT2 mode, only every second logical processor is available (0, 2, 4, and so on), or in ST mode, only every fourth logical processor is available (0, 4, 8, and so on).

High SMT modes are best for maximizing total system throughput, while lower SMT modes might be appropriate for high performance threads and low latency applications. For code with low levels of instruction-level parallelism (often seen in Java code, for example), high SMT modes are generally preferred.

The setaffinity API allows processes and threads to have affinity to specific logical processors.

For more information about this topic, see 5.2, “Related publications” on page 106.

Using setaffinity to bind to specific logical processors

The setaffinity API allows processes and threads to have affinity to specific logical processors. The number and numbering of logical processors is a product of the number of processor cores (in the partition) and the SMT capability of the machine (four-way SMT for POWER7).

Linux completely fair scheduler

Java applications scale better on Linux in some cases if the `sched_compat_yield` scheduler tunable is set to 1 by running the following command:

```
sysctl -w kernel.sched_compat_yield=1
```

5.2 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *Red Hat Enterprise Linux 6 Performance Tuning Guide, Optimizing subsystem throughput in Red Hat Enterprise Linux 6, Edition 3.0*, found at:
http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html
- ▶ *SUSE Linux Enterprise Server System Analysis and Tuning Guide (Version 11 SP2)*, found at:
http://www.suse.com/documentation/sles11/pdfdoc/book_sle_tuning/book_sle_tuning.pdf



Compilers and optimization tools for C, C++, and Fortran

This chapter describes the optimization and tuning of the POWER7 processor-based server using compilers and tools. It covers the following topics:

- ▶ Compiler versions and optimization levels
- ▶ Advanced compiler optimization techniques
- ▶ IBM Feedback Directed Program Restructuring

6.1 Compiler versions and optimization levels

The IBM XL compilers are updated periodically to improve application performance and add processor-specific tuning and capabilities. The XLC11/XLF13 compilers for AIX and Linux are the first versions to include the capabilities of POWER7, and are the preferred version for projects that target current generation systems. The newer XLC12/XLF14 compilers provide performance improvements, and are preferred for template-heavy C++ codes.

The enterprise Linux distributions (RHEL6.1 GCC- 4.4 and SLES11/SP1 GCC- 4.3) include GCC compilers with POWER7 enabled (using the `-mcpu` and `-mtune` options), but do not have the latest Higher Order Optimizations. For the GNU GCC, G++ and gfortran compilers on Linux, the IBM Advance Toolchain 4.0 (GCC- 4.5) and 5.0 (GCC- 4.6) versions contain releases that are preferred for POWER7. XLF is preferred over gfortran for its high floating point performance characteristics.

For all production codes, it is imperative to enable a minimum level of compiler optimization by adding the `-O` option for the XL compilers, or `-O2` with the GNU compilers (`-O3` is the preferred option). Without optimization, the focus of the compiler is on faster compilation and debug ability, and it generates code that performs poorly at run time. In practice, many projects set up a dual build environment, with a development build without optimization for use during development and debugging, and a production build with optimization to be used for performance verification and production delivery.

For projects with increased focus on runtime performance, you should take advantage of the more advanced compiler optimization. For numerical or compute-intensive codes, the XL compiler options `-O3` or `-qhot -O3` enable loop transformations, which improve program performance by restructuring loops to make their execution more efficient by the target system. These options perform aggressive transformations that can sometimes cause minor differences on precision of floating point computations. If that is a concern, the original program semantics can be fully recovered with the `-qstrict` option.

For GCC, the minimum suggested level of optimization is `-O3`. The GCC default is a strict mode, but the `-ffast-math` option disables strict mode. The `-Ofast` option combines `-O3` with `-ffast-math` in a single option. Other important options include `-fpeel-loops`, `-funroll-loops`, `-ftree-vectorize`, `-fvect-cost-model`, and `-mmodel=medium`.

By default, these compilers generate code that run on various Power Systems. Options should be added to exclude older processor chips that are not supported by the target application. This configuration might enable better code generation as the compiler takes advantage of capabilities not available on those older systems.

There are two major XL compiler options to control this support:

- ▶ `-qarch`: Indicates the oldest processor chip generation that the binary file supports.
- ▶ `-qtune`: Indicates the processor chip generation of most interest for performance.

For example, for an application that must run on POWER6 systems, but for which most users are on a POWER7 system, the appropriate combination is `-qarch=pwr6 -qtune=pwr7`. For an application that must run well across both POWER6 and POWER7 Systems in current common usage, consider using `-qtune=balanced`.

On GCC, the equivalent options are `-mcpu` and `-mtune`. So, for an application that must run on POWER6, but which is usually run on POWER7, the options are `-mcpu=power6` and `-mtune=power7`.

The POWER7 processor supports the VSX instruction set, which improves performance for numerical applications over regular data sets. These performance features can increase the performance of some computations, and can be accessed manually by using the AltiVec vector extensions, or automatically by the XL compiler by using the `-qarch=pwr7 -qhot -O3 -qsimd` options.

The GCC compiler equivalents are the `-maltivec` and `-mvsx` options, which you should combine with `-ftree-vectorize` and `-fvect-cost-model`. On GCC, the combination of `-O3` and `-mcpu=power7` implicitly enables AltiVec and VSX code generation with auto-vector (`-ftree-vectorize`) and `-mpopcntd`. Other important options include `-mrecip=rsqrt` and `-mveclibabi=mass` (which *require* `-ffast-math` or `-Ofast` to be effective). If the compiler uses optimizations dependent on the MASS libraries, the link command must explicitly name the MASS library directories and library names.

For more information about this topic, see 6.4, “Related publications” on page 123.

6.2 Advanced compiler optimization techniques

This section describes some of the more advanced compiler optimization techniques.

6.2.1 Common prerequisites

Compiler analysis and transformations improve runtime performance by changing the translation of the program source into assembly code. Changes in these translations might cause the application to behave differently, possibly even causing it to produce incorrect results.

Compilers follow rules and assumptions that are part of the programming language to perform this transformation. If the programmer breaks some of these rules, it is possible for the application to misbehave, and it might do so only at higher optimization levels, where it is more difficult for the problem to be diagnosed.

To put this situation into perspective, imagine a C program with three variables: “int a[4], b, c;”. These variables are normally placed contiguously in memory. If the user runs a statement of the form `a[5]=0`, this statement breaks the language rules, but if variable b is unused, the statement might overwrite variable b and the program might continue to behave correctly. However, if, at a higher optimization level, variable b is eliminated, as the compiler determines it is unused, the incorrect statement might overwrite variable c, triggering a runtime failure.

It is critical, then, to eliminate programming errors as higher optimization is applied. Testing the application thoroughly without optimization is a good initial step, but it is not required or sufficient. The application must be tested at the optimization level to be used in production.

6.2.2 XL compiler family

There are several XL compiler programming errors that can provide guidance toward optimization.

Prerequisites

The XL compilers assist with identifying certain programming errors that are outlined 6.2.1, “Common prerequisites” on page 109:

- ▶ **Static analysis/warnings:** The XL compilers can identify suspicious code constructs, and provide some information about these constructs through the `-qinfo=all` option. You should examine the output of this option to identify suspicious code constructs and validate that the constructs are correct.
- ▶ **Runtime analysis or warning:** The XL compilers can cause the application to perform runtime checks to validate program correctness by using the `-qcheck` option. This option triggers a program abort when an error condition (such as a null pointer dereference or out-of-bounds array access) is run, identifying a problem and making it easier for you to identify it. This option has a significant performance cost, so use it only during functional verification, not on a production environment.
- ▶ **Aliasing compliance:** The C, C++, and Fortran languages specify rules that govern the access of data through overlapping pointers. These rules are brought into play aggressively by optimization techniques, but they can lead to incorrect results if they are broken. The compiler can be instructed not to take advantage of these rules, at a cost of runtime performance. This situation can be useful for older code that is written without following these rules. The options to request this optimization are `-qalias=noansi` for C/C++ and `-qalias=nostd` for Fortran.

High-order transformations

The XL compilers have sophisticated optimizations to improve the performance of numeric applications. These applications often contain regular loops that process large amounts of data. The high-order transformations (HOT) optimizations in these compilers analyze these loops, identify opportunities for restructuring them to improve cache usage, improve data reuse, and expose more instruction-level parallelism to the hardware. For these types of applications, the performance impact of this option can be substantial.

There are two levels of aggressiveness to the HOT optimization framework in these compilers:

- ▶ **Level 0**, which is the default at optimization level `-O3`, performs a minimal amount of loop optimization, focusing on simple opportunities while it minimizes compilation time.
- ▶ **Level 1**, which is the default at optimization levels `-O4` and up, performs full loop analysis and transformation of loops, and is preferred for numerical applications.

The HOT optimizations can be explicitly requested through the `-qhot=level=0` and `-qhot=level=1` options.

OpenMP

The OpenMP API is an industry specification for shared-memory parallel programming. The latest XL Compilers provide a full implementation of the OpenMP 3.0 specification in C, C++, and Fortran. You can program with OpenMP to capitalize on the incremental introduction of parallelism in an existing application by adding pragmas or directives to specify how the application can be parallelized.

For applications with available parallelism, OpenMP can provide a simple solution for parallel programming, without requiring low-level thread manipulation. The OpenMP implementation on the XL compilers is available by using the `-qsmp=omp` option.

Whole-program analysis

Traditional compiler optimizations operate independently on each application source file. Inter-procedural optimizations operate at the whole-program scope, using the interaction between parts of the application on different source files. It is often effective for large-scale applications that are composed of hundreds or thousands of source files.

On the XL compilers, these capabilities are accessed by using the `-qipa` option. It is also implied when you use optimization levels `-O4` and `-O5`. In this phase, the compiler saves a high-level representation of the program in the object files during compilation, and reoptimizes it at the whole-program scope during the link phase. For this situation to occur, the compiler driver must be used to link the resulting binary, instead of invoking the system linker directly.

Whole-program analysis (IPA) is effective on programs that use many global variables, overflowing the default AIX limit on global symbols. If the application requires the use of the `-bbigtoc` option to link successfully on AIX, it is likely a good candidate for IPA optimization.

There are three levels of IPA optimization on the XL compilers (0, 1, and 2). By default, `-qipa` implies `ipa=level=1`, which performs basic program restructuring. For more aggressive optimization, apply `-qipa=level=2`, which performs full program restructuring during the link step. The time that it takes to complete the link step can increase significantly.

Optimization that is based on Profile Directed Feedback

Profile-based optimization allows the compiler to collect information about the program behavior and use that information when you make code generation decisions. It involves compiling the program twice: first, to generate an *instrumented* version of the application that collects program behavior data when run, and a second time to generate an optimized binary file using information that is collected by running the instrumented binary through a set of typical inputs for the application.

Profile-based optimization in the XL compiler is accessed through the `-qpdf1` and `-qpdf2` options, on top of `-O` or higher optimization levels. The instrumented binary file is generated by using `-qpdf1` on top of all other options, and the resulting binary file generates the profile data on a file, named `._pdf` by default.

The Profile Directed Feedback (PDF) framework on the XL compilers is built on top of the IPA infrastructure, with `-qpdf1` and `-qpdf2` implying `-qipa=level=0`. For the PDF2 step, it is possible to reuse the object files from the `-qpdf1` compilation step, and relink only the application with the `-qpdf2` option.

For PDF optimizations to be successful, the instrumented workload must be run with common workloads that reflect common usage of the application. Use multiple workloads that can exercise the program in different ways. The data for all instrumentation runs are aggregated into a single PDF file and used during optimization.

For the PDF profile data to be written out at the end of execution, the program must either implicitly or explicitly call the `exit()` library subroutine. Using `exit()` causes code that is introduced as part of the PDF instrumentation to be run and write out the PDF profile data. In contrast, running the `_exit()` system call skips the writing of the PDF profile data file, which results in inaccurate profile data being recorded.

6.2.3 GCC compiler family

The information in this section applies specifically to the GCC compiler family.

Prerequisites

The GCC compiler assists with identifying certain programming errors that are outlined in 6.2.1, “Common prerequisites” on page 109:

- ▶ Static analysis and warnings. The **-pedantic** and **-pedantic-errors** options warn of violations of ISO C or ISO C++ standards.
- ▶ The language standard to enforce and the aliasing compliance requirements are specified by the **-std**, **-ansi**, and **-fno-strict-aliasing** options. For example:
 - ISO C 1990 level: **-std=c89**, **-std=iso9899:1990**, and **-ansi**
 - ISO C 1998 level: **-std=c99** and **-std=iso9899:1999**
 - Do not assume strict aliasing rules for the language level: **-fno-strict-aliasing**

The GCC compiler documentation contains more details about these options.^{1, 2, 3}

High-order transformations

The GCC compilers have sophisticated additional optimizations beyond **-O3** to improve the performance of numeric applications. These applications often contain regular loops that process large amounts of data. These optimizations, when enabled, analyze these loops, identify opportunities for restructuring them to improve cache usage, improve data reuse, and expose more instruction-level parallelism to the hardware. For these types of applications, the performance impact of this option can be substantial. The key compiler options include:

- ▶ **-fpeel-loops**
- ▶ **-funroll-loops**
- ▶ **-ftree-vectorize**
- ▶ **-fvect-cost-model**
- ▶ **-mmodel=medium**

Specifying the **-mveclibabi=mass** option and linking to the MASS libraries enables more loops for **-ftree-vectorize**. The MASS libraries support only static archives for linking, and so they require explicit naming and library search order for each platform/mode:

- ▶ POWER7 32-bit: **-L<MASS-dir>/lib -lmassvp -lmass_simdp7 -lmass -lm**
- ▶ POWER7 64-bit: **-L<MASS-dir>/lib64 -lmassvp_64 -lmass_simdp7_64 -lmass_64 -lm**
- ▶ POWER6 32-bit: **-L<MASS-dir>/lib -lmassvp6 -lmass -lm**
- ▶ POWER6 64-bit: **-L<MASS-dir>/lib64 -lmassvp6_64 -lmass_64 -lm**

ABI improvements

The **-mmodel={medium|large}** option implements important ABI improvements that are further optimized in hardware for future generations of the POWER processor. This optimization extends the TOC to 2 GB and eliminates the previous requirement for **-mminimal-toc** or multi-TOC switching within a single a program or library. The default for newer GCC compilers (including Advance Toolchain 4.0 and later) is **-mmodel=medium**. This model logically extends the TOC to include local static data and constants and allows direct data access relative to the TOC pointer.

¹ *Language Standards Supported by GCC*, available at:
<http://gcc.gnu.org/onlinedocs/gcc-3.4.2/gcc/Standards.html#Standards>

² *Options Controlling C Dialect*, available at:
<http://gcc.gnu.org/onlinedocs/gcc-3.4.2/gcc/C-Dialect-Options.html#C-Dialect-Options>

³ *Options That Control Optimization, and specifically the discussion of -fstrict-aliasing*, available at:
<http://gcc.gnu.org/onlinedocs/gcc-3.4.2/gcc/Optimize-Options.html#Optimize-Options>

OpenMP

The OpenMP API is an industry specification for shared-memory parallel programming. The current GCC compilers, starting with GCC- 4.4 (Advance Toolchain 4.0+), provide a full implementation of the OpenMP 3.0 specification in C, C++, and Fortran. Programming with OpenMP allows you to benefit from the incremental introduction of parallelism in an existing application by adding pragmas or directives to specify how the application can be parallelized.

For applications with available parallelism, OpenMP can provide a simple solution for parallel programming, without requiring low-level thread manipulation. The GNU OpenMP implementation on the GCC compilers is available under the **-fopenmp** option. GCC also provides auto-parallelization under the **-ftree-parallelize-loops** option.

Whole-program analysis

Traditional compiler optimizations operate independently on each application source file. Inter-procedural optimizations operate at the whole-program scope, using the interaction between parts of the application on different source files. It is often effective for large-scale applications that are composed of hundreds or thousands of source files.

Starting with GCC- 4.6 (Advance Toolchain 5.0), there is the Link Time Optimization (LTO) feature. LTO allows separate compilation of multiple source files but saves additional (abstract program description) information in the resulting object file. Then, at application link time, the linker can collect all the objects (with additional information) and pass them back to the compiler (GCC) for whole program IPA and final code generation.

The GCC LTO feature is enabled on the compile and link phases by the **-flto** option. A simple example follows:

```
gcc -flto -O3 -c a.c
gcc -flto -O3 -c b.c
gcc -flto -o program a.o b.o
```

Additional options that can be used with **-flto** include:

- ▶ **-flto-partition={1to1|balanced|none}**
- ▶ **-flto-compression-level=*n***

Detailed descriptions about **-flto** and its related options are in *Options That Control Optimization*, available at:

<http://gcc.gnu.org/onlinedocs/gcc-4.6.3/gcc/Optimize-Options.html#Optimize-Options>

Profiled-based optimization

Profile-based optimization allows the compiler to collect information about the program behavior and use that information when you make code generation decisions. It involves compiling the program twice: first, to generate an *instrumented* version of the application that collects program behavior data when run, and a second time to generate an optimized binary using information that is collected by running the instrumented binary through a set of typical inputs for the application.

Profile-based optimization in the GCC compiler is accessed through the **-fprofile-generate** and **-fprofile-use** options on top of **-O2** optimization levels. The instrumented binary is generated by using **-fprofile-generate** on top of all other options, and the resulting binary file generates the profile data in a file, named `._pdf` by default. For example:

```
gcc -fprofile-generate -O3 -c a.c
gcc -fprofile-generate -O3 -c b.c
```

```
gcc -fprofile-generate -o program a.o b.o
program < sample1
program < sample2
program < sample3
gcc -fprofile-use -O3 -c a.c
gcc -fprofile-use -O3 -c b.c
gcc -fprofile-use -o program a.o b.o
```

Additional options that are related to GCC PDF include:

- fprofile-correction** Corrects for missing counter samples from multi-threaded applications.
- fprofile-dir=PATH** Specifies the directory for generating and using profile data.
- fprofile-generate=PATH** Combines `-fprofile-generate` and `-fprofile-dir`.
- fprofile-use=PATH** Combines `-fprofile-use` and `-fprofile-dir`.

Detailed descriptions about **-fprofile-generate** and its related options can be found *Options That Control Optimization*, available at:

<http://gcc.gnu.org/onlinedocs/gcc-4.6.3/gcc/Optimize-Options.html#Optimize-Options>

For more information about this topic, see 6.4, “Related publications” on page 123.

6.3 IBM Feedback Directed Program Restructuring

Feedback Directed Program Restructuring (FDPR) is a feedback-based, directed, and post-link optimization tool.

6.3.1 Introduction

FDPR optimizes the executable binary file of a program by collecting information about the behavior of the program while the program is used for a typical workload, and then creates a new version of the program that is optimized for that workload. Both main executable and dynamically shared libraries (DLLs) are supported.

FDPR performs global optimizations at the level of the entire executable library, including statically linked library code. Because the executable library to be optimized by FDPR is not relinked, the compiler and linker conventions do not need to be preserved, thus allowing aggressive optimizations that are not available to optimizing compilers.

The main advantage that is provided by FDPR is the reduced footprint of both code and data, resulting in more effective cache usage. The principal optimizations of FDPR include global code reordering, global data reordering, function inlining, and loop unrolling, along with various tuning options tailored for the specific Power target. The effectiveness of the optimization depends largely on how representative the collected profile is regarding the true workload.

FDPR runs on both Linux and AIX and produces optimized code for all versions of the Power Architecture. POWER7 is its default target architecture.

Figure 6-1 shows how FDPR is used to optimize executable programs.

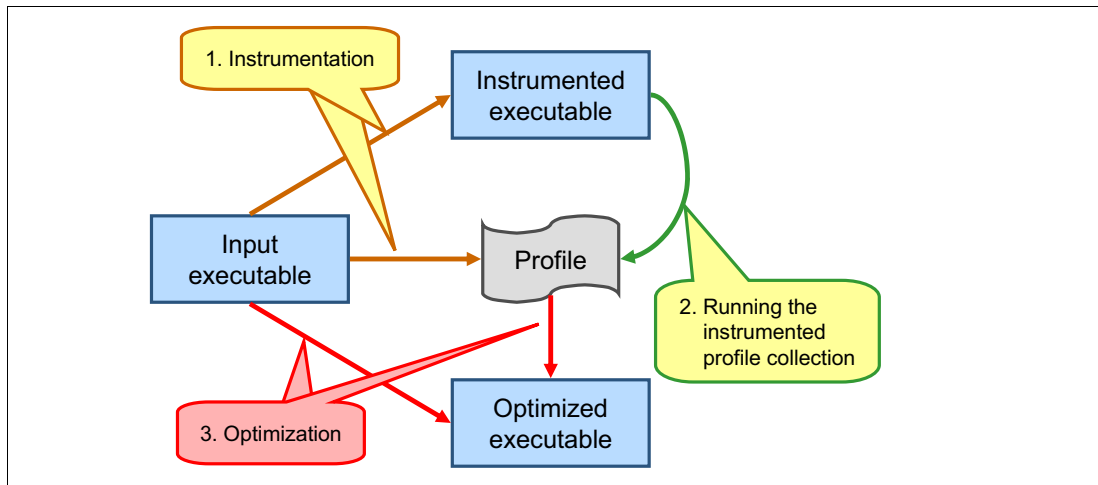


Figure 6-1 FDPR operation

FDPR builds an optimized executable program in three distinct phases:

1. Instrumentation (Yellow)
 - Creates an instrumented version of the input program and an empty profile file.
 - The input program can be an executable file or a dynamically linked shared library.
2. Profiling (Green)
 - Runs the instrumented program on a representative workload.
 - The profile file is filled with count data at run time.
3. Optimization (Red)

FDPR receives the original input program along with the filled profile file to create an optimized version of the input

6.3.2 FDPR supported environments

FDPR is available on the following platforms:

- ▶ AIX and Power Systems: Part of the AIX 5L V5 operating system and higher for both 32-bit and 64-bit applications. For more information, see *AIX 5L Performance Tools Handbook*, SG24-6039:
- ▶ Software Development Toolkit for PowerLinux: Available for use through the IBM SDK for PowerLinux. Linux distributions of Red Hat EL5 and above, and SUSE SLES10 and above are supported. For more information, see:

<http://www14.software.ibm.com/webapp/set2/sas/f/topdiags/sdklop.html>

In these resources, detailed online help, including manuals, is provided for each of these environments.

6.3.3 Acceptable input formats

The input binary can be a main executable program or a shared library, originally written in any language (for example, C, C++, or Fortran), if it is statically compiled. Thus, Java byte code is not acceptable. Code that is written in assembly language is acceptable, but must follow the Power ABI convention. For more information, see *64-bit PowerPC ELF Application Binary Interface Supplement 1.9*, available at:

<http://refspecs.linuxfoundation.org/ELF/ppc64/PPC-elf64abi-1.9.pdf>

It is important that the file includes relocation information. Although this is the default in AIX, on Linux you must add `-Wl,-q`, or `-Wl,--emit-relocs` to the command used for linking the program (or `-q` if the `ld` command is used directly).

The input binary can include debug information. FDPR correctly processes line number information so that the optimized output can be debugged.

6.3.4 General operation

FDPR is started by running the `fdprpro` program as follows:

```
$ fdprpro -a action [-p] in -o out -f prof [opt ...]
```

The action indicates the specific processing that is requested. The most common ones are `instr` for the instrumentation step and `opt` for the optimization step.

The `in`, `out`, and `prof` indicate the input and output binary files and profile files.

FDPR comes also with a wrapper command, named `fdpr`, which performs the instrumentation, profiling, and optimization under one roof. Run `man fdpr` for more information about this wrapper.

Special input and output files

FDPR has a number of options that control input and output files. One option that controls the input files is `--ignored-function-list file (-ifl file)`.

In some cases, the structure of some functions confuses FDPR, which can result in bad code generation. The file that is specified by `--ignored-function-list file (-ifl file)` contains a list of functions that are considered unsafe for optimization. This configuration prevents the potential bad code generation that might otherwise occur.

In addition to the profile and the instrumented and output optimized files, FDPR can optionally produce various secondary files to help you understand the static and dynamic nature of the input binary program. These secondary files have the same base name as the output file and a special extension. The options that control important output files are:

- ▶ `--disassemble_text (-d)` and `--dump-mapper (-dm)`: The `-d` option creates a disassembly of (the code segment) of the program (extension `.dis_text`). The disassembly is useful to understand the structure of program as analyzed or created by FDPR. The `-dm` option produces a mapping of basic-blocks from their original address to their address in the optimized code. This mapping can be used, for example, to understand how a specific piece of code was broken, or for user-specific post-processing tools.
- ▶ `--dump-ascii-profile (-dap)`: This option dumps the profile file in a human readable ASCII format (extension `.aprof`). The `.aprof` file is useful for manual inspection or user-defined post-processing of the collected profile.

- ▶ **--verbose *n* (-v *n*), --print-inlined-funcs (-pif), and --journal *file* (-j *file*):**
These options generate different analyses of the optimized file. **-v *n*** generates general and optimization-specific statistics (.stat extension). The amount of verbosity is set by *n*. Basic statistics are provided by **-v 1**. Optimization-specific statistics are added in level 2 and instruction mix in level 3. The list of inlining and inlined functions is produced with the **-pif** option (.inl_list extension). The **-j *file*** produces a journal of the main optimizations, in an XML formal, with detailed information about each optimization site, including the corresponding source file and line information. This information can be used by GUI tools to display optimizations in the context of the source code.

Controlling output to the console

The amount of progress information that is printed to the console can be controlled by two options. The default progress information is as follows

```
fdprpro (FDPR) Version vvv for Linux/POWER
fdprpro -a opt -O3 in -o out -f prof
> reading_exe ...
> adjusting_exe ...
> analyzing ...
> building_program_infrastructure ...
> building_profiling_cfg ...
> add_profiling ...
>> reading_profile ...
>> building_control_flow_transfer_profiling ...
> pre_reorder_optimizations ...
>> derat_optimization ...
...
```

This information might also be interspersed with warning and debugging messages. Use the **-quiet (-q)** option to avoid progress information. To limit the warning information, use the **-warning 1 (-w 1)** option.

6.3.5 Instrumentation and profiling

FDPR instrumentation is performed by running the following command:

```
$ fdprpro -a instr in [-o out] [-f prof] [opts...]
```

If **out** is not specified, the output file is in `in.instr`. If the profile is not specified, `in.nprof` is used.

Two files are created: the instrumented program and an empty profile. The instrumented program (or shared library), when run on a representative workload, fills the profile with execution counts of nodes and edges of the binary control flow graph (CFG). A node in this CFG is a basic block (piece of code with single entry and exit points). An edge indicates a control transfer between two basic blocks through a branch (regular branch, call, or return instruction).

To run the instrumented program, use the same command parameters as with the original program. As indicated in 6.3.1, “Introduction” on page 114, the workload that is exercised during the instrumented run should be representative, making the optimization step more effective. Because of the instrumentation code, the program is slower.

Successive runs of the instrumented program accumulate the counts in the same profile. Similarly, if the instrumented program is a shared library, each time the shared library participates in a process, the corresponding profile is updated with added counts.

Profiling shared libraries

When the dynamic linker searches for and links a shared library during execution, it looks for the original name that is used to the command used for linking the program. To ensure that the instrumented library is run, ensure that the following items are true:

1. The instrumented library should have the same name as the original library. The user can rename the original or place the libraries in different folders.
2. The folder that contains the library must be in the library search path: LIBPATH on AIX and LD_LIBRARY_PATH on Linux.

Moving and renaming the profile file

The location of the profile file is specified in the instrumented program, as indicated by the `-f` option. However, the profile file might be moved, or if its original specification is relative, the real location can change before execution.

Use the `-fdir` option to set the profile directory if it is known at instrumentation time and is different from the one implied or specified by the `-f` option.

Use the `FDPR_PROF_DIR` environment variable to specify the profile directory if the profile file is not present in the relative or absolute location where it was created in the instrumentation step (or where specified originally by `-fdir`).

Use the `FDPR_PROF_NAME` environment variable to specify the profile file name if the profile file name changed.

Profile file descriptor

When the instrumented binary file is run, the profile file is mapped to shared memory. The process is using a default file descriptor (FD) number (1023 on Linux and 1999 on AIX) for the mapping. If the application uses this specific FD, an error can occur during the profiling phase because of this conflict of use. Use the `-fd` option to change the default FD used by FDPR:

```
$ fdprpro -a instr my_prog -fd <fd num>
```

The FD can also be controlled by using the `FDPR_PROF_FD` environment variable by changing the FD at run time:

```
$ export FDPR_PROF_FD=fd_num
```

FDPR can be used to profile several binary executable files in a single run of an application. If so, you must specify a different FD for each binary. For example:

- ▶ `$ fdprpro -a instr in/libmy_lib1 -o out/libmy_lib1 -f out/libmy_lib1.prof -fd 1023`
- ▶ `$ fdprpro -a instr in/libmy_lib2 -o out/libmy_lib2 -f out/libmy_lib2.prof -fd 1022`

Because environment variables are global in nature, when profiling several binary files at the same time, use explicit instrumentation options (`-f`, `-fd`, and `-fdir`) to differentiate between the profiles rather than using the environment variables (`FDPR_PROF_FD` and `FDPR_PROF_NAME`).

Instrumentation stack

The instrumentation is using the stack for saving registers by dynamically allocating space on the stack at a default location below the current stack pointer. On AIX, this default is at offset -10240, and on Linux it is -1800. In some cases, especially in multi-threaded applications where the stack space is divided between the threads, following a deep calling sequence, the application can be quite close to the end of the stack, which can cause the application to fail. To allocate the instrumentation closer to the current stack pointer, use the `-iso` option:

```
$ fdprpro -a instr my_prog -iso -300
```

6.3.6 Optimization

The optimization step is performed by running the following command:

```
$ fdprpro -a opt in [-o out] -f prof [opts...]
```

If `out` is not specified, the output file is `in.fdpr`. No profile is provided by default. If `none` is specified or if the profile is empty, the resulting output binary file is not optimized.

Code reordering

Global code reordering works in two phases: making chains and reordering the chains.

The initial chains are sequentially ordered basic blocks, with branch conditions inverted where necessary, so that branches between the basic blocks are mostly not taken. This configuration makes instruction prefetching more efficient. Chains are terminated when the heat (that is, execution count) goes below a certain threshold relative to the initial heat.

The second phase orders chains by successively merging the more strongly linked two chains, based on how frequent the calls between the chains are. Combining chains crosses function boundaries. Thus, a function can be broken into multiple chunks in which different pieces of different functions are placed closely if there is a high frequency of call, branch, and return between them. This approach improves code locality and thus i-cache and page table efficiency.

You use the following options for code reordering:

- ▶ **--reorder-code (-RC)**: This component is the hard-working component of the global code reordering. Use **--rcaf** to determine the aggressiveness level:
 - 0: no change
 - 1: standard (default)
 - 2: most aggressive.

Use **--rcctf** to lower the threshold for terminating chains. Use **-pp** to preserve function integrity and **-pc** to preserve CSECT integrity (AIX only). These two options limit global code reordering and might be requested for ease of debugging.

- ▶ **--branch-folding (-bf)** and **--branch-prediction (-bp)**: These options control important parts of the code reordering process. The **-bf** folds branch to branch into a single branch. The **-bp** sets the static branch prediction bit when taken or not taken statistics justify it.

Function inlining

FDPR performs function inlining of function bodies into their respective calling sites if the call site is selected by one of a number of user-selected filters:

- ▶ Dominant callers (**--selective-inlining** (**-si**), **-sidf** *f*, and **-siht** *f*): The filter criteria here is that the site is dominant regarding other callers of the called function (the callee). It is controlled by two attributes. The **-sidf** option sets the domination percentage threshold (default 80). The **-siht** option further restricts the selection to functions hotter than the threshold, which is specified in percents relative to the average (default 100).
- ▶ Hot functions (**--inline-hot-functions** *f* (**-ihf** *f*)): This filter selects inlining for all call sites where the call is hotter than the heat threshold (in percent, relative to the average).
- ▶ Small functions (**--inline-small-functions** *f* (**-isf** *f*)): This filter selects for inlining all functions whose size, in bytes, is smaller than or equal to the parameter.
- ▶ Selective hot code (**--selective-hot-code-inline** *f* (**-shci** *f*)): The filter computes how much execution count is saved if the function is inlined at a call site and selects those sites where the relative saving is above the percentage.

De-virtualization

De-virtualization is addressed by the **--ptrgl-optimization** (**-pto**) option. It is full-blown call by a pointer mechanism (**ptrgl**) sets a new TOC anchor, loads the function address, moves it to the counter register (CTR), and jumps indirectly through the CTR. The **-pto** optimizes this mechanism in cases where there is few hot targets from a calling site. In terms of C++, it de-virtualizes the virtual method calls by calling the actual targets directly. The optimized code compares the address of the function descriptor, which is used for the indirect call, against the address of a hot candidate, as identified in the profile, and conditionally calls such target directly. If none of the hot targets match, the code invokes the original indirect call mechanism. The idea is that most of the time the conditional direct branches are run instead of the **ptrgl** mechanism. The impact of the optimization on performance depends heavily on the function call profile.

The following thresholds can help to tune the optimization and to adjust it to different workloads:

- ▶ Use **-ptoht** *thres* to set the frequency threshold for indirect calls that are to be optimized (*thres* can be 0 - 1, with 0.8 by default).
- ▶ Use **-ptos1** *n* to set the limit of the number of hot functions to optimize in a given indirect call site (the default for *n* is 3).

Loop-unrolling

Most programs spend their time in loops. This statement is true regardless of the target architecture or application. FDPR has one option to control the unrolling optimization for loops: **--loop-unrolling** *factor* (**-lu** *factor*).

FDPR optimizes loop using a technique called *loop-unrolling*. By unrolling a loop *n* times, the number of back branches is reduced *n* times, so code prefetch efficiency can be improved. The downside with loop-unrolling is code inflation, which results in increased code footprint and increased i-cache misses. Unlike traditional loop-unrolling, FDPR is able to mitigate this problem by unrolling only the hottest paths in the loop. The *factor* parameter determines the aggressiveness of the optimization. With **-03**, the optimization is invoked with **-lu 9**.

By default, loops are unrolled two times. Use **-lu** *factor* to change that default.

Architecture-specific optimizations

Here are some architecture-specific optimizations:

- ▶ **--machine *tgt* (-m *tgt*):** FDPR optimizations include general optimizations that are based on a high-level program representation as a control and data flow, in addition to peephole optimizations, relying on different architecture features. Those optimizations can perform better when tuned for specific platforms. The **-m** flag allows the user to specify the target machine model when known in cases where the program is not intended for use on multiple target platforms. The default target is POWER7.
- ▶ **--align-code *code* (-A *code*):** Optimizing the alignment and the placement of the code is crucial to the performance of the program. Correct alignment can improve instruction fetching and dispatching. The alignment algorithm in FDPR uses different techniques that are based on the target platform. Some techniques are generic for the Power Architecture, and others are considered dispatch rules of the specific machine model. If ***code*** is 1 (the default), FDPR applies a standard alignment algorithm that is adapted for the selected target machine (see **-m** in the previous bullet point). If ***code*** is 2, FDPR applies a more advanced version, using dispatch rules and other heuristics to decide how the program code chunks are placed relatively to i-cache sectors, again based on the selected target. A value of 0 disables the alignment algorithm.

Function optimization

FDPR includes a number of function level optimizations that are based on detailed data flow analysis (DFA). With DFA, optimizations can determine the data that is contained in each register at each point in the function and whether this value is used later.

The function optimizations are:

- ▶ **--killed-regs (-kr):** A register is considered killed at a point (in the function) if its value is not used in any ensuing path. FDPR uses the Power ABI convention that defines which registers are non-volatile (NV) across function calls. NV registers that are used inside a function are saved in its prolog and restored in its epilog. The **-kr** optimization analyzes called functions that are looking for save and restore instructions of killed NV registers. If the register is killed at the calling site, then the save and restore instructions for this register are removed. The optimization considers all calls to this function, because an NV might be alive when the function is called. When needed, the optimization might also reassign (rename) registers at the calling side to ensure that an NV is indeed killed and can be optimized.
- ▶ **--hco-reschedule (-hr):** The optimization analyzes the flow through hot basic blocks and looks for instructions that can be moved to dominating colder basic blocks (basic block b1 dominates b2 if all paths to b2 first go through b1). For example, an instruction that loads a constant to a register is a candidate for such motion.
- ▶ **--simplify-early-exit *factor* (-see *factor*):** Sometimes a function starts with an early exit condition so that if the condition is met, the whole body of the function is ignored. If the condition is commonly taken, it makes sense to avoid saving the registers in the prolog and restoring them in the epilog. The **-see** optimization detects such a condition and provides a reduced epilog that restores only registers modified by computing the condition. If ***factor*** is 1, a more aggressive optimization is performed where the prolog is also optimized.

Peephole optimization

Peephole optimizations require a small context around the specific site in the code that is problematic. The more important ones that FDPR performs are **-1as**, **-t1o**, and **-nop**.

- ▶ **--load-after-store (-1as)**: In recent Power Architectures, when a load instruction from address A closely follows a store to that address, it can cause the load to be rejected. The instruction is then tried in a slower mode, which produces a large performance penalty. This behavior is also called *Load-Hit-Store (LHS)*. With the **-1as** optimization, the load is pushed further from the store, thus avoiding the reject condition.
- ▶ **--toc-load-optimization (-t1o)**: The TOC (Table-Of-Content) is a data section in programs where pointers are kept to avoid the lengthy address computation at run time. Loading an address (a pointer) is a costly operation and FDPR is able to save on processing if the address is close enough to the TOC anchor (R2). In such cases, the load from TOC is replaced by an `addi Rt,R2,offset`, where $R2+offset = \text{loaded address}$. The optimization is performed after data is reordered so that commonly accessed data is placed closer to R2, increasing the potential of this optimization. A TOC is used in 32-bit and 64-bit programs on AIX, and in 64-bit programs on Power Systems running Linux. Linux 32-bit uses a GOT and this optimization is not relevant there.
- ▶ **--nop-removal (-nop)**: The compiler (or the linker) sometimes inserts no-operation (NOP) instructions in various places to create some necessary space in the instruction stream. The most common place is following a function call in code. Because the call might have modified the TOC anchor register (R2), the compiler inserts a load instruction that resets R2 to its correct value for the current function. Because FDPR has a global view of the program, the optimization can remove the NOP if the called function uses the same TOC (the TOC anchor is used in AIX and in Linux 64-bit).

Data reordering

The profile that is collected by FDPR provides important information about the running of branch instructions, thus enabling efficient code reordering. The profile does not provide this direct information whether to put specific objects one after the other. Nevertheless, FDPR is able to infer such placement by using the collected profile.

The relevant options are:

- ▶ **--reorder-data (-RD)**: This optimization reorders data by placing pointers and data closer to the TOC anchor, depending on their hotness. FDPR uses a heuristic where the hotness is computed as the total count of basic blocks where the pointer to the data was retrieved from the TOC.
- ▶ **--reduce-toc *thres* (-rt *thres*)**: The optimization removes from the TOC entries that are colder than the threshold. Their access, if any, is replaced by computing the address (see **-t1o** optimization in “Peephole optimization” on page 122). Typically, you use **-rt 0**, which removes only the entries that are never accessed.

Combination optimizations

FDPR has predefined optimization sets that provide a good starting point for performance tuning:

- ▶ **-0**: Performs code reordering (**-RC**) with branch prediction bit setting (**-bp**), branch folding (**-bf**), and NOOP instructions removal (**-nop**).
- ▶ **-02**: Adds to **-0** function de-virtualization (**-pto**), TOC-load optimization (**-t1o**), function inlining (**-isf 8**), and some function optimizations (**-hr**, **-see 0**, and **-kr**).

- ▶ **-O3**: Switches on data reordering (**-RD** and **-rt 0**), loop-unrolling (**-lu**), more aggressive function optimization (**-see 1** and **-vro**), and employs more aggressive inlining (**-lro** and **-isf 12**). This set provides an aggressive but still stable set of optimizations that are beneficial for many benchmarks and applications.
- ▶ **-O4**: Essentially turns on more aggressive inlining (**-sidf 50**, **-ihf 20**, and **-shci 90**). As a result, the number of branches is reduced, but at the cost of increasing code footprint. This option works well with large i-caches or with small to medium programs/threads.

For more information about this topic, see 6.4, “Related publications” on page 123.

6.4 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *C/C++ Cafe* (IBM Rational), found at:
 - <http://www.ibm.com/rational/cafe/community/ccpp>
- ▶ *FDPR, Post-Link Optimization for Linux on Power*, found at:
 - <https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=5a116d75-b560-4152-9113-7515fa73e67a>
- ▶ *Feedback Directed Program Restructuring (FDPR)*, found at:
 - <https://www.research.ibm.com/haifa/projects/systems/cot/fdpr/>
- ▶ GCC online documentation
 - All versions: <http://gcc.gnu.org/onlinedocs/>
 - Advance Toolchain 4.0: <http://gcc.gnu.org/onlinedocs/gcc-4.5.3/gcc/>
 - Advance Toolchain 5.0: <http://gcc.gnu.org/onlinedocs/gcc-4.6.3/gcc/>
 - Advance Toolchain 6.0 (3Q2012): <http://gcc.gnu.org/onlinedocs/gcc-4.7.1/gcc/>
- ▶ XL Compiler Documentation:
 - C and C++ Compilers
 - *C and C++ Compilers*, found at:
 - <http://www.ibm.com/software/awdtools/xlcpp/>
 - *Optimization and Programming Guide - XL C/C++ for AIX, V12.1*, found at:
 - <http://www.ibm.com/support/docview.wss?uid=swg27024208>
 - Fortran compilers
 - *Fortran Compilers family*, found at:
 - <http://www.ibm.com/software/awdtools/fortran/>
 - *Optimization and Programming Guide - XL Fortran for AIX, V14.1*, found at:
 - <http://www.ibm.com/support/docview.wss?uid=swg27024219>



Java

This chapter describes the optimization and tuning of Java based applications that are running on a POWER7 processor-based server. It covers the following topics:

- ▶ Java levels
- ▶ 32-bit versus 64-bit Java
- ▶ Memory and page size considerations
- ▶ Java garbage collection tuning
- ▶ Application scaling

7.1 Java levels

You should use Java 6 SR7 or later for POWER7 Systems for two primary reasons. First, 64 KB pages are used for JVM text, data, and stack memory segments and the Java heap by default on systems where 64 KB pages are available. Second, the JIT compiler in Java 6 SR7 and later takes advantage of POWER7 specific hardware features for performance. As a result, Java 6 SR7 or later perform better on POWER7 than older releases of the JVM.

Although 64 KB pages are available since AIX 5L V5.3 and POWER5+ systems, the default page size on AIX is still 4 KB. JVM releases before Java 6 SR7 use the default memory pages that are provided by AIX unless an AIX environment variable is set to override it. (For more information, see “Tuning to capitalize on hardware performance features” on page 12.) The Java heap is mapped to use the default page size unless the appropriate option is used to override it. (For more information, see 7.3.1, “Medium and large pages for Java heap and code cache” on page 127.) In reality, performance-savvy users override the default page size anyway with 64 KB pages to obtain most of the performance benefits from using larger memory pages on Power Systems. Java6 SR7 and later use 64 KB pages almost everywhere 64 KB pages are available and applicable. With Java 6 SR7 or later, no overriding option is necessary, and no AIX environment variable must be set.

The JIT compiler automatically detects on what platform it is running and generates binary code most suitable to, and performing best on, on that platform. Java 6 SR7 and later is able to recognize POWER7 and best use its hardware features. For example, POWER7 supports prefetch instructions for transient data, which is needed, but this data must be evacuated from the CPU caches with priority, which results in more efficient usage of CPU caches and leads to better performance if the workload had identifiable transient data. Many Java objects are inherently transient. Java 6 SR7 and later takes advantage of these prefetch instructions if the appropriate option is specified, as indicated in 7.3.3, “Prefetching” on page 128.

There is a newer Java release that is only available for Power when bundled with WebSphere Application Server. WebSphere Application Server V8 contains Java 6 with a later, improved V2.6 J9 virtual machine (VM). This VM is preferred over Java 6 with the older V2.4 J9 VM because of the performance improvements it contains.

The newest release of Java, Java 7, was generally available at the end of September 2011. It is available as a stand-alone release (not only when bundled with WebSphere Application Server). As with the updated Java 6 in WebSphere Application Server V8, Java7 contains significant improvements in the JDK as compared to the original Java 6. Java 7 is the preferred version of Java to use on POWER7.

7.2 32-bit versus 64-bit Java

64-bit applications that do not require large amounts of memory typically run slower than 32-bit applications. This situation occurs because of the larger data types, like 64-bit pointers instead of 32-bit pointers, which increase the demand on memory throughput.

The exception to this situation is when the processor architecture has more processor registers in 64-bit mode than in 32-bit mode and 32-bit application performance is negatively impacted by this configuration. Because of few registers, the demand on memory throughput can be higher in 32-bit mode than in 64-bit mode. In such a situation, running an application in 64-bit mode is required to achieve best performance.

The Power Architecture does not require running applications in 64-bit mode to achieve best performance because 32-bit and 64-bit modes have the same number of processor registers.

Consider the following items:

- ▶ Applications with a small memory requirement typically run faster as 32-bit applications than as 64-bit applications.
- ▶ 64-bit applications have a larger demand on memory because of the larger data types, such as pointers being 64-bit instead of 32-bit, which leads to the following circumstances:
 - The memory foot print increases because of the larger data types.
 - The memory alignment of application data contributes to memory demand.
 - More memory bandwidth is required.

For best performance, use 32-bit Java unless the memory requirement of the application requires running in 64-bit mode.

For more information about this topic, see 7.6, “Related publications” on page 136.

7.3 Memory and page size considerations

IBM Java can take advantage of medium (64 KB) and large (16 MB) page sizes that are supported by the current AIX versions and POWER processors. Using medium or large pages instead of the default 4 KB page size can improve application performance. The performance improvement of using medium or large pages is a result of a more efficient use of the hardware translation caches, which are used when you translate application page addresses to physical page addresses. Applications that are frequently accessing a vast amount of memory benefit most from using pages sizes that are larger than 4 KB.

Table 7-1 shows the hardware and software requirements for 4 KB, 64 KB, and 16 MB pages:

Table 7-1 Page sizes that are supported by AIX and POWER processors

Page size	Platform	AIX version	Requires user configuration
4 KB	All	All	No
64 KB	POWER5+ or later	AIX 5L V5.3 and later	No
16 MB	POWER4 or later	AIX 5L V5.3 and later	Yes

7.3.1 Medium and large pages for Java heap and code cache

Medium and large pages can be enabled for the Java heap and JIT code cache independently of other memory areas. The JVM supports at least three page sizes, depending on the platform:

- ▶ 4 KB (default)
- ▶ 64 KB
- ▶ 16 MB

The **-X1p64k** and **-X1p16m** options can be used to select the wanted page granularity for the heap and code cache. The **-X1p** option is an alias for **-X1p16m**. Large pages, specifically 16 MB pages, do have some processing impact and are best suited for long running applications with large memory requirements. The **-X1p64k** option provides many of the benefits of 16 MB pages with less impact and can be suitable for workloads that benefit from large pages but do not take full advantage of 16 MB pages.

Starting with IBM Java 6 SR7, the default page size is 64 KB.

7.3.2 Configuring large pages for Java heap and code cache

Large pages must be configured on AIX by the system administrator by running **vmo**. The following example demonstrates how to dynamically configure 1 GB of 16 MB pages:

```
# vmo -o lpgg_regions=64 -o lpgg_size=16777216
```

To permanently configure large pages, the **-r** option must be specified with the **vmo** command. Run **bosboot** to configure the large pages at boot time:

```
▶ # vmo -r -o lpgg_regions=64 -o lpgg_size=16777216
▶ # bosboot -a
```

Non-root users must have the **CAP_BYPASS_RAC_VMM** capability on AIX enabled to use large pages. The system administrator can add this capability by running **chuser**:

```
# chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user_id>
```

On Linux, 1 GB of 16 MB pages are configured by running **echo**:

```
# echo 64 > /proc/sys/vm/nr_hugepages
```

7.3.3 Prefetching

Prefetching is an important strategy to reduce memory latency and take full advantage of on-chip caches. The **-Xt1hPrefetch** option can be specified to enable aggressive prefetching of thread-local heap memory shortly before objects are allocated. This option ensures that the memory required for new objects that are allocated from the TLH is fetched into cache ahead of time if possible, reducing latency and increasing overall object allocation speed. This option can give noticeable gains on workloads that frequently allocate objects, such as transactional workloads.

7.3.4 Compressed references

For huge workloads, 64-bit JVMs might be necessary to meet application needs. The 64-bit processes primarily offer a much larger address space, allowing for larger Java heaps, JIT code caches, and reducing the effects of memory fragmentation in the native heap. However, 64-bit processes also must deal with the increased processing impact. The impact comes from the increased memory usage and decreased cache usage. This impact is present with every object allocation, as each object must now be referred to with a 64-bit address rather than a 32-bit address.

To alleviate this impact, use the **-Xcompressedrefs** option. When this option is enabled, the JVM uses 32-bit references to objects instead of 64-bit references wherever possible. Object references are compressed and extracted as necessary at minimal cost. The need for compression and decompression is determined by the overall heap size and the platform the JVM is running on; smaller heaps can do without compression and decompression, eliminating even this impact. To determine the compression and decompression impact for a heap size on a particular platform, run the following command:

```
java -Xcompressedrefs -verbose:gc -version ...
```

The resulting output has the following content:

```
<attribute name="compressedRefsDisplacement" value="0x0" />  
<attribute name="compressedRefsShift" value="0x0" />
```

Values of 0 for the named attributes essentially indicate that no work must be done to convert between 32-bit and 64-bit references for the invocation. Under these circumstances, 64-bit JVMs running with **-Xcompressedrefs** can reduce the impact of 64-bit addressing even more and achieve better performance.

With **-Xcompressedrefs**, the maximum size of the heap is much smaller than the theoretical maximum size allowed by a 64-bit JVM, although greater than the maximum heap under a 32-bit JVM. Currently, the maximum heap size with **-Xcompressedrefs** is around 31 GB on both AIX and Linux.

7.3.5 JIT code cache

JIT compilation is an important factor in optimizing performance. Because compilation is carried out at run time, it is complicated to estimate the size of the program or the number of compilations that are carried out. The JIT compiler has a cap on how much memory it can allocate at run time to store compiled code and for most of applications the default cap is more than sufficient.

However, certain programs, especially those programs that take advantage of certain language features, such as reflection, can produce a number of compilations and use up the allowed amount of code cache. After the limit of code cache is consumed, no more compilations are performed. This situation can have a negative impact on performance if the program begins to call many interpreted methods that cannot be compiled as a result. The **-Xjit:codetotal=<nnn>** (where *nnn* is a number in KB units) option can be used to specify the cap of the JIT code cache. The default is 64 MB or 128 MB for 32-bit and 64-bit JVMs.

Another consideration is how the code caches are allocated. If they are allocated far apart from each other (more than 32 MB away), calls from one code cache to another carry higher processing impact. The **-Xcodecache<size>** option can be used to specify how large each allocation of code cache is. For example, **-Xcodecache4m** means 4 MB is allocated as code cache each time the JIT compiler needs a new one, until the cap is reached. Typically, there are multiple pieces (for example, 4) of code cache available at boot-up time to support multiple compilation threads. It is important to alter the default code cache size only if it is insufficient, as a large but empty code cache needlessly consumes resources.

Two techniques can be used to determine if the code cache allocation sizes or total limit must be altered. First, a Java core file can be produced by running **kill -3 <pid>** at the end/stable state of your application. The core file shows how many pieces of code cache are allocated. The active amount of code cache can be estimated by summing up all of the pieces.

For example, if 20 MB is needed to run the application, `-Xcodecache5m` (four pieces of 5 MB each) typically allocates 20 MB code caches at boot-up time, and they are likely close to each other and have better performance for cross-code cache calls. Second, to determine if the total code cache is sufficient, the `-Xjit:verbose` option can be used to print method names as they are compiled. If compilation fails because the limit of code cache is reached, an error to that effect is printed.

7.3.6 Shared classes

The IBM JVM supports class data sharing between multiple JVMs. The `-Xshareclasses` option can be used to enable it, and the `-Xscmx<size>` option can be used to specify the maximum cache size of the stored data, where `<size>` can be `<nnn>K`, `<nnn>M`, or `<nnn>G` for sizes in KB, MB, or GB.

The shared class data is stored in a memory-mapped cache file on disk. Sharing reduces the overall virtual storage consumption when more than one JVM shares a cache. Sharing also reduces the start time for a JVM after the cache is created. The shared class cache is independent of any running JVM and persists until it is deleted.

A shared cache can contain:

- ▶ Bootstrap classes
- ▶ Application classes
- ▶ Metadata that describes the classes
- ▶ Ahead-of-time (AOT) compiled code

7.4 Java garbage collection tuning

The IBM Java VM supports multiple garbage collection (GC) strategies to allow software developers an opportunity to prioritize various factors. Throughput, latency, and scaling are the main factors that are addressed by the different collection strategies. Understanding how an application behaves regarding allocation frequencies, required heap size, expected lifetime of objects, and other factors can make one or more of the non-default GC strategies preferable. The GC strategy can be specified with the `-Xgcpolicy:<policy>` option.

7.4.1 GC strategy: Optthruput

This strategy prioritizes throughput at the expense of maximum latency by waiting until the last possible time to do a GC. A global GC of the entire heap is performed, creating a longer pause time at the expense of latency. After GC is triggered, the GC stops all application threads and performs the three GC phases:

- ▶ Mark
- ▶ Sweep
- ▶ Compact (if necessary)

All phases are parallelized to perform GC as quickly as possible.

The optthruput strategy is the default in the original Java 6 that uses the V2.4 J9 VM.

7.4.2 GC strategy: Optavgpause

This strategy prioritizes latency and response time by performing the initial mark phase of GC concurrently with the execution of the application. The application is halted only for the sweep and compact phases, minimizing the total time that the application is paused. Performing the mark phase concurrently with the execution of the application might affect throughput, because the CPU time that would otherwise go to the application can be diverted to low priority GC threads to carry out the mark phase. This situation can be acceptable on machines with many processor cores and relatively few application threads, as idle processor cores can be put to good use otherwise.

7.4.3 GC strategy: Gencon

This strategy employs a generational GC scheme that attempts to deal with many varying workloads and memory usage patterns. In addition, gencon also uses concurrent marking to minimize pause times. The gencon strategy works by dividing the heap into two categories:

- ▶ New space
- ▶ Old space

The new space is dedicated to short-lived objects that are created frequently and unreferenced shortly thereafter. The old space is for long-lived objects that survived long enough to be promoted from the new space. This GC policy is suited to workloads that have many short-lived objects, such as transactional workloads, because GC in the new space (carried out by the *scavenger*) is cheaper per object overall than GC in the old space. By default, up to 25% of the heap is dedicated to the new space. The division between the new space and the old space can be controlled with the `-Xmn` option, which specifies the size of the new space; the remaining space is then designated as the old space. Alternatively, `-Xmns` and `-Xmnx` can be used to set the starting and maximum new space sizes if a non-constant new space size is wanted. For more information about constant versus non-constant heaps in general, see 7.4.5, “Optimal heap size” on page 132.

The gencon strategy is the default in the updated Java 6 that uses the V2.6 J9 VM, and in the later Java 7 version.

7.4.4 GC strategy: Balanced

This strategy evens out pause times across GC operations that are based on the amount of work that is being generated. This strategy can be affected by object allocation rates, object survival rates, and fragmentation levels within the heap. This smoothing of pause times is a best effort rather than a real-time guarantee. A fundamental aspect of the balanced collector's architecture, which is critical to achieving its goals of reducing the impact of large collection times, is that it is a region-based garbage collector. A region is a clearly delineated portion of the Java object heap that categorizes how the associated memory is used and groups related objects together.

During the JVM startup, the garbage collector divides the heap memory into equal-sized regions, and these region delineations remain static for the lifetime of the JVM. Regions are the basic unit of GC and allocation operations. For example, when the heap is expanded or contracted, the memory that is committed or released corresponds to a number of regions.

Although the Java heap is a contiguous range of memory addresses, any region within that range can be committed or released as required. This situation enables the balanced collector to contract the heap more dynamically and aggressively than other garbage collectors, which typically require the committed portion of the heap to be contiguous. Java heap configuration for `-Xgcpolicy:balanced` strategy can be specified through the `-Xmn`, `-Xmx`, and `-Xms` options.

7.4.5 Optimal heap size

By default, the JVM provides a considerably flexible heap configuration that allows the heap to grow and shrink dynamically in response to the needs of the application. This configuration allows the JVM to claim only as much memory as necessary at any time, thus cooperating with other processes that are running on the system. The starting and maximum size of the heap can be specified with the `-Xms` and `-Xmx` options.

This flexibility comes at a cost, as the JVM must request memory from the operating system whenever the heap must grow and return memory whenever it shrinks. This behavior can lead to various unwanted scenarios. If the application heap requirements oscillate, this situation can cause excessive heap growth and shrinkage.

If the JVM is running on a dedicated machine, the processing impact of heap resizing can be eliminated by requesting a constant sized heap. This situation can be accomplished by setting `-Xms` equal to `-Xmx`. Choosing the correct size for the heap is highly important, as GC impact is directly proportional to the size of the heap. The heap must be large enough to satisfy the application's maximum memory requirements and contain extra space. The GC must work much harder when the heap is near full capacity because of fragmentation and other issues, so 20 - 30% of extra space above the maximum needs of the application can lower the overall GC impact.

If an application requires more flexibility than can be achieved with a constant sized heap, it might be beneficial to tune the sizing parameters for a dynamic heap. One of the most expensive GC events is *object allocation failure*. This failure occurs when there is not enough contiguous space in the current heap to satisfy the allocation, and results in a GC collection and a possible heap expansion. If the current heap size is less than the `-Xmx` size, the heap is expanded in response to the allocation failure if the amount of free space is below a certain threshold. Therefore, it is important to ensure that when an allocation fails, the heap is expanded to allow not only the failed allocation to succeed, but also many future allocations, or the next failed allocation might trigger yet another GC collection. This situation is known as *heap thrashing*.

The `-Xminf`, `-Xmaxf`, `-Xmine`, and `-Xmaxe` group of options can be used to affect when and how the GC resizes the heap. The `-Xminf<factor>` option (where factor is a real number 0 - 1) specifies the minimum free space in the heap; if the total free space falls below this factor, the heap is expanded. The `-Xmaxf<factor>` option specifies the maximum free space; if the total free space rises above this factor, the heap is shrunk. These options can be used to minimize heap thrashing and excessive resizing. The `-Xmine` and `-Xmaxe` options specify the minimum and maximum sizes to shrink and grow the heap by. These options can be used to ensure that the heap has enough free contiguous space to allow it to satisfy a reasonable number of allocations before failure.

Regardless of whether the heap size is constant, it should never be allowed to exceed the physical memory available to the process; otherwise, the operating system might have to swap data in and out of memory. An application's memory behavior can be determined by using various tools, including verbose GC logs. For more information about verbose GC logs and other tools, see "Java (either AIX or Linux)" on page 176.

7.5 Application scaling

Large workloads using many threads on multi-CPU machines face extra challenges regarding concurrency and scaling. In such cases, steps can be taken to decrease contention on shared resources and reduce the processing impact.

7.5.1 Choosing the correct SMT mode

AIX and Linux represent each SMT thread as a logical CPU. Therefore, the number of logical CPUs in an LPAR depends on the SMT mode. For example, an LPAR with four virtual processors that are running in SMT4 mode has 16 logical CPUs; an LPAR with that same number of virtual processors that are running in SMT2 mode has only eight logical CPUs.

Table 7-2 shows the number of SMT threads and logical CPUs available in ST, SMT2, and SMT4 modes.

Table 7-2 ST, SMT2, and SMT4 modes - SMT threads and CPUs available

SMT mode	Number of SMT threads	Number of logical CPUs
ST	1	1
SMT2	2	2
SMT4	4	4

The default SMT mode on POWER7 depends on the AIX version and the compatibility mode the processor cores are running with. Table 7-3 shows the default SMT modes.

Table 7-3 SMT mode on POWER7 is dependent upon AIX and compatibility mode

AIX version	Compatibility mode	Default SMT mode
AIX V6.1	POWER7	SMT4
AIX V6.1	POWER6/POWER6+	SMT2
AIX 5L V5.3	POWER6/POWER6+	SMT2

Most applications benefit from SMT. However, some applications do not scale with an increased number of logical CPUs on an SMT-enabled system. One way to address such an application scalability issue is to make a smaller LPAR, or use processor binding, as described in 7.5.2, “Using resource sets” on page 133. For applications that might benefit from a lower SMT mode with fewer logical CPUs, experiment with using SMT2 or ST modes (see “Hybrid thread and core” on page 79 and “Selecting different SMT modes” on page 105).

7.5.2 Using resource sets

Resource sets (RSETS) allow specifying which logical CPUs an application can run on. They are useful when an application that does not scale beyond a certain number of logical CPUs is run on a large LPAR. For example, an application that scales up to eight logical CPUs but is run on an LPAR that has 64 logical CPUs.

Resource sets can be created with the `mkrset` command and attached to a process using the `attachrset` command. An alternative way is creating a resource set and attaching it to an application in a single step through the `execrset` command.

The following example demonstrates how to use `execrset` to create an RSET with CPUs 4 - 7 and run an application that is attached to it:

```
execrset -c 4-7 -e <application>
```

In addition to running the application attached to an RSET, set the `MEMORY_AFFINITY` environment variable to `MCM` to assure that the application's private and shared memory is allocated from memory that is local to the logical CPUs of the RSET:

```
MEMORY_AFFINITY=MCM
```

In general, RSETs are created on core boundaries. For example, a partition with four POWER7 cores that are running in SMT4 mode has 16 logical CPUs. Create an RSET with four logical CPUs by selecting four SMT threads that belong to one core. Create an RSET with eight logical CPUs by selecting eight SMT threads that belong to two cores. The `smtctl` command can be used to determine which logical CPUs belong to which core, as shown in Example 7-1.

Example 7-1 Use the `smtctl` command to determine which logical CPUs belong to which core

```
# smtctl
This system is SMT capable.
This system supports up to 4 SMT threads per processor.
SMT is currently enabled.
SMT boot mode is not set.
SMT threads are bound to the same physical processor.

proc0 has 4 SMT threads.
Bind processor 0 is bound with proc0
Bind processor 1 is bound with proc0
Bind processor 2 is bound with proc0
Bind processor 3 is bound with proc0

proc4 has 4 SMT threads.
Bind processor 4 is bound with proc4
Bind processor 5 is bound with proc4
Bind processor 6 is bound with proc4
Bind processor 7 is bound with proc4
```

The `smtctl` output in Example 7-1 shows that the system is running in SMT4 mode with bind processors (logical CPU) 0 - 3 belonging to `proc0` and bind processors 4 - 7 belonging to `proc1`. Create an RSET with four logical CPUs either for CPUs 0 - 3 or for CPUs 4 - 7.

To achieve the best performance with RSETs that are created across multiple cores, all cores of the RSET must be from the same chip and in the same scheduler resource allocation domain (SRAD). The `lssrad` command can be used to determine which logical CPUs belong to which SRAD, as shown in Example 7-2:

Example 7-2 Use the `lssrad` command to determine which logical CPUs belong to which SRAD

```
# lssrad -av
REF1 SRAD      MEM      CPU
0
      0 22397.25   0-31
1
      1 29801.75   32-63
```

The output in Example 7-2 shows a system that has two SRADs. CPUs 0 - 31 belong to the first SRAD, and CPUs 32 - 63 belong to the second SRAD. In this example, create an RSET with multiple cores either using the CPUs of the first or second SRAD.

Authority for RSETs: A user must have root authority or have `CAP_NUMA_ATTACH` capability to use RSETs.

7.5.3 Java lock reservation

Synchronization and locking are an important part of any multi-threaded application. Shared resources must be adequately protected by monitors to ensure correctness, even if some resources are only infrequently shared. If a resource is primarily accessed by a single thread at any time, that thread is frequently the only thread to acquire the monitor that is guarding the resource. In such cases, the cost of acquiring the monitor can be reduced by using the `-XlockReservation` option. With this option, it is assumed that the last thread to acquire the monitor is also likely to be the next thread to acquire it. The lock is, therefore, said to be reserved for that thread, minimizing its cost to acquire and release the monitor. This option is suited to workloads using many threads and many shared resources that are infrequently shared in practice.

7.5.4 Java GC threads

The GC used by the JVM takes every opportunity to use parallelism on multi-CPU machines. All phases of the GC can be run in parallel with multiple helper threads dividing up the work to complete the task as quickly as possible. Depending on the GC strategy and heap size in use, it can be beneficial to adjust the number of threads that the GC uses. The number of GC threads can be specified with the `-Xgcthreads<number>` option. The default number of GC threads is generally equal to the number of logical processors on the partition, and it is usually not helpful to exceed this value. Reducing it, however, reduces the GC impact and might be wanted in some situations, such as when RSETs are used. The number of GC threads is capped at 64 starting in V2.6 J9 VM.

7.5.5 Java concurrent marking

The gencon policy combines concurrent marking with generational GC. If generational GC is wanted but the impact of concurrent marking, regarding both the impact of the marking thread and the extra book-keeping that is required when you allocate and manipulate objects, is not wanted, then concurrent marking can be disabled by using the `-Xconcurrentlevel10` option. This option is appropriate for workloads that benefit from the gencon policy for object allocation and lifetimes, but also require maximum throughput and minimal GC impact while the application threads are running.

In general, for both the gencon and optavgpause GC policies, concurrent marking can be tuned with the `-Xconcurrentlevel<number>` option, which specifies the ratio between the amounts of heap that is allocated and heap marked. The default value is 8. The number of low priority mark threads can be set with the `-Xconcurrentbackground<number>` option. By default, one thread is used for concurrent marking.

For more information about this topic, see 7.6, “Related publications” on page 136.

7.6 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *Java performance for AIX on POWER7 – best practices*, found at:

https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_java_performance_on_power7

- ▶ *Java Performance on POWER7*, found at:

<https://www.ibm.com/developerworks/wikis/display/LinuxP/Java+Performance+on+POWER7>

- ▶ *Top 10 64-bit IBM WebSphere Application Server FAQ*, found at:

ftp://public.dhe.ibm.com/software/webservers/appserv/WAS_64-bit_FAQ.pdf



DB2

This chapter describes the optimization and tuning of the POWER7 processor-based server running IBM DB2. It covers the following topics:

- ▶ DB2 and the POWER7 processor
- ▶ Taking advantage of the POWER7 processor
- ▶ Capitalizing on the compilers and optimization tools for POWER7
- ▶ Capitalizing on POWER7 virtualization
- ▶ Capitalizing on the AIX system libraries
- ▶ Capitalizing on performance tooling
- ▶ Conclusion

8.1 DB2 and the POWER7 processor

IBM DB2 10.1 is uniquely positioned to take advantage of the POWER7 architecture. DB2 10.1 offers many capabilities that are tailored to use POWER7 features. The DB2 self-tuning memory manager (STMM) feature is one of many features that can help DB2 workloads efficiently consolidate on POWER7 Systems. Additionally, DB2 is one of the most optimized software applications on POWER7. During the DB2 development cycles of new features, there is a specific focus on the POWER7 guidelines and the various technologies of the POWER7 architecture are characterized. So far, in the earlier chapters in this book, you read detailed descriptions about most of these POWER7 guidelines and technologies. The focus of this chapter is to showcase how IBM DB2 10.1 uses various POWER7 features and preferred practices from this guide during its own software development cycle, which is done to maximize performance on the Power Architecture. General DB2 tuning and preferred practices of DB2 10.1 are covered extensively in many other places, some of which are listed at in 8.8, “Related publications” on page 144.

8.2 Taking advantage of the POWER7 processor

Methods for taking advantage of the inherent power of the POWER7 processor include affinitization, page size, decimal arithmetics, and the usage of SMT priorities for internal lock implementation.

8.2.1 Affinitization

New to DB2 V10.1 is a much easier way to achieve affinitization on POWER7 Systems through the DB2 registry variable `DB2_RESOURCE_POLICY`. In general, this variable defines which operating system resources are available for DB2 databases or assigns specific resources to a particular database object. A typical example is to define a resource policy that restricts a DB2 database to run only on a specific set of processor cores.

On POWER7 Systems running AIX V6.1 Technology Level (TL) 5 or higher, this variable can be set to `AUTOMATIC`. This feature is a new one introduced in DB2 10.1 and can result in enhanced query performance on some workloads. With this setting, the DB2 database system automatically detects the Power hardware topology and computes the best way to assign engine dispatchable units (EDUs) to various hardware modules. The goal is to determine the most efficient way to share memory between multiple EDUs that need access to the same regions of memory. This setting is intended for larger POWER7 Systems with 16 or more cores. It is best to run a performance analysis of the workload before and after you set this variable to `AUTOMATIC` to validate the performance improvement.

The following example demonstrates how to set the registry variable to `AUTOMATIC` using the `db2set` command and then starting the DB2 database manager:

- ▶ `db2set DB2_RESOURCE_POLICY=AUTOMATIC`
- ▶ `db2start`

In DB2 10.1, `DB2_RESOURCE_POLICY` uses Scheduler Resource Allocation Domain Identifier (SRADID) instead of resource set (RSET) attachments to identify resources for the database.

For more information about other usages of `DB2_RESOURCE_POLICY` other memory-related DB2 registry variables, see Chapter 2, “AIX configuration”, in *Best Practices for DB2 on AIX 6.1 for POWER Systems*, SG24-7821.

8.2.2 Page sizes

Physical objects on disks, such as tables and indexes, are stored in pages. DB2 supports 4 KB, 8 KB, 16 KB, and 32 KB page sizes. During the processing of such objects, they are brought into DB2 buffer pools in main memory. The default AIX page size is 4 KB, but other page sizes are available. To achieve increased performance on Power Systems, DB2 10.1 by default uses 64 KB, which is a medium page size.

Large page size

For some workloads, particularly ones that require intensive memory access, there are performance benefits in using large page size support on AIX. However, certain drawbacks must be considered. When large pages support is enabled through DB2, all the memory that is set for large pages is pinned. It is possible that too much memory is allocated for large pages and not enough for 4 KB pages, which can result in heavy paging activities. Furthermore, enabling large pages prevents the STMM from automatically tuning overall database memory consumption. Consider using this variable only for well-defined workloads that have a relatively static database memory requirement.

The POWER7 large page size support can be enabled by setting the DB2 registry variable `DB2_LARGE_PAGE_MEM`.

Here are the steps to enable large page support in DB2 database system on AIX operating systems:¹

1. Configure AIX server for large pages support by running `vmo`:

```
vmo -r -o lpgg_size=LargePageSize -o lpgg_regions=LargePages
```

LargePageSize is the size in bytes of the hardware-supported large pages, and LargePages specifies the number of large pages to reserve.

2. Run `bosboot` to pick up the changes (made by running `vmo`) for the next system boot.
3. After reboot, run `vmo` to enable memory pinning:

```
vmo -o v_pinshm=1
```

4. Set the `DB2_LARGE_PAGE_MEM` registry variable by running `db2set`, then start the DB2 database manager by running `db2start`:

```
- db2set DB2_LARGE_PAGE_MEM=DB
- db2start
```

8.2.3 Decimal arithmetics

DB2 uses the hardware DFP unit in IBM POWER6 and POWER7 processors in its implementation of decimal-encoded formats and arithmetics. One example of a data type that uses this hardware support is the DECFLOAT data type that is introduced in DB2 9.5. This decimal-floating point data type supports business applications that require exact decimal values, with precision of 16 or 34 digits. When the DECFLOAT data type is used for a DB2 database that is on a POWER6 or POWER7 processor, the native hardware support for decimal arithmetic is used. In comparison to other platforms, where such business operations can be achieved only through software emulation, applications that run on POWER6 or POWER7 can use the hardware support to gain performance improvements.

¹ *DB2 Version 10.1 for Linux, UNIX, and Windows, Enabling large page support (AIX)*, available at: <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.dboj.doc%2Fdoc%2Ft0010405.html>

DECFLOAT: The Data Type of the Future describes this topic in more details. The paper is available at:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0801chainani/>

8.2.4 Using SMT priorities for internal lock implementation

DB2 uses SMT and hardware priorities in its internal lock implementation. Internal locks are short duration locks that are required to ensure consistency of various values in highly concurrent applications such as DB2. In certain cases, it is beneficial to prioritize different DB2 agent threads to maximize system resource utilization.

For more information about this topic, see 8.8, “Related publications” on page 144.

8.3 Capitalizing on the compilers and optimization tools for POWER7

DB2 10.1 is built by using an IBM XL C/C++ Version 11 compiler using various compiler optimization flags along with optimization techniques based on the common three steps of software profiling:

- ▶ Application preparation/instrumentation
- ▶ Application profiling
- ▶ Application optimization

8.3.1 Whole-program analysis and profile-based optimizations

On the AIX platform, whole-program analysis (IPA) and profile-based optimizations (PDF) compiler options are used to optimize DB2 using a set of customer representative workloads. This technique produces a highly optimized DB2 executable file that is targeted at making the best usage of the Power Architecture.

8.3.2 Feedback directed program restructuring (FDPR)

In addition to IPA/PDF optimizations using IBM XL C/C++ compiler, a post-link optimization step provides further performance improvement. The particular tool that is used is IBM Feedback Directed Program Restructuring (FDPR). Similar to IPA/PDF, a set of DB2 customer representative workloads is employed in this step, and IBM FDPR-Pro profiles and ultimately creates an optimized version of the DB2 product. For more information, see 6.3, “IBM Feedback Directed Program Restructuring” on page 114.

For more information about this topic, see 8.8, “Related publications” on page 144.

8.4 Capitalizing on POWER7 virtualization

DB2 10.1 supports and fully draws upon the virtualization technologies that are provided by the POWER7 architecture. These technologies include PowerVM for AIX and Linux and System Workload Partitioning (WPAR) for AIX. Many of the DB2 performance preferred practices for non-virtualized environment also extend to a virtualized environment.

Furthermore, DB2 offers IBM SubCapacity Licensing, which enables customers to more effectively consolidate their infrastructure and reduce their overall total cost of ownership (TCO). DB2 also provides a flexible software licensing model that supports advanced virtualization capabilities, such as shared processor pools, Micro-Partitioning, virtual machines, and dynamic reallocation of resources. To support this type of licensing model, a tool is provided that allows customers to track and manage their own software license usage.

8.4.1 DB2 virtualization

DB2 10.1 is engineered to take advantage of the many benefits of virtualization on POWER7 and therefore allows various types of workload to be deployed in a virtualized environment. One key DB2 feature that enables workloads to run efficiently in virtualized environments is STMM. STMM is designed to automatically adjust the values of several memory configuration parameters in DB2. When enabled, it dynamically evaluates and redistributes available memory resources among the buffer pools, lock memory, package cache, and sort memory to maximize performance. The changes are applied dynamically and can simplify the task of manual configuration of memory parameters. This feature is useful in a virtualized environment because STMM can respond to dynamic changes in partition memory allocation.

By default, most DB2 parameters are set to automatic to enable STMM. As a preferred practice, leave the `instance_memory` parameter and other memory parameters as automatic, especially when you are running in a virtualized environment because DB2 is designed to allow STMM to look for available memory in the system when `instance_memory` is set to automatic.

DB2 also supports the PowerVM Live Partition Mobility (LPM) feature when virtual I/O is configured. LPM allows an active database to be moved from a system with limited memory to one with more memory without disrupting the operating system or applications. When coupling dynamic LPAR (DLPAR) with STMM, the newly migrated database can automatically adjust to the additional memory resource for better performance.

DB2 Virtualization, SG24-7805, describes in considerable detail the concept of DB2 virtualization, in addition to setup, configuration, and management of DB2 on IBM Power Systems with PowerVM technology. That book follows many of the preferred practices for Power virtualization and has a list of preferred practices for DB2 on PowerVM.

8.4.2 DB2 in an AIX workload partition

DB2 supports product installation on system WPARs. Since Version 9.7, the product can be installed either within a local file system on a system WPAR or in a global environment under either the `/usr` or `/opt` directory with each instance created on the local WPARs. In both cases, each DB2 instance is only visible and managed by the system WPAR that it is created in. If DB2 is installed in a global environment, different instances on different WPARs share the globally installed DB2 copy to improve i-cache efficiency and memory usage. WPAR mobility is also supported where a DB2 instance that is running on a system WPAR can migrate to a remote WPAR on a different physical machine.

There are certain restrictions and considerations to keep in mind when you install DB2 in a global environment:

- ▶ Certain DB2 installation features cannot be installed on a system WPAR. These features are IBM Tivoli® System Automation for Multiplatforms (SA MP) and IBM Data Studio Administration Console.

- ▶ When you uninstall a DB2 copy in a global environment, all associated instances must be dropped or updated to another DB2 copy and its corresponding system WPARs must be active.
- ▶ When you apply fix packs to a DB2 copy in a global environment, all associated instances must be stopped and its corresponding system WPARs must be active.

For information about installing a DB2 copy on a WPAR, see Chapter 8, “Workload Partitioning”, in *Best Practices for DB2 on AIX 6.1 for POWER Systems*, SG24-7821.

For more information about this topic, see 8.8, “Related publications” on page 144.

8.5 Capitalizing on the AIX system libraries

This section describes methods for capitalizing on the AIX system libraries.

8.5.1 Using the `thread_post_many` API

DB2 uses `thread_wait` and `thread_post_many` to improve the efficiency of DB2 threads running on multi-processor Power Systems. DB2 takes advantage of the `thread_post_many` function. The availability of such an API on AIX directly impacts the efficiency of DB2 processing, as it allows for waking many EDUs with a single function call, which in other operating systems requires many individual function calls (typically as many as the number of EDUs being woken up).

8.5.2 File systems

DB2 uses most of the advanced features within the AIX file systems. These features include Direct I/O (DIO), Concurrent I/O (CIO), Asynchronous I/O, and I/O Completion Ports (IOCP).

Non-buffered I/O

By default, DB2 uses CIO or DIO for newly created table space containers because non-buffered I/O provides more efficient underlying storage access over buffered I/O on most workloads, with most of the benefit that is realized by bypassing the file system cache. Non-buffered I/O is configured through the `NO FILE SYSTEM CACHING` clause of the table space definition. To maximize the benefits of non-buffered I/O, a correct buffer pool size is essential. This size can be achieved by using STMM to tune the buffer pool sizes. (The default buffer pool is always tuned by STMM, but user created buffer pools must specify the **automatic** keyword for the size to allow STMM to tune them.) When STMM is enabled, it automatically adjusts the buffer pool size for optimal performance.

For file systems that support CIO, such as AIX JFS2, DB2 automatically uses this I/O method because of its performance benefits over DIO.

The DB2 log file by default uses DIO, which brings similar performance benefits as avoiding file system cache for table spaces.

Asynchronous I/O

In general, DB2 users cannot explicitly choose synchronous or asynchronous I/O. However, to improve the overall response time of the database system, minimizing synchronous I/O is preferred and can be achieved through correct database tuning. Consider the following items:

- ▶ Synchronous read I/O can occur when a DB2 agent needs a page that is not in the buffer pool to process an SQL statement. In addition, a synchronous write I/O can occur if no clean pages are available in the buffer pool to make room to bring another page from disk into that buffer pool. This situation can be minimized by having sufficiently large buffer pools or setting the buffer pool size to automatic to allow STMM to find its optimal size, in addition to tuning the page cleaning (by using the `chngpgs_thresh` database parameter).
- ▶ Not all pages read into buffer pools are done synchronously. Depending on the SQL statement, DB2 can prefetch pages of data into buffer pools through asynchronous I/O. When prefetching is enabled, two parallel activities occur during query processing: data processing and data page I/O. The latter is done through the I/O servers that wait for prefetch requests from the former. These prefetch requests contain a description of the I/O that must satisfy the query. The number of I/O servers for a database is specified through the `num_ioservers` configuration parameter. By default, this parameter is automatically tuned during database startup.

For more information about how to monitor and tune AIO for DB2, see *Best Practices for DB2 on AIX 6.1 for POWER Systems*, SG24-7821.

I/O Completion Port (IOCP)

You should configure the AIX I/O completion port for performance purposes, even though it is not mandatory, as part of the DB2 10.1 installation process. For more information, see *Configuring IOCP (AIX)*, available at:

<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.perf.doc/doc/t0054518.html>

After IOCP is configured on AIX, then DB2, by default, capitalizes on this feature for all asynchronous I/O requests. With IOCP configured, AIO server processes from the AIX operating system manage the I/O requests by processing many requests in the most optimal way for the system.

For more information about this topic, see 8.8, “Related publications” on page 144.

8.6 Capitalizing on performance tooling

Correct performance tooling is crucial for maximizing DB2 performance. Zoning in on potential performance bottlenecks is impossible without a strong performance tool set, such as the ones on Power Systems.

8.6.1 High-level investigation

During the general analysis of any performance investigation, the identification of the system resource bottlenecks is key to determining the root cause of the bottleneck. System resource bottlenecks can be classified into several categories, such as CPU bound, IO bound, network bound, or excessive idling, all of which can be identified with AIX system commands.

8.6.2 Low-level investigation

Various system level tools are essential in drilling down to find a potential root cause for the type of the bottlenecks that are listed in 8.6, “Capitalizing on performance tooling” on page 143. Profiling tools are especially invaluable for identifying CPU bound issues and are available on AIX and Linux platforms for POWER7.

AIX tprof

tprof is a powerful profiling tool on AIX and Linux platforms that does program counter-sampling in clock interrupts. It can work on any binary without recompilation and is a great tool for codepath analysis.

For instructions about using **tprof**, go to the AIX V7.1 Information Center and search for *tprof command* at:

<http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm/>

AIX tprof microprofiling

Beyond the high-level **tprof** profiling, DB2 also uses the microprofiling option of **tprof** during development. Microprofiling allows DB2 to perform instruction level profiling to attributes the CPU time spent on source program lines.

Linux OProfile

OProfile is a system profiling tool, similar in nature to **tprof**, that is popular on the Linux platform. **OProfile** uses hardware counters to provide functional level profiling in both the kernel and user space. Like **tprof**, this tool is useful during DB2 development for codepath analysis.

For more information about this topic, see 8.8, “Related publications” on page 144.

8.7 Conclusion

DB2 is positioned to capitalize on many Power features to maximize the ROI of the full IBM stack. During the entire DB2 development cycle, there is a targeted effort to take advantage of Power features and ensure that the highest level of optimization is employed on this platform. With every new Power generation, DB2 ensures that the key features are supported and brought into play at Power launch time, by working on such features well in advance of general availability. This type of targeted effort ensures that DB2 is at the forefront of optimization for Power applications.

8.8 Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this chapter:

- ▶ *Best Practices for DB2 on AIX 6.1 for Power Systems*, SG24-7821
- ▶ *Best practices for DB2 for Linux, UNIX, and Windows*, found at:

<http://www.ibm.com/developerworks/data/bestpractices/db2luw/>

- ▶ *DB2 database products in a workload partition (AIX)*, found at:
<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.qb.server.doc/doc/c0053344.html>
- ▶ DB2 performance registry variables, including **DB2_LOGGER_NON_BUFFERED_IO** and **DB2_USE_IOCP**, are described in *Performance variables*, found at:
<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.html>
- ▶ *DB2 Version 10.1 for Linux, UNIX, and Windows, DECFLOAT scalar function*, found at:
<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.sql.ref.doc/doc/r0050508.html>
- ▶ *DB2 Version 10.1 for Linux, UNIX, and Windows, Performance variables* describes DB2 performance registry variables, including **DB2_RESOURCE_POLICY** and **DB2_LARGE_PAGE_MEM**:
<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.html>
- ▶ *DB2 Virtualization, SG24-7805*
- ▶ *DECFLOAT: The data type of the future*, found at:
<http://www.ibm.com/developerworks/data/library/techarticle/dm-0801chainani/>
- ▶ *Feedback Directed Program Restructuring (FDPR)*, found at:
<https://www.research.ibm.com/haifa/projects/systems/cot/fdpr/>
- ▶ *FDPR-Pro - Usage: Feedback Directed Program Restructuring*, found at:
http://www.research.ibm.com/haifa/projects/systems/cot/fdpr/papers/fdpr_pro_usage_cs.pdf
- ▶ *IBM DB2 Version 10.1 Information Center*, found at:
<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.welcome.doc/doc/welcome.html>
- ▶ *Smashing performance with OProfile*, found at:
<http://www.ibm.com/developerworks/linux/library/l-oprof/index.html>
- ▶ *tprof Command*, found at:
<http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm>



WebSphere Application Server

This chapter describes the optimization and tuning of the POWER7 processor-based server running WebSphere Application Server.

9.1 IBM WebSphere

This chapter is intended to provide you with performance and functional considerations for running WebSphere Application Server middleware on Power Systems. It primarily describes POWER7. Even though WebSphere Application Server is designed to run on many operating systems and platforms, some specific capabilities of Power Systems are used by WebSphere Application Server as a part of platform optimization efforts.

The intent of this chapter is to explain WebSphere Application Server installation, deployment, and migration topics when WebSphere Application Server is running on Power Systems. This chapter also describes preferred practices for performance when you run enterprise Java applications on Power Systems. This chapter also highlights some of the known WebSphere Application Server topics and solutions for Power Systems.

9.1.1 Installation

As there are multiple versions of WebSphere Application Server, there are also multiple versions of AIX supported by POWER7. Table 9-1 shows some of the installation considerations.

Important: You should use the versions of WebSphere Application Server that run in *POWER7 mode with performance enhancements*.

Table 9-1 Installation considerations

Consideration	Associated website	Information about website
IBM WebSphere Application Server support on POWER7 hardware	http://www.ibm.com/support/docview.wss?uid=swg21422150	Various fix pack levels and 64-bit considerations for running in POWER7 mode

9.1.2 Deployment

When you start the WebSphere Application Server, there is an option to bind the Java processors to specific CPU processor cores to circumvent the operating system scheduler to send the work to available processors in the pool. In certain cases, this action improves the performance. Table 9-2 lists some of the deployment considerations.

Table 9-2 Deployment considerations

Consideration	Associated website	Information about website
Workload partitioning (WPAR) in AIX V6.1	http://www.ibm.com/developmentworks/aix/library/au-wpar6laix/	Determining when it is useful to move from LPAR deployment to WPAR deployment
Troubleshooting and performance analysis of different applications in versioned WPARs	http://www.ibm.com/developmentworks/aix/library/au-wpars/index.html	The benefits of moving from old hardware to the new POWER7 hardware in the form of versioned WPARs

9.1.3 Performance

When you run WebSphere Application Server on IBM POWER7 Systems, the end-to-end performance depends on many subsystems. The subsystems include network, memory, disk, and CPU subsystems of POWER7; a crucial consideration is Java configuration and tuning. Topology also plays a major role in the performance of the enterprise application that is being deployed. The architecture of the application must be considered when you determine the best deployment topology. Table 9-3 includes links to preferred practices documents, which target each of these major areas.

Table 9-3 Performance considerations

Consideration	Associated website	Information provided
<i>Java Performance on POWER7 - Best practices</i>	http://public.dhe.ibm.com/common/ssi/ecm/en/pow03066usen/POW03066USEN.PDF	This white paper highlights key preferred practices for all Java applications that are running on Power Systems and SMT considerations when you are migrating from POWER 5 or 6 to POWER7.
<i>Optimizing AIX 7 network performance: Part 1, Network overview - Monitoring the hardware</i>	http://www.ibm.com/developerworks/aix/library/au-aix7networkoptimize1/index.html	This three-part white paper reviews AIX V7.1 networking and includes suggestions for achieving the best network performance.
<i>Optimizing AIX V7 memory performance: Part 1, Memory overview and tuning memory parameters</i>	http://www.ibm.com/developerworks/aix/library/au-aix7memoryoptimize1/index.html	Memory optimization is essential for running WebSphere Application Server faster on POWER7.
<i>Optimizing AIX V7 performance: Part 2, Monitoring logical volumes and analyzing the results</i>	http://www.ibm.com/developerworks/aix/library/au-aix7optimize2/index.html	Optimizing the disk and troubleshooting the I/O bottlenecks is crucial for I/O-intensive applications.

WebSphere channel framework degradation on POWER7

Certain applications that run on WebSphere Application Server on POWER7 can experience performance degradation because of asynchronous I/O (AIO). AIO can be disabled to improve the performance of these applications. For instructions about how to accomplish this task, see *Disabling AIO (Asynchronous Input/Output) native transport in WebSphere Application Server*, available at:

<http://www.ibm.com/support/docview.wss?uid=swg21366862>

Scalability challenges when moving from POWER5 or POWER6 to POWER7

By default, POWER7 runs in SMT4 mode. As such, there are four hardware threads (or four logical CPUs) per core that provide tremendous concurrency for applications. If the enterprise applications are migrated to POWER7 from an earlier version of POWER hardware (POWER5 or POWER6), you might experience scalability issues because the default SMT mode on POWER7 is SMT4, but on POWER5 and POWER6, the default is SMT and SMT2 mode. As some of these applications might not be designed for the massive parallelism of POWER7, performance and scalability can be improved by using smaller partitions or processor binding. Processor binding is described in “Processor affinity benefits for WebSphere applications” on page 150.

Processor affinity benefits for WebSphere applications

When an application is running on top of WebSphere Application Server is deployed on a large LPAR, it might not use all the cores in that LPAR, resulting in less than optimum application performance. If this situation occurs, performance improvements to these applications can be obtained by binding the application server to certain cores. This task can be accomplished by creating the resource sets and attaching them to the application server that is running `excerset`. For an example of using the `taskset` and `numactl` commands in a Linux environment, see “Partition sizes and affinity” on page 14.

Memory affinity benefits for WebSphere applications

In addition to the processor affinity described in “Processor affinity benefits for WebSphere applications”, applications can benefit from avoiding remote memory accesses by setting the environment variable `MEMORY_AFFINITY` to `MCM`. This variable allocates application private and shared memory from processor local memory.

These three tuning techniques (SMT scalability, CPU affinity, and memory affinity) can improve the performance of WebSphere Application Server on POWER7 Systems. For an example of using the `taskset` and `numactl` commands in a Linux environment, see “Partition sizes and affinity” on page 14.

More information about these topics is in *Java Performance on POWER7 - Best practices*, found at:

<http://public.dhe.ibm.com/common/ssi/ecm/en/pow03066usen/P0W03066USEN.PDF>

9.1.4 Performance analysis, problem determination, and diagnostic tests

Resources for addressing issues regarding performance analysis, problem determination, and diagnostic tests are listed in Table 9-4.

Table 9-4 Performance analysis and problem determination

Consideration	Associated website	Information provided
Java Performance Advisor	https://www.ibm.com/developerworks/wikis/display/WikiPtype/Java+Performance+Advisor	The Java Performance Advisor (JPA) tool provides suggestions for improving the performance of Java/WebSphere Application Server applications that are running on Power Systems.
The performance detective: Where does it hurt?	http://www.ibm.com/developerworks/aix/library/au-performancedetective/index.html	Describes how to isolate performance problems.
MustGather: Performance, hang, or high CPU issues with WebSphere Application Server on AIX	http://www.ibm.com/support/docview.wss?uid=swg21052641	MustGather assists with collecting the data necessary to diagnose and resolve issues with hanging or CPU usage issues.



A

Analyzing malloc usage under AIX

This appendix describes the optimization and tuning of the POWER7 processor-based server by using the AIX `malloc` subroutine. It covers the following topics:

- ▶ Introduction
- ▶ How to collect malloc usage information

Introduction

There is a simple methodology on AIX to collect useful information about how an application uses the C heap. That information can then be used to choose and tune the appropriate malloc settings. The type of information that typically must be collected is:

- ▶ The distribution of malloc allocation sizes that are used by an application, which shows if AIX **MALLOCOPTIONS**, such as pool and buckets, are expected to perform well. This information can be used to fine-tune bucket sizes.
- ▶ The steady state size of the heap, which shows how to size the pool option.

Additional information about thread counts, malloc usage per thread, and so on, can be useful, but the information that is presented here presents a basic view.

How to collect malloc usage information

To discover the distribution of allocation sizes, set the following environment variable:

```
export MALLOCOPTIONS=buckets,bucket_statistics:stdout
```

Run an application. When the application completes, a summary of the malloc activity is output. Example A-1 shows a sample output from a simple test program.

Example A-1 Output from a simple test program

```
=====
Malloc buckets statistical summary
=====
Configuration values:
  Number of buckets: 16
  Bucket sizing factor: 32
  Blocks per bucket: 1024
Allocation request totals:
  Buckets allocator: 118870654
  Default allocator: 343383
  Total for process: 119214037
Allocation requests by bucket
Bucket      Maximum      Number of
Number      Block Size   Allocations
-----
  0          32          104906782
  1          64          9658271
  2          96          1838903
  3         128          880723
  4         160          300990
  5         192          422310
  6         224          143923
  7         256          126939
  8         288          157459
  9         320          72162
 10         352          87108
 11         384          56136
 12         416          63137
 13         448          66160
 14         480          45571
```


15	512	44080
Allocation requests by heap		
Heap	Buckets	Default
Number	Allocator	Allocator
-----	-----	-----
0	118870654	343383

This environment variable causes the program to produce a histogram of allocation sizes when it terminates. The number of allocation requests satisfied by the default allocator indicates the fraction of requests that are too large for the buckets allocator (larger than 512 bytes, in this example). By modifying some of the malloc buckets configuration options, you can, for example, obtain more details about larger allocation sizes.

To discover the steady state size of the heap, set the following environment variable:

```
export MALLOCDEBUG=log
```

Run an application to a steady state point, attach it by running **dbx**, and then run **malloc**. Example A-2 shows a sample output.

Example A-2 Sample output from the malloc subroutine

```
(dbx) malloc
The following options are enabled:
    Implementation Algorithm..... Default Allocator (Yorktown)
    Malloc Log
        Stack Depth..... 4
Statistical Report on the Malloc Subsystem:
    Heap 0
        heap lock held by..... pthread ID 0x20023358
        bytes acquired from sbrk()..... 5309664
        bytes in the freespace tree..... 334032
        bytes held by the user..... 4975632
        allocations currently active..... 76102
        allocations since process start.. 20999785
The Process Heap
    Initial process brk value..... 0x20013850
    current process brk value..... 0x214924c0
    sbrk()s called by malloc..... 78
```

The bytes held by the user value indicates how much heap space is allocated. By stopping multiple times when you run **dbx** and then running **malloc**, you can get a good estimate of the heap space needed by the application.

For more information, see *System Memory Allocation Using the malloc Subsystem*, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.genprog/doc/genprog/sys_mem_alloc.htm



B

Performance tooling and empirical performance analysis

This appendix describes the optimization and tuning of the POWER7 processor-based server from the perspective of performance tooling and empirical performance analysis. It covers the following topics:

- ▶ Introduction
- ▶ Performance advisors
- ▶ AIX
- ▶ Linux
- ▶ Java (either AIX or Linux)

Introduction

This appendix includes a general description about performance advisors, and descriptions specific to the three performance advisors that are referenced in this book:

- ▶ AIX
- ▶ Linux
- ▶ Java (either AIX or Linux)

Performance advisors

IBM developed four new performance advisors that empower users to address their own performance issues to best use their Power Systems server. These performance advisors can be run by a broad class of users.

The first three of these advisors are tools that run and analyze the configuration of a system and the software that is running on it. They also provide advice about the performance implications of the current configuration and suggestions for improvement. These three advisors are documented in “Expert system advisors” on page 156.

The fourth advisor is part of the IBM Rational Developer for Power Systems Software. It is a component of an integrated development environment (IDE), which provides a set of features for performance tuning of C and C++ applications on AIX and Linux. That advisor is documented in “Rational Performance Advisor” on page 161.

Expert system advisors

The expert system advisors are three new tools that are developed by IBM. What is unique about these applications is that they collect and interpret performance data. In one step, they collect performance metrics, analyze data, and provide a one-page visual report. This report summarizes the performance health of the environment, and includes instructions for alleviating detected problems. The performance advisors produce advice that is based on the expertise of IBM performance analysts, and IBM documented preferred practices. These expert systems focus on AIX Partition Virtualization, VIOS, and Java performance.

All of the advisors follow the same reporting format, which is a single page XML file you can use to quickly assess conditions by visually inspecting the report and looking at the descriptive icons, as shown in Figure B-1.






ICON	DEFINITION
	Informative: Context relevant data helpful in making adjustments.
	Optimal: Current condition likely to deliver best performance.
	Warning: Current condition deviates from best practices. Opportunity likely exists for better performance.
	Critical: Current condition likely causing negative impacts.
	Investigate: Further investigation or information required by user to determine if observation is impacting performance.

Figure B-1 Descriptive icons in expert system advisors (AIX Partition Virtualization, VIOS Advisor, and Java Performance Advisor)

The XML reports generated by all of the advisors are interactive. If a problem is detected, three pieces of information are shared with the user.

1. What is this?

This section explains why a particular topic was monitored, and provides a definition of the performance metric or setting.

2. Why is it Important?

This report entry explains why the topic is relevant and how it impacts performance.

3. How do I modify?

Instructions for addressing the problem are listed in this section.

VIOS Performance Advisor

The VIOS Performance Advisor provides guidance about various aspects of VIOS, including:

- ▶ CPU
- ▶ Shared processing pool
- ▶ Memory
- ▶ Fibre Channel performance
- ▶ Disk I/O subsystem
- ▶ Shared Ethernet adapter

The output is presented on a single page, and copies of the report can be saved, making it easy to document the settings and performance of VIOS over time. The goal of the advisor is for you to be able to self-assess the health of your VIOS and act to attain optimal performance.

Figure B-2 is a screen capture of the VIOS Performance Advisor, focusing on the FC adapter section of the report, which attempts to guide the user in determining if any of the FC ports are being saturated, and, if so, to what extent. An investigate image was displayed next to the idle FC port to confirm that the idle adapter port is intentional and because of an administrative configuration design choice.

The VIOS Performance Advisor can be found at:

<https://www.ibm.com/developerworks/wikis/display/WikiPtype/VIOS+Advisor>

Figure B-2 shows a screen capture from the VIOS Advisor.

VIOS - DISK ADAPTERS							
	Name	Measured Value	Recommended Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
	FC Adapter Count	2	-	06/12 20:32:25	-	n/a	n/a
	FC Avg IOps	avg: 7134 iops @ 4KB		06/12 20:32:25	06/12 20:37:25	n/a	n/a
	FC Avg IOps(fcs2)	idle	-	06/12 20:32:25	06/12 20:37:25	n/a	n/a
	FC Avg IOps(fcs3)	avg: 7134 iops @ 4KB peak: 7251 iops @ 4KB	-	06/12 20:32:25	06/12 20:37:25	n/a	n/a
	FC Adapter Utilization			-	-	n/a	n/a
	FC Adapter Utilization(fcs2)	idle	-	06/12 20:32:44	06/12 20:37:16	4	4
	FC Adapter Utilization(fcs3)	high:14.3% util. avg:14.1%	-	06/12 20:32:44	06/12 20:37:16	4	4

Figure B-2 The VIOS Advisor

Virtualization Performance Advisor

The Virtualization Performance Advisor provides guidance for various aspects of an LPAR, both dedicated and shared, including:

- ▶ LPAR physical memory domain allocation
- ▶ Physical CPU entitlement and virtual CPU optimization
- ▶ SMT effectiveness
- ▶ Processor folding effectiveness
- ▶ Shared processing pool
- ▶ Memory optimization
- ▶ Physical Fibre Channel adapter optimization
- ▶ Virtual disk I/O optimization (vSCSI and NPIV)

The output is presented on a single page, and copies of the report can be saved, making it easy for the user to document the settings and performance of their LPAR over time. The goal of the advisor is for the user to be able to self-assess the health of their LPAR and act to attain optimal performance.

Figure B-3 is a snapshot of the LPAR performance advisor, focusing on the LPAR optimization section of the report, which applies virtualization preferred practice guidance to LPAR configuration, resource usage of the LPAR, and shared processor pool, and determines if the LPAR configuration is optimized. If the advisor finds that the LPAR configuration is not optimal for the workload, it guides the user in determining the best possible configuration. The LPAR Performance Advisor can be found at:

<https://www.ibm.com/developerworks/wikis/display/WikiPtype/PowerVM+Virtualization+performance+advisor>

LPAR PROCESSOR OPTIMIZATION																																											
	Name	Current Value	Recommended Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest																																				
✘	Lpar Placement Optimization	Placement <table border="1"> <thead> <tr> <th>Global Domain</th> <th>Chip Domain</th> <th>Memory</th> <th>CPU</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>63228.560</td> <td>0-11</td> </tr> <tr> <td>0</td> <td>6</td> <td>60756.000</td> <td></td> </tr> <tr> <td>1</td> <td>1</td> <td>63246.000</td> <td>12-15</td> </tr> <tr> <td>1</td> <td>2</td> <td>63246.000</td> <td>16-19</td> </tr> <tr> <td>2</td> <td>3</td> <td>63223.000</td> <td>20-23</td> </tr> <tr> <td>2</td> <td>4</td> <td>63478.690</td> <td>24-27</td> </tr> <tr> <td>3</td> <td>5</td> <td>61503.000</td> <td>28-31</td> </tr> <tr> <td>3</td> <td>7</td> <td>52539.000</td> <td></td> </tr> </tbody> </table> Only memory assigned from 0 Global Domain 6 Chip Domain Only memory assigned from 3 Global Domain 7 Chip Domain	Global Domain	Chip Domain	Memory	CPU	0	0	63228.560	0-11	0	6	60756.000		1	1	63246.000	12-15	1	2	63246.000	16-19	2	3	63223.000	20-23	2	4	63478.690	24-27	3	5	61503.000	28-31	3	7	52539.000		Memory is allocated from multiple domains, containing memory in one or fewer domains will improve performance. However, changing this allocation requires assistance from IBM. Also, refer to P7 Virtualization Performance best practice document.	Tue Mar 6 20:14:28 2012	-	NA	NA
		Global Domain	Chip Domain	Memory	CPU																																						
0	0	63228.560	0-11																																								
0	6	60756.000																																									
1	1	63246.000	12-15																																								
1	2	63246.000	16-19																																								
2	3	63223.000	20-23																																								
2	4	63478.690	24-27																																								
3	5	61503.000	28-31																																								
3	7	52539.000																																									
✘		Local Memory access - 0.00 and Remote memory access - 1210.84 Remote memory access is high. Local Memory access - 0.00 and Distant memory access - 9275.13 Distant memory access is high.	Memory is allocated from multiple domains, containing memory in one or fewer domains will improve performance. However, changing this allocation requires assistance from IBM. Also, refer to P7 Virtualization Performance best practice document.	Tue Mar 6 20:14:28 2012	-	NA	NA																																				
✔	SMT Effectiveness	SMT-4 is set	-	Tue Mar 6 20:14:28 2012	-	NA	NA																																				
✔	Virtual Processor Folding Optimization	Virtual Processor Folding Threshold - 49%	Rerun when lpar is busy	Tue Mar 6 20:14:33 2012	-	NA	NA																																				

Figure B-3 LPAR Virtualization Advisor

Java Performance Advisor

The Java Performance Advisor provides recommendations to improve performance of a stand-alone Java or WebSphere Application Server application that is running on an AIX machine. The guidance that is provided is categorized into four groups as follows:

- ▶ Hardware and LPAR-related parameters: Processor sharing, SMT levels, memory, and so on
- ▶ AIX specific tunables: Process RSET, TCP buffers, memory affinity, and so on

- ▶ JVM tunables: Heap sizing, garbage collection (GC) policy, page size, and so on
- ▶ WebSphere Application Server related settings for a WebSphere Application Server process

The guidance is based on Java tuning preferred practices. The criteria that are used to determine the guidance include the relative importance of the Java application, machine usage (test/production), and the user's expertise level.

Figure B-4 is a snapshot of Java and WebSphere Application Server recommendations from a sample run, indicating the best JVM optimization and WebSphere Application Server settings for better results, as per Java preferred practices. Details about the metrics can be obtained by expanding each of the metrics. The output of the run is a simple XML file that can be viewed by using the supplied XSL viewer and any browser. The Java Performance Advisor can be found at:

<https://www.ibm.com/developerworks/wikis/display/WikiPtype/Java+Performance+Advisor>

Java					
	Name	Current Value	Recommended Value	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
	JVM Version	1.6.0 SR2	More Details...	4	4
	JVM Type	64 bit	32 bit	4	4
	Initial Heap Size	100 MB	400 MB to 1.5625 GB	2	3
	Maximum Heap Size	1.5625 GB	1.5625 GB	5	5
	JVM Debug	Off	Off	1	5
	Verbose Class Loading	Off	Off	1	2
	Verbose Garbage Collection	Off	On	1	1
WebSphere					
	Name	Current Value	Recommended Value	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
	WebSphere Version	7.0.0.0	More Details...	3	4
	WebSphere PMI	On	Off	3	5
	Session Time Out	30 Minutes	5 Minutes to 30 Minutes	3	1
	Minimum Web Container Threads	50 Threads	10 Threads to 72 Threads	3	3
	Maximum Web Container Threads	50 Threads	24 Threads to 144 Threads	4	5

Figure B-4 Java Performance Advisor

Rational Performance Advisor

IBM Rational Developer for Power Systems Software IDE V8.5 introduces a new component that is called Performance Advisor, which provides a rich set of features for performance tuning C and C++ applications on IBM AIX and IBM PowerLinux systems. Although not directly related to the tooling described in “Expert system advisors” on page 156, Rational Performance Advisor has the same goal of helping users to best use Power hardware with tooling that offers simple collection, management, and analysis of performance data.

Performance Advisor gathers data from several sources. The raw application performance data comes from the same expert-level `tprof` and `OProfile` CPU profilers described in “AIX” on page 162 and “Linux” on page 171, and other low-level operating system tools. The debug information that is generated by the compiler allows this data to be matched back to the original source code. XLC compilers can generate XML report files that provide information about optimizations that were performed during compilation. Finally, the application build and runtime systems are analyzed to determine whether there are any potential environmental problems.

All of this data is automatically gathered, correlated, analyzed, and presented in a way that is quick to access and easy to understand (Figure B-5).

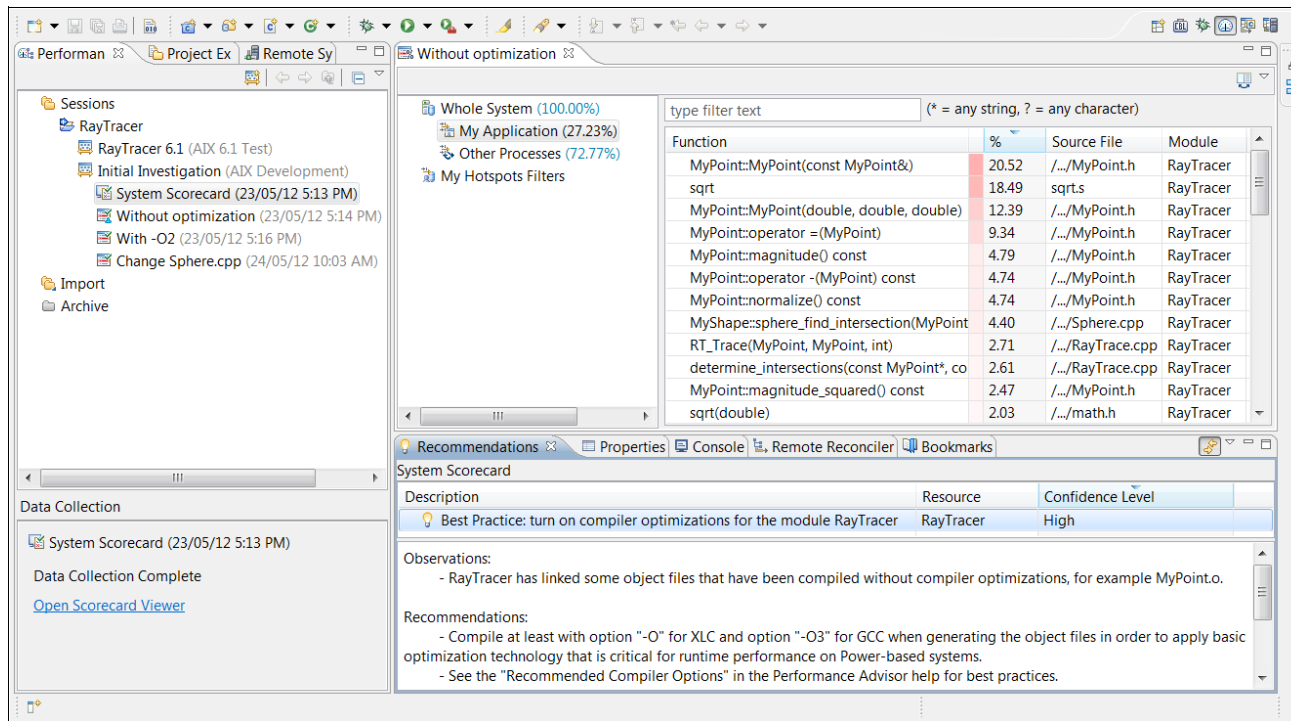


Figure B-5 Rational Performance Advisor

Key features include:

- ▶ Performance Explorer organizes your performance tuning sessions and data.
- ▶ System Scorecard reports on your Power build and runtime environments.
- ▶ Hotspots Browser shows CPU profiling results for your application and its functions.
- ▶ Hotspots Comparison Browser compares runs for regression analysis or fix verification.
- ▶ The Performance Source Viewer and Outline view gives precise line-level profiling results.
- ▶ Invocations Browser displays dynamic call information from your application
- ▶ The Recommendations view offers expert-system guidance.

More information about Rational Performance Advisor, including a trial download, can be found in *Rational Developer for Power Systems Software*, available at:

<http://www.ibm.com/software/rational/products/rdp/>

AIX

The section introduces tools and techniques that are used for optimizing software for a combination of Power Systems and AIX. The intended audience for this section is software development teams. As such, this section does not address performance topics that are related to capacity planning, and system-level performance monitoring and tuning.

For capacity planning, see the IBM Systems Workload Estimator, available at:

<http://www-912.ibm.com/estimator>

For system-level performance monitoring and tuning information for AIX, see *Performance Management*, available at:

http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

The bedrock of any empirically based software optimization effort is a suite of repeatable benchmark tests. To be useful, such tests must be representative of the manner in which users interact with the software. For many commercial applications, a benchmark test simulates the actions of multiple users that drive a prescribed mix of application transactions. Here, the fundamental measure of performance is throughput (the number of transactions that are run over a period) with an acceptable response time. Other applications are more *batch-oriented*, where few jobs are started and the time that is taken to completion is measured. Whichever benchmark style is used, it must be repeatable. Within some small tolerance (typically a few percent), running the benchmark several times on the same setup yields the same result.

Tools and techniques that are employed in software performance analysis focus on pinpointing aspects of the software that inhibit performance. At a high level, the two most common inhibitors to application performance are:

- ▶ Areas of code that consume large amounts of CPU resources. This code is usually caused by using inefficient algorithms, poor coding practices, or inadequate compiler optimization
- ▶ Waiting for locks or external events. Locks are used to serialize execution through critical sections, that is, sections of code where the need for data consistency requires that only one software thread run at a time. An example of an external event is the system that is waiting for a disk I/O to complete. Although the amount of time that an application must wait for external events might be outside of the control of the application (for example, the time that is required for a disk I/O depends on the type of storage employed), simply being aware that the application is having to wait for such an event can open the door to potential optimizations.

CPU profiling

A CPU profiler is a performance tool that shows in which code CPU resources are being consumed. **tprof** is a powerful CPU profiler that encompasses a broad spectrum of profiling functionality:

- ▶ It can profile any program, library, or kernel extension that is compiled with C, C++, Fortran, or Java compilers. It can profile machine code that is created in real time by the JIT compiler.
- ▶ It can attribute time to processes, threads, subroutines (user mode, kernel mode, shared library, and Java methods), source statements, and even individual machine instructions.
- ▶ In most cases, no recompilation of object files is required.

Usage of **tprof** typically focuses on generating subroutine-level profiles to pinpoint code hotspots, and to examine the impact of an attempted code optimization. A common way to invoke **tprof** is as follows:

```
$ tprof -E -skeuz -x sleep 10
```

The **-E** flag instructs **tprof** to employ the PMU as the sampling mechanism to generate the profile. Using the PMU as the sampling mechanism provides a more accurate profile than the default time-based sampling mechanism, as the PMU sampling mechanism can accurately sample regions of kernel code where interrupts are disabled. The **s**, **k**, **e**, and **u** flags instruct **tprof** to generate subroutine-level profiles for shared library, kernel, kernel extension, and user-level activity. The **z** flag instructs **tprof** to report CPU time in the number of *ticks* (that is, samples), instead of percentages. The **-x sleep 10** argument instructs **tprof** to collect profiling data during the running of the **sleep 10** command. This command collects profile data over the entire system (including all running processes) over a period of 10 seconds.

Excerpts from a **tprof** report are shown in Example B-1, Example B-2 on page 164, and Example B-3 on page 164.

Example B-1 is a breakdown of samples of the processes that are running on the system. When multiple processes have the same name, they have only one line in this report: the number of processes with that name is in the “Freq” column. “Total” is the total number of samples that are accumulated by the process, and “Kernel”, “User”, and “Shared” are the number of samples that are accumulated by the processes in kernel (including kernel extensions), user space, and shared libraries. “Other” is a catchall for samples that do not fall in the other categories. The most common scenario where samples wind up in “Other” is because of CPU resources that are being consumed by machine code that is generated in real time by the JIT compiler. The **-j** flag of **tprof** can be used to attribute these samples to Java methods.

Example B-1 Excerpt from a tprof report - breakdown of samples of processes running on the system

Process	Freq	Total	Kernel	User	Shared	Other	
wait	4	4	5810	5810	0	0	0
./version1	1	1	1672	35	1637	0	0
/usr/bin/tprof	2	2	15	13	0	2	0
/etc/syncd	1	1	2	2	0	0	0
/usr/bin/sh	2	2	2	2	0	0	0
swapper	1	1	1	1	0	0	0
/usr/bin/trcstop	1	1	1	1	0	0	0
rmcd	1	1	1	1	0	0	0
Total	13	13	7504	5865	1637	2	0

Example B-2 is a breakdown of samples of the threads that are running on the system. In addition to the columns described in Example B-1 on page 163, this report has *PID* and *TID* columns that detail the process IDs and thread IDs.

Example B-2 Excerpt from a tprof report - breakdown of threads that are running on the system

Process	PID	TID	Total	Kernel	User	Shared	Other	
	=====	===	===	=====	=====	=====	=====	=====
	wait	16392	16393	1874	1874	0	0	0
	wait	12294	12295	1873	1873	0	0	0
	wait	20490	20491	1860	1860	0	0	0
	./version1	245974	606263	1672	35	1637	0	0
	wait	8196	8197	203	203	0	0	0
	/usr/bin/tprof	291002	643291	13	13	0	0	0
	/usr/bin/tprof	274580	610467	2	0	0	2	0
	/etc/syncd	73824	110691	2	2	0	0	0
	/usr/bin/sh	245974	606263	1	1	0	0	0
	/usr/bin/sh	245976	606265	1	1	0	0	0
	/usr/bin/trcstop	245976	606263	1	1	0	0	0
	swapper	0	3	1	1	0	0	0
	rmcd	155876	348337	1	1	0	0	0
	=====	===	===	=====	=====	=====	=====	=====
	Total			7504	5865	1637	2	0

Total Samples = 7504 Total Elapsed Time = 18.76s

Example B-3 from the report gives the subroutine-level profile for the version1 program. In this simple example, all of the time is spent in **main()**.

Example B-3 Excerpt from a tprof report - subroutine-level profile for the version1 program, with all time spent in main()

```

Profile: ./version1
Total Ticks For All Processes (./version1) = 1637
Subroutine  Ticks  %  Source  Address  Bytes
=====  =====  =====  =====  =====  =====
      .main  1637  21.82  version1.c  350  536

```

More information about using AIX **tprof** for Java programs is available in “Hot method or routine analysis” on page 177.

The functionality of **tprof** is rich. As such, it cannot be fully described in this guide. For complete **tprof** documentation, see *tprof Command*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.commands/doc/aixcmds5/tprof.htm>

AIX trace-based analysis tools

Trace¹ is a powerful utility that is provided by AIX for collecting a time-sequenced log of operating system events on a Power Systems server. The AIX kernel and kernel extensions are richly instrumented with trace *hooks* that, when trace is activated, append trace records with context-relevant data, to a pinned, kernel-resident trace buffer. These records can be later read from that buffer and logged to a disk-resident file. Further utilities are provided to interpret and summarize trace logs and generate human-readable reports. The **tprof** CPU profiler is one such utility. Besides **tprof**, two of the most-commonly used trace-based utilities are **curt**² and **splat**.^{3,4}

The **curt** command takes as its input a trace collected using the AIX trace facility, and generates a report that breaks down how CPU time is consumed by various entities, including:

- ▶ Processes (grouped by process name)
- ▶ Individual processes
- ▶ Individual threads
- ▶ System calls (either on a system-wide or per-thread basis)
- ▶ Interrupts

One of the most useful reports from **curt** is the *System Calls Summary*. This report provides a system-wide summary of the system calls that are executed while the trace is collected. For each system call, the following information is provided:

- ▶ Count: The number of times the system call was run during the monitoring interval
- ▶ Total Time: Amount of CPU time (in milliseconds) consumed in running the system call
- ▶ % sys time: Percentage of overall CPU capacity that is spent in running the system call
- ▶ Avg Time: Average CPU time that is consumed for each execution of the system call
- ▶ Min Time: Minimum CPU time that is consumed during an execution of the system call
- ▶ Max Time: Maximum CPU time that is consumed during an execution of the system call
- ▶ SVC: Name and address of the system call

An excerpt from a System Calls Summary report is shown in Example B-4.

Example B-4 System Calls Summary report (excerpt)

System Calls Summary						
Count	Total Time (msec)	% sys time	Avg Time (msec)	Min Time (msec)	Max Time (msec)	SVC (Address)
123647	3172.0694	14.60%	0.0257	0.0128	0.9064	kpread(2a2d5e8)
539	1354.6939	6.24%	2.5133	0.0163	4.1719	listio64(516ea40)
26496	757.6204	3.49%	0.0286	0.0162	0.0580	_esend(2a29f88)
26414	447.7029	2.06%	0.0169	0.0082	0.0426	_erecv(2a29e98)

¹ *trace Daemon*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/trace.htm>

² *CPU Utilization Reporting Tool (curt)*, available at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_cpu.htm

³ *Simple performance lock analysis tool (splat)*, available at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_splat.htm

⁴ *splat Command*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/splat.htm>

9907	266.1382	1.23%	0.0269	0.0143	0.5350	kpwrite(2a2d588)
34282	167.8132	0.77%	0.0049	0.0032	0.0204	_thread_wait(2a28778)

As a first step, compare the mix of system calls to the expectation of how the application is expected to behave. Is the mix aligned with expectations? If not, first confirm that the trace is collected while the wanted workload runs. If the trace is collected at the correct time and the mix still differs from expectations, then investigate the application logic. Also, examine the list of system calls for potential optimizations. For example, if **select** or **poll** is used frequently, consider employing the pollset facility (see “pollset” on page 71).

As a further breakdown, **curt** provides a report of the system calls run by each thread. An example report is shown in Example B-5.

Example B-5 system calls run by each thread

Report for Thread Id: 549305 (hex 861b9) Pid: 323930 (hex 4f15a)
Process Name: proc1

```

-----
Total Application Time (ms): 89.010297
Total System Call Time (ms): 160.465531
Total Hypervisor Call Time (ms): 18.303531
      Thread System Call Summary
      -----
Count   Total Time   Avg Time   Min Time   Max Time   SVC (Address)
=====  =====
492     157.0663     0.3192    0.0032    0.6596    listio64(516ea40)
494       3.3656     0.0068    0.0002    0.0163    GetMultipleCompletionStatus(549a6a8)
12       0.0238     0.0020    0.0017    0.0022    _thread_wait(2a28778)
6        0.0060     0.0010    0.0007    0.0014    thread_unlock(2a28838)
4        0.0028     0.0007    0.0005    0.0008    thread_post(2a288f8)
-----

```

Another useful report that is provided by **curt** is the *Pending System Calls Summary*. This summary shows the list of threads that are in an unfinished system call at the end of the trace. An example report is given in Example B-6.

Example B-6 Threads that are in an unfinished system call at the end of the trace

```

Pending System Calls Summary
-----
Accumulated   SVC (Address)                               Procname (Pid Tid)
Time (msec)
=====
0.0082    GetMultipleCompletionStatus(549a6a8)        proc1(323930 532813)
0.0089    _nsleep(2a28d30)                            proc2(270398 545277)
0.0054    _thread_wait(2a28778)                      proc1(323930 549305)
0.0088    GetMultipleCompletionStatus(549a6a8)        proc1(323930 561437)
3.3981    listio64(516ea40)                          proc1(323930 577917)
0.0130    kpwrite(2a2d588)                            proc1(323930 794729)
-----

```

For each thread in an unfinished system call, the following items are provided:

- ▶ The accumulated time in the system call
- ▶ The name of the system call (followed by the system call address in parentheses)
- ▶ The process name, followed by the Process ID and Thread ID in parentheses

This report is useful in determining what system calls are blocking threads from proceeding. For example, threads appearing in this report with an unfinished `recv` call are waiting on data to be received over a socket.

Another useful trace-based tool is `splat`, which is the Simple Performance Lock Analysis Tool. The `splat` tool provides reports about the usage of kernel and application (pthread-level) locks. At the pthread level, `splat` can report about the usage of pthread synchronizers: mutexes, read/write locks, and condition variables. Importantly, `splat` provides data about the degree of contention and blocking on these objects, an important consideration in creating highly scalable and pthread-based applications.

The pthread library instrumentation does not provide names or classes of synchronizers, so the addresses are the only way that you have to identify them. Under certain conditions, the instrumentation can capture the return addresses of the function call stack, and these addresses are used with the output of the `gensyms` tool to identify the call chains when these synchronizers are created. The creation and deletion times of the synchronizer can sometimes be determined as well, along with the ID of the pthread that created them.

An example of a mutex report from `splat` is shown in Example B-7.

Example B-7 Mutex report from splat

```
[pthread MUTEX] ADDRESS: 00000000F0154CD0
Parent Thread: 0000000000000001 creation time: 26.232305
Pid: 18396 Process Name: trcstop
Creation call-chain =====
00000000D268606C .pthread_mutex_lock
00000000D268EB88 .pthread_once
00000000D01FE588 .__libs_init
00000000D01EB2FC ne_callbacks
00000000D01EB280 .__libc_declare_data_functions
00000000D269F960 .__pth_init_libc
00000000D268A2B4 .pthread_init
00000000D01EAC08 .__modinit
000000001000014C .__start

Acqui- | Miss Spin Wait Busy | Secs Held | Percent Held ( 26.235284s )
sitions | Rate Count Count Count | CPU Elapsed | CPU Elapsed Spin Wait
1 | 0.000 0 0 0 | 0.000006 0.000006 | 0.00 0.00 0.00 0.00

-----
Depth Min Max Avg
SpinQ 0 0 0
WaitQ 0 0 0
Recursion 0 1 0

PThreadID Acqui- Miss Spin Wait Busy Percent Held of Total Time
sitions Rate Count Count Count CPU Elapse Spin Wait
1 1 0.00 0 0 0 0.00 0.00 0.00 0.00

Function Name Acqui- Miss Spin Wait Busy Percent Held of Total Time
Offset sitions Rate Count Count Count CPU Elapse Spin Wait Return Address Start Address
-----
pthread_once 0 0.00 0 0 0 99.99 99.99 0.00 0.00 00000000D268EC98 00000000D2684180
pthread_once 1 0.00 0 0 0 0.01 0.01 0.00 0.00 00000000D268EB88 00000000D2684180
```

In addition to the common header information and the `[pthread MUTEX]` identifier, this report lists the following lock details:

- Parent thread** Pthread ID of the parent pthread
- Creation time** Elapsed time in seconds after the first event recorded in trace (if available)

Deletion time	Elapsed time in seconds after the first event recorded in trace (if available)	
PID	Process identifier	
Process Name	Name of the process using the lock	
Call-chain	Stack of called methods (if available)	
Acquisitions	The number of times the lock was acquired in the analysis interval	
Miss Rate	The percentage of attempts that failed to acquire the lock	
Spin Count	The number of unsuccessful attempts to acquire the lock	
Wait Count	The number of times a thread is forced into a suspended wait state while waiting for the lock to come available	
Busy Count	The number of trylock calls that returned busy	
Seconds Held	This field contains the following subfields:	
	CPU	The total number of processor seconds the lock is held by a running thread.
	Elapse(d)	The total number of elapsed seconds the lock is held, whether the thread was running or suspended.
Percent Held	This field contains the following subfields:	
	Real CPU	The percentage of the cumulative processor time the lock was held by a running thread.
	Real Elapsed	The percentage of the elapsed real time the lock is held by any thread, either running or suspended.
	Comb(ined) Spin	The percentage of the cumulative processor time that running threads spend spinning while it tries to acquire this lock.
	Real Wait	The percentage of elapsed real time that any thread was waiting to acquire this lock. If two or more threads are waiting simultaneously, this wait time is only charged one time. To learn how many threads are waiting simultaneously, look at the WaitQ Depth statistics.
Depth	This field contains the following subfields:	
	SpinQ	The minimum, maximum, and average number of threads that are spinning on the lock, whether running or suspended, across the analysis interval
	WaitQ	The minimum, maximum, and average number of threads that are waiting on the lock, across the analysis interval
	Recursion	The minimum, maximum, and average recursion depth to which each thread held the lock

Finding alignment issues

Improperly aligned code or data can cause performance degradation. By default, the IBM compilers and linkers correctly align code and data, including stack and statically allocated variables. Incorrect typecasting can result in references to storage that are not correctly aligned. There are two types of alignment issues to be concerned with:

- ▶ Alignment issues that are handled by microcode in the POWER7 processor
- ▶ Alignment issues that are handled through alignment interrupts.

Examples of alignment issues that are handled by microcode with a performance penalty in the POWER7 processor are loads that cross a 128-byte boundary and stores that cross a 4 KB page boundary. To give an indication of the penalty for this type of misalignment, on a 4 GHz processor, a nine-instruction loop that contains an 8 byte load that crosses a 128-byte boundary takes double the time of the same loop with the load correctly aligned.

Alignment issues that are handled by microcode can be detected by running **hpmcount** or **hpmstat**. The **hpmcount** command is a command-line utility that runs a command and collects statistics from the POWER7 PMU while the command runs. To detect alignment issues that are handled by microcode, run **hpmcount** to collect data for group 38. An example is provided in Example B-8.

Example B-8 Example of the results of the hpmcount command

```
# hpmcount -g 38 ./unaligned
Group: 38
Counting mode: user
Counting duration: 21.048874056 seconds
PM_LSU_FLUSH_ULD (LRQ unaligned load flushes)      :      4320840034
PM_LSU_FLUSH_UST (SRQ unaligned store flushes)     :                0
PM_LSU_FLUSH_LRQ (LRQ flushes)                    :      450842085
PM_LSU_FLUSH_SRQ (SRQ flushes)                    :                149
PM_RUN_INST_CMPL (Run instructions completed)      :      19327363517
PM_RUN_CYC (Run cycles)                            :      84219113069
Normalization base: time
Counting mode: user
Derived metric group: General
[  ] Run cycles per run instruction                  :                4.358
```

The **hpmstat** command is similar to **hpmcount**, except that it collects performance data on a system-wide basis, rather than just for the execution of a command.

Generally, scenarios in which the ratio of (*LRQ unaligned load flushes* + *SRQ unaligned store flushes*) divided by *Run instructions completed* is greater than 0.5% must be further investigated. The **tprof** command can be used to further pinpoint where in the code the unaligned storage references are occurring. To pinpoint unaligned loads, the **-E PM_MRK_LSU_FLUSH_ULD** flag is added to the **tprof** command line, and to pinpoint unaligned stores, the **-E PM_MRK_LSU_FLUSH_UST** flag is added. When these flags are used, **tprof** generates a profile where unaligned loads and stores are sampled instead of time-based sampling.

Examples of alignment issues that cause an alignment interrupt include execution of a `lmmw` or `lwarx` instruction on a non-word-aligned boundary. These issues can be detected by running `alstat`. This command can be invoked with an interval, which is the number of seconds between each report. An example is presented in Example B-9.

Example B-9 Alignment issues can be addressed with the alstat command

```
> alstat 5
Alignment  Alignment
SinceBoot   Delta
    2016      0
    2016      0
    2016      0
    2016      0
    2016      0
    2016      0
```

The key metric in the `alstat` report is the Alignment Delta. This metric is the number of alignment interrupts that occurred during the interval. Non-zero counts in this column merit further investigation with `tprof`. Invoking `tprof` with the `-E ALIGNMENT` flag generates a profile that shows where the unaligned references are occurring.

For more information, see *alstat Command*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.commands/doc/aixcmds1/alstat.htm>

Finding emulation issues

Over the 20+ year evolution of the Power instruction set, a few instructions were removed. Instead of trapping programs that run these instructions, AIX emulates them in the kernel, although with a significant processing impact. Generally, programs that are written in a third-generation language (for example, C and C++) and compiled with an up-to-date compiler do not contain these emulated instructions. However, older binary files or older hand-written assembly language might contain such instructions, and because they are silently emulated by AIX, the performance penalty might not be readily apparent.

The `emstat` command detects the presence of these instructions. Like `alstat`, it is invoked with an interval, which is the number of seconds between reports. An example is shown in Example B-10.

Example B-10 The emstat command detects the presence of emulated instructions

```
> emstat 5
Emulation  Emulation
SinceBoot   Delta
    0        0
    0        0
    0        0
    0        0
    0        0
```

The key metric is the Emulation Delta (the number of instructions that are emulated during each interval). Non-zero values merit further investigation. Invoking `tprof` with the `-E EMULATION` flag generates a profile that shows where the emulated instructions are.

For more information, see *emstat Command*, available at:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.commands/doc/aixcmds2/emstat.htm>

hpmstat, hpmcount, and tprof -E

The POWER7 processor provides a powerful on-chip PMU that can be used to count the number of occurrences of performance-critical processor events. A rich set of events is countable; examples include level 2 and level 3 d-cache misses, and cache reloads from local, remote, and distant memory. *Local memory* is memory that is attached to the same POWER7 processor chip that the software thread is running on. *Remote memory* is memory that is attached to a different POWER7 processor that is in the same CEC (that is, the same node or building block in the case of a multi-CEC system, such as a Power 780) that the software thread is running on. *Distant memory* is memory that is attached to a POWER7 processor that is in a different CEC from the CEC the software thread is running on.

Two commands exist to count PMU events: **hpmcount** and **hpmstat**. The **hpmcount** command is a command-line utility that runs a command and collects statistics from the PMU while the command runs. The **hpmstat** command is similar to **hpmcount**, except that it collects performance data on a system-wide basis, rather than just for the execution of a command.

Further documentation about **hpmcount** and **hpmstat** can be found at:

- ▶ <http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.commands/doc/aixcmds2/hpmcount.htm>
- ▶ <http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.commands/doc/aixcmds2/hpmstat.htm>

In addition to simply counting processor events, the PMU can be configured to sample instructions based on processor events. With this capability, profiles can be generated that show which parts of an application are experiencing specified processor events. For example, you can show which subroutines of an application are generating level 2 or level 3 cache misses. The **tprof** profiler includes this functionality through the **-E** flag, which allows a PMU event name to be provided to **tprof** as the sampled event. The list of PMU events can be generated by running **pm1ist -c -1**. Whenever possible, perform profiling using *marked* events, as profiling using marked events is more accurate than using unmarked events. The marked events begin with the prefix `PM_MRK_`.

For more information about using the **-E** flag of **tprof**, go to:

<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.commands/doc/aixcmds5/tprof.htm>

Linux

The section introduces tools and techniques used for optimizing software on the combination of Power Systems and Linux. The intended audience for this section is software development teams.

Empirical performance analysis using the IBM SDK for PowerLinux

After you apply the best high-level optimization techniques, a deeper level of analysis might be required to gain more performance improvements. You can use the IBM SDK for PowerLinux to help you gain these improvements.

The IBM SDK for PowerLinux is a set of tools that support:

- ▶ Hot spot analysis
- ▶ Analysis of ported code for missed platform-specific optimization
- ▶ Whole program analysis for coding issues, for example, pipeline hazards, inlining opportunities, early exits and hidden path length, devirtualization, and branch prediction hints
- ▶ Lock contention and IO delay analysis

The IBM SDK for PowerLinux can be found at:

<http://www14.software.ibm.com/webapp/set2/sas/f/lopdiags/sdklop.html>

The SDK provides an Eclipse C/C++ IDE with Linux tools integrations. The SDK provides graphical presentation and source code view integration with Linux execution profiling (**gprof/Oprofile**), malloc and memory usage (**valgrind**), pthread synchronization (**helgrind**), SystemTap tapsets, and tapset development.

Hotspot analysis

You should profile the application and look for hotspots. When you run the application under one or more representative workloads, use a hardware-based profiling tool such as **OProfile**. **OProfile** can be run directly as a command-line tool or under the IBM SDK for PowerLinux.

The **OProfile** tools can monitor the whole system (LPAR), including all the tasks and the kernel. This action requires root authority, but is the best way to profile the kernel and complex applications with multiple cooperating processes. **OProfile** is fully enabled to take samples using the full set of the PMU events (run **ophe1p** for a complete list of events). **OProfile** can produce text file reports organized by process, program and libraries, function symbols, and annotated source file and line number or machine code disassembly.

The IBM SDK can profile applications that are associated with Eclipse projects. The SDK automates the setup and running of the profile, but is restricted to a single application, its libraries, and direct kernel calls. The SDK is easier to use, as it is hierarchically organized by percentage with program, function symbol, and line number. Clicking the line number in the profile pane *jumps* the source view pane to the matching source file and line number. This action simplifies edit, compile, and profile tuning activities.

The whole system profile is a good place to start. You might find that your application is consuming most of the CPU cycles, and deeper analysis of the application is the next logical step. The IBM SDK for PowerLinux provides a number of helpful tools, including integrated application profiling (**OProfile** and **valgrind**), Migration Assistant, and the Source Code Advisor.

High kernel usage

If the bulk of the CPU cycles are consumed in the kernel or runtime libraries that are not part of your application, then a different type of analysis is required. If the kernel is consuming significant cycles, then the application might be I/O or lock contention bound. This situation can occur when an application moves to larger systems (higher core count) and fails to scale up.

I/O bound applications can be constrained by small buffer sizes or a poor choice of an access method. One issue to look for is applications that use local loopback sockets for interprocess communications (IPC). This situation is common for applications that are migrating from early scale-out designs to larger systems (and core-count). The first application change is to choose a lighter weight form of IPC for in-system communications.

Excessive locking or poor lock granularity can also result in high kernel usage (in the kernel's `spin_lock`, `futex`, and scheduler components) when applications move to larger system configurations. This situation might require adjusting the application lock strategy and possibly the type of lock mechanism that is used as well:

- ▶ POSIX `pthread_mutex` and `pthread_rwlock` locks are complex and heavy, and POSIX semaphores are simpler and lighter.
- ▶ Use `trylock` forms to spin in user mode for a limited time when appropriate. Use this technique when there is normally a finite lock hold time and limited contention for the resource. This situation avoids context switch and scheduler impact in the kernel.
- ▶ Reserve POSIX `pthread_spinlock` and `sched_yield` for applications that have exclusive use of the system and with carefully designed thread affinity (assigning specific threads to specific cores).
- ▶ The compiler provides inline functions (`__sync_fetch_and_add`, `__sync_fetch_and_or`, and so on) that are better suited for simple atomic updates than POSIX lock and unlock. Use thread local storage, where appropriate, to avoid locking for thread safe code.

Using the IBM SDK for PowerLinux Trace Analyzer

The IBM SDK for PowerLinux provides tools, including the SystemTap and pthread monitor, for tracking I/O and lock usage of a running application. The higher level Trace Analyzer tools can target a specific application for combined SystemTap syscall trace and Lock Trace. The resulting trace information is correlated for time strip display and analysis within the tool.

High library usage

If libraries are consuming significant cycles, then you must determine if:

- ▶ Those libraries are part of your application, provided by a third party, or the Linux distribution
- ▶ There are alternative libraries that are better optimized
- ▶ You can recompile those libraries at a higher optimization

Libraries that are part of your application require the same level of empirical analysis as the rest of your application (by using source profiling and the Source Code Advisor). Libraries that are used by but not part of your application implies a number of options and strategies:

- ▶ Most open source packages in the Linux environment are compiled with optimization level `-O2` and tend to avoid additional (higher level GCC) compiler options. This configuration might be sufficient for a CISC processor with limited register resources, but not sufficient for a RISC based register-rich processor, such as POWER7.

- ▶ A RISC-based, superscalar, out-of-order execution processor chip such as POWER7 requires more aggressive inlining and loop-unrolling to capitalize on the larger register set and superscalar design point. Also, automatic vectorization is not enabled at this lower (-O2) optimization level, and so the vector registers and ISA feature go unused.
- ▶ In GCC, you must specify the -O3 optimization level and inform the compiler that you are running on a newer processor chip with the Vector ISA extensions. In fact, with GCC, you need both -O3 and -mcpu=power7 for the compiler to generate code that capitalizes on the new VSX feature of POWER7.

One source of optimized libraries is the IBM Advance Toolchain for PowerLinux. The Advance Toolchain provides alternative runtime libraries for all the common POSIX C language, Math, and pthread libraries that are highly optimized (-O3 and -mcpu=) for multiple Power platforms (including POWER7). The Advance Toolchain run time RPM provides multiple CPU tuned library instances and automatically selects the specific library version that is optimized for the specific POWER5, POWER6, or POWER7 machine.

If there are specific open source or third-party libraries that are dominating the execution profile of your application, you must ask the distribution or library product owner to provide a build using higher optimization. Alternatively, for open source library packages, you can build your own optimized binary version of those packages.

Deeper empirical analysis

If simple recompilation with higher optimization options or even a more capable compiler does not provide acceptable performance, then deeper analysis is required. The IBM SDK for PowerLinux integrates the following analysis tools:

- ▶ Migration Assistant analysis, non-performing codes, and data types
- ▶ Application-specific hotspot profiling
- ▶ Source Code Advisor (SCA) analysis for non-performing code idioms and induced execution hazards

The Migration Assistant analyzes the source code directly and does not require a running binary application for analysis. Profiling and the SCA do require compiled application binary files and an application-specific benchmark or repeatable workload for analysis.

The Migration Assistant

For applications that originate on another platform, the Migration Assistant (MA) can identify non-portable code that must be addressed for a successful port to Power Systems. The MA uses the Eclipse infrastructure to analyze:

- ▶ Data endian dependent unions and structures
- ▶ Casts with potential endian issues
- ▶ Non-portable data types
- ▶ Non-portable inline assembler code
- ▶ Non-portable or arch dependent compiler built-ins
- ▶ Proprietary or architectural-specific APIs

Program usage of non-portable data types and an inline assembler can cause poor performance on the POWER processor, which always must be investigated and addressed.

For example, the long double data type is supported for both Intel x86 and Power, but has a different size, data range, and implementation. The x86 80-bit Floating Point format is implemented in hardware and is usually faster than the AIX long double, which is implemented as an algorithm using two, 64-bit doubles. Neither one is fully IEEE-compliant, and both must be avoided in cross-platform application codes and libraries.

Another example is small Intel specific optimization using inline x86 assembler and conditionally providing a generic C implementation for other platforms. In most cases, GCC provides an equivalent built-in function that generates the optimal code for each platform. Replacing inline assembler with GCC built-in functions makes the application more portable and provides equivalent or better performance on all platforms.

To use the MA tool, complete the following steps:

1. Import your project into the SDK.
2. Select Project **properties**.
3. Check the **Linux/x86 to PowerLinux application Migration** check box under C/C++ General/Code Analysis.

Hotspot profiling

IBM SDK for PowerLinux integrates the Linux **Oprofile** hardware event profiling with the application source code view. This configuration is a convenient way to do hotspot analysis. The integrated Linux Tools profiler focuses on an application that is selected from the current SDK project.

After you run the application, the SDK opens an **Oprofile** tab in console window. This window shows a nested set of *twisties*, starting with the *event* (cycles by default), then *program/library*, *function*, and *source line* (within function). The developer drills-down by opening the twisties in the profile window, opening the next level of detail. Items are ordered by profile frequency with highest frequency first. Clicking the function or line number entries in the profile window causes the source view to *jump* to the corresponding source file or line number.

This process is a convenient way to do hotspot analysis, focusing only on the top three to five items at each level in the profile. Examine the source code for algorithmic problems, excess conversions, unneeded debug code, and so on, and make the appropriate source code changes.

With your application code (or subset) imported in to the SDK, it is easy to edit, compile, and profile code changes and verify improvements. As the developer makes code improvements, the hotspots in the profile change. Repeat this process until performance is satisfactory or all the profile entries at the function level are in the low single digits.

To use the integrated profiler, right-click the project and select **Profile As** → **Profile with Oprofile**. If you project contains multiple applications or the application needs setup or inputs to run the specific workload, then create Profile Configurations as needed.

Detailed analysis with the Source Code Advisor

Hotspot analysis might not find all of the latent performance problems, especially coding style and some machine-specific hazards. These problems tend to be diffused across the application, and do not show up in hotspot analysis. Common examples of machine hazards include address translation, cache misses, and branch miss-predictions.

Complex C++ applications or C programs that use object-based techniques might see performance issues that are related to using many small functions of indirect calls. Unless the compiler or optimizer can see the whole program or library, it cannot prove that it is safe to optimize these cases. However, it is possible for the developer to manually optimize at the source level, as the developer knows the original intent or actual usage in context.

The SCA can find and recommend solutions for many of these coding style and machine hazards. The process generates a journal that associates performance problems (including hazards) with specific source file and line numbers.

The SCA window has a drill-down hierarchy similar to the profile window described in “Hotspot profiling” on page 175. The SCA window is organized as a list of problem categories, and then nested twisties, for affected functions and source line numbers within functions. Functions and lines are ordered by the percent of overall contribution to execution time. Associated with each problem is a plain language description and suggested solution that describes a source change or compiler or linker options that are expected to resolve the problem. Clicking the line number item *jumps* the source display to the associated source file and line number for editing.

SCA uses the Feedback Directed Program Restructuring (FDPR) tool to instrument your application (or library) for code and data flow trace while you run a workload. The resulting FDPR journal is used to drive the SCA analysis. Running FDPR and retrieving the journal is can be automated by clicking **Profile as** → **Profile with Source Code Advisor**.

Java (either AIX or Linux)

Focused empirical analysis of Java applications involves gathering specific types of performance information, making and assessing changes, and repeating the process. The specific areas to consider, the types of performance information to gather, and the tools to use, are described in this section.

32-bit or 64-bit JDK

All other things being equal, a 64-bit JDK using **-Xcompressedrefs** generally has about 5% lower performance than does a 32-bit JDK. Without the **-Xcompressedrefs** option, a 64-bit JDK might have 10% or more reduced performance, which is compared to a 32-bit JDK. Give careful consideration to the choice of a 32-bit or 64-bit JVM. It is *not* a good choice to take an application that suffers from excessive object allocation rates and switch to a 64-bit JVM simply to allow a larger heap size. The references in the related tools and analysis techniques information can be used to diagnose object allocation issues in an application.

For more information about this topic, see:

- ▶ “Verbose GC Log” on page 177
- ▶ 7.2, “32-bit versus 64-bit Java” on page 126.

Java heap size, and garbage collection policies and parameters

The performance of Java applications is often influenced by the heap size, GC policy, and GC parameters. Try different combinations, which are guided by appropriate data gathering and analysis. Various tools and diagnostic options are available that can provide detailed information about the state of the JVM. The information that is provided can be used to guide tuning decisions to maximize performance for an application or workload.

Verbose GC Log

The verbose GC log is a key tool to understanding the memory characteristics of a particular workload. The information that is provided in the log can be used to guide tuning decisions to minimize GC impact and improve overall performance. Logging can be activated with the `-verbose:gc` option and is directed to the command terminal. Logging can be redirected to a file with the `-Xverbosegclog:<file>` option.

Verbose logs capture many types of GC events, such as regular GC cycles, allocation failures, heap expansion and contraction, events related to concurrent marking, and scavenger collections. Verbose logs also show the approximate length of time many events take, the number of bytes processed (if applicable), and other relevant metrics. Information relevant to many of the tuning issues for GC can be obtained from the log, such as appropriate GC policies, optimal constant heap size, optimal min and max free space factors, and growth and shrink sizes. For a detailed description of verbose log output, consult the material on this subject in the *Java Diagnostics Guide*, available at:

http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/tools/gcpd_verbosegc.html

Garbage collection and memory visualizer

For large, long-running workloads, verbose logs can quickly grow in size, making them difficult to work with and to analyze an application's behavior over time. The Garbage Collection and Memory Visualizer is a tool that can parse verbose GC logs and present them in a visual manner using graphs and other diagrams, allowing trends and totals to be easily and quickly recognized. The graphs can be used to determine the minimum and maximum heap usage, growth and shrink rates over time, and identify oscillating behaviors. This information can be especially helpful when you choose optimal GC parameters. The GC and Memory Visualizer can also compare multiple logs side by side, which can aid in testing various options in isolation and determining their effects.

For more information about the GC and Memory Visualizer, see *Java diagnostics, IBM style, Part 2: Garbage collection with the IBM Monitoring and Diagnostic Tools for Java – Garbage Collection and Memory Visualizer*, available at:

<http://www.ibm.com/developerworks/java/library/j-ibmtools2>

Java Health Center

The Java Health Center is the successor to both the GC and Memory Visualizer and the Java Lock Monitor. It is an all-in-one tool that provides information about GC activity, memory usage, and lock contention. The Health Center also functions as a profiler, providing sample-based statistics on method execution. The Health Center functions as an agent of the JVM being monitored and can provide information throughout the life of a running application.

For more information about the Java Health Center, see *Java diagnostics, IBM style, Part 5: Optimizing your application with the Health Center*, available at:

<https://www.ibm.com/developerworks/java/library/j-ibmtools5>

For more information, see 7.4, “Java garbage collection tuning” on page 130.

Hot method or routine analysis

A CPU profile shows a breakdown of the time that is spent in Java methods and JNI or system routines. Investigate any hot methods or routines to determine if the concentration of execution time in them is warranted or whether there is poor coding or other issues.

Some tools and techniques for this analysis include:

- ▶ AIX **tprof** profiling. For more information, see *tprof Command*, available at:
<http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm>
- ▶ Linux **Oprofile** profiling. For more information, see the following resources:
 - *Taking advantage of oprofile*, available at:
<https://www.ibm.com/developerworks/wikis/display/LinuxP/Taking+advantage+of+oprofile>
 - *Oprofile with Java Support*, available at:
<https://www.ibm.com/developerworks/wikis/display/LinuxP/Java+Performance+on+POWER7#JavaPerformanceonPOWER7-4.50profilewithJavaSupport>
 - *OProfile manual*, available at:
<http://oprofile.sourceforge.net/doc/index.html>

General information about running the profiler and interpreting the results are contained in the sections on profiling in “AIX” on page 162 and “Linux” on page 171. For Java profiling, additional Java options are required to be able to profile the machine code that is generated for methods by the JIT compiler:

- ▶ AIX 32-bit: **-agentlib:jpa=instructions=1**
- ▶ AIX 64-bit: **-agentlib:jpa64=instructions=1**
- ▶ Linux Oprofile: **-agentlib:jvmti_oprofile**

The entire execution of a Java program can be profiled, for example on AIX by running the following command:

```
tprof -ujeskl -A -I -E -x java ...
```

However, it is more common to profile Java after a warm-up period so that JIT compilation activity has generally completed. To profile after a warm-up, start Java and wait an appropriate interval until steady-state performance is reached, which is anywhere from a few seconds to a few minutes for large applications. Then, invoke the profiler, for example, on AIX, by running the following command:

```
tprof -ujeskl -A -I -E -x sleep 60
```

On Linux, **Oprofile** can be used in a similar fashion; for more information, see “Java profiling example”, and follow the appropriate documentation in the resources included in this section.

Java profiling example

Example B-11 contains a sample Java program that is profiled on AIX and Linux. This program does some meaningless work and is purposely poorly written to illustrate lock contention and GC impact in the profile. The program creates three threads but serializes their execution by having them attempt to lock the same object. One thread at a time acquires the lock, forcing the other two threads to wait until they can get the lock and run the code that is protected by the synchronized statement in the `doWork` method. While they wait to acquire the lock, the threads initially use *spin locking*, repeatedly checking if the lock is free. After a suitable amount of spinning, the threads block rather than continuing to use CPU resources.

Example B-11 Sample Java program

```
public class ProfileTest extends Thread {  
  
    static Object o; /* used for locking to serialize threads */  
    static Double A[], B[], C[];
```

```

static int Num=1000;

public static void main(String[] args) {
    o = new Object();
    new ProfileTest().start(); /* start 3 threads */
    new ProfileTest().start(); /* each thread executes the "run" method */
    new ProfileTest().start();
}

public void run() {
    double sum = 0.0;
    for (int i = 0; i < 50000; i++) {
        sum += doWork(); /* repeatedly do some work */
    }
    System.out.println("sum: "+sum); /* use the results of the work */
}

public double doWork() {
    double d;
    synchronized (o) { /* serialize the threads to create lock contention */
        A = new Double [Num];
        B = new Double [Num];
        C = new Double [Num];
        initialize();
        calculate();
        d = C[0].doubleValue();
    }
    return(d); /* use the calculated values */
}

public static void initialize() {
    /* Initialize A and B. */
    for (int i = 0; i < Num; i++) {
        A[i] = new Double(Math.random()); /* use new to create objects */
        B[i] = new Double(Math.random()); /* to force garbage collection */
    }
}

public static void calculate() {
    for (int i = 0; i < Num; i++) {
        C[i] = new Double(A[i].doubleValue() * B[i].doubleValue());
    }
}
}

```

The program also uses the Double class, creating many short-lived objects by using **new**. By running the program with a small Java heap, GC frequently is required to free the Java heap space that is taken by the Double objects that are no longer in use.

Example B-12 shows how this program was run and profiled on AIX. 64-bit Java was used with the options `-Xms10m` and `-Xmx10m` to specify the size of the Java heap. The profile that is generated appears in the `java.prof` file.

Example B-12 Results of running tprof on AIX

```
# tprof -ujeskl -A -I -E -x java -Xms10m -Xmx10m -agentlib:jpa64=instructions=1 ProfileTest

Starting Command java -Xms10m -Xmx10m -agentlib:jpa64=instructions=1 ProfileTest

sum: 12518.481782746869
sum: 12507.63528674597
sum: 12526.320955364286
stopping trace collection.
Sun Oct 30 15:04:21 2011
System: AIX 6.1 Node: e19-90-28 Machine: 00F603F74C00
Generating java.trc
Generating java.syms

Generating java.prof
```

Example B-13 and Example B-14 on page 181 contain excerpts from the `java.prof` file that is created on AIX. The notable elements of the profile are:

- ▶ **Lock contention impact:** The impact of spin locking is shown in Example B-13 as ticks in the `libj9jit24.so` helper routine `jitMonitorEntry`, in the AIX pthreads library `libpthreads.a`, and in the AIX kernel routine `_check_lock`. This Java program clearly has excessive lock contention with `jitMonitorEntry` consuming 26.66% of the ticks in the profile. `jitMonitorEntry` and other routines, such as `jitMethodMonitorEntry`, indicate spin locking at the Java language level, and impact in the pthreads library or `_check_lock` is locking at the system level, which might or might not be associated with Java locks. For example, `libpthreads.a` and `_check_lock` are active for lock contention that is related to `malloc` on AIX.

Example B-13 AIX profile excerpt showing kernel and shared library ticks

```
Total Ticks For All Processes (KERNEL) = 690

Subroutine          Ticks   %   Source          Address  Bytes
=====
._check_lock        240    5.71 low.s           3420     40

Shared Object
=====
libj9jit24.so       1157   27.51 900000003e81240 5c8878
libj9gc24.so        510   12.13 900000004534200 91d66
/usr/lib/libpthreads.a[shr_xpg5_64.o] 175   4.16 900000000b83200 30aa0

Profile: libj9jit24.so

Total Ticks For All Processes (libj9jit24.so) = 1157

Subroutine          Ticks   %   Source          Address  Bytes
```

```

=====
.jitMonitorEntry          1121  26.66  nathe1p.s          549fc0  cc0
=====

```

- ▶ **Garbage Collection impact:** The impact of initializing new objects and of GC is shown in Example B-13 on page 180 as the 12.13% of ticks in the libj9gc24.so shared object. This high GC impact is related to the excessive creation of Double objects in the sample program.
- ▶ **Java method execution:** In Example B-14, the profile shows the time that is spent in the ProfileTest class, which is broken down by method. Some methods appear more than one time in the breakdown because they are compiled multiple times at increasing optimization levels by the JIT compiler. Most of the ticks appear in the final highly optimized version of the **doWork()D** method, into which the **initialize()V** and **calculate()V** methods are inlined by the JIT compiler.

Example B-14 AIX profile excerpt showing Java classes and methods

Total Ticks For All Processes (JAVA) = 1450

Class	Ticks	%
ProfileTest	1401	33.32
java/util/Random	38	0.90
java/lang/Float	5	0.12
java/lang/Double	3	0.07
java/lang/Math	3	0.07

Profile: ProfileTest

Total Ticks For All Processes (ProfileTest) = 1401

Method	Ticks	%	Source	Address	Bytes
doWork()D	1385	32.94	ProfileTest.java	1107283bc	b54
doWork()D	6	0.14	ProfileTest.java	110725148	464
doWork()D	4	0.10	ProfileTest.java	110726e3c	156c
initialize()V	3	0.07	ProfileTest.java	1107262dc	b4c
calculate()V	2	0.05	ProfileTest.java	110724400	144
initialize()V	1	0.02	ProfileTest.java	1107255c4	d04

Example B-15 contains a shell program to collect a profile on Linux using **Oprofile**. The resulting profile might be similar to the previous example profile on AIX, indicating substantial time in spin locking and in GC. Depending on some specifics of the Linux system, however, the locking impact can appear in routines in the `libj9thr24.so` shared object, as compared to the AIX spin locking seen in `libj9jit24.so`. In some cases, an environment variable setting might be necessary to indicate the location of the JVMTI library that is needed for running **Oprofile** with Java:

- ▶ Linux 32-bit: `LD_LIBRARY_PATH=/usr/lib/oprofile`
- ▶ Linux 64-bit: `LD_LIBRARY_PATH=/usr/lib64/oprofile`

Example B-15 Linux shell to collect a profile using Oprofile

```
#!/bin/ksh

# Use --no-vmlinux if we either have a compressed kernel or do not care about the kernel symbols.
# Otherwise, use "opcontrol --vmlinux=/boot/vmlinux", for example.
opcontrol --no-vmlinux

# Stop data collection and remove daemon. Make sure we start from scratch.
opcontrol --shutdown

# Load the Oprofile module if required and makes the Oprofile driver interface available.
opcontrol --init

# Clear out data from current session.
# opcontrol --reset

# Select the performance counter that counts non-idle cycles and generates a sample after 500,000
# such events.
opcontrol -e PM_RUN_CYC_GRP1:500000

# Start the daemon for data collection.
opcontrol --start

# Run the Java program. "-agentlib:jvmti_oprofile" allows Oprofile to resolve the jitted methods.
java -Xms10m -Xmx10m -agentlib:jvmti_oprofile ProfileTest

# Stop data collection.
opcontrol --stop

# Flush the collected profiling data.
opcontrol --dump

# Generate a summary report at the module level.
opreport > ProfileTest_summary.log

# Generate a long report at the function level.
opreport -l > ProfileTest_long.log
```

Locking analysis

Locking bottlenecks are fairly common in Java applications. Collect locking information to identify any bottlenecks, and then take appropriate steps to eliminate the problems. A common case is when older `java/util` classes, such as `Hashtable`, do not scale well and cause a locking bottleneck. An easy solution is to use `java/util/concurrent` classes instead, such as `ConcurrentHashMap`.

Locking can be at the Java code level or at the system level. Java Lock Monitor is an easy to use tool that identifies locking bottlenecks at the Java language level or in internal JVM locking. A profile that is slowing a significant fraction of time in kernel locking routines indicates that system level locking that might be related to an underlying Java locking issue. Other AIX tools, such as **sp1at**, are helpful in diagnosing locking problems at the system level.

Always evaluate locking in the largest required scalability configuration (the largest number of cores).

Java Lock Monitor

The Java Lock Monitor is a valuable tool to deal with concurrency and synchronization in multi-threaded applications. The JLM can provide detailed information, such as how contested every monitor in the application is, how often a particular thread acquires a particular monitor, and how often a monitor is reacquired by a thread that already owns it. The locks that are surveyed by the JLM include both application locks and locks used internally by the JVM, such as GC locks. These statistics can be used to make decisions about GC policies, lock reservation, and so on, to make optimal usage of processing resources. For more information about the Java Lock Monitor, see *Java diagnostics, IBM style, Part 3: Diagnosing synchronization and locking problems with the Lock Analyzer for Java*, available at:

<http://www.ibm.com/developerworks/java/library/j-ibmtools3>

Also, see “Hot method or routine analysis” on page 177.

Thread state analysis

Multi-threaded Java applications, especially applications that are running on top of WebSphere Application Server, often have many threads that might be blocked or waiting on locks, database operations, or file system operations. A powerful analysis technique is to look at the state of the threads to diagnose performance issues.

Always evaluate thread state analysis in the largest required scalability configuration (the largest number of cores).

WAIT

WAIT is a lightweight tool to assess various performance issues that range from GC to lock contention to file system bottlenecks and database bottlenecks, to client delays and authentication server delays, and more, including traditional performance issues, such as identifying hot methods.

WAIT was originally developed for Java and Java Platform, Enterprise Edition workloads, but a beta version that works with C/C++ native code is also available. The WAIT diagnostic capabilities are not limited to traditional Java bottlenecks such as GC problems or hot methods. WAIT employs an expert rule system to look at how Java code communicates with the wider world to provide a high-level view of system and application bottlenecks.

WAIT is also agentless (relying on **javacores**, **ps**, **vmstat**, and similar information, all of which are subject to availability). For example, WAIT produces a report with whatever subset of data can be extracted on a machine. Getting **javacores**, **ps**, and **vmstat** data almost never requires a change to command lines, environment variables, and so on.

Output is viewed in a browser such as Firefox, Chrome, Safari, and Internet Explorer, and assuming one has a browser, no additional installation is needed to view the WAIT output.

Reports are interactive, and clicking different elements reveals more information. Manuals, animated demonstrations, and sample reports are also available on the WAIT website.

For more information about WAIT, go to:

<http://wait.researchlabs.ibm.com>

This site also has sample input files for WAIT, so users can try out the data analysis and visualization aspects without collecting any data.



POWER7 optimization and tuning with third-party applications

This appendix describes the optimization and tuning of the POWER7 processor-based server running third-party applications. It covers the following topics:

- ▶ Migrating Oracle to POWER7
- ▶ Migrating Sybase ASE to POWER7
- ▶ Migrating SAS to POWER7
- ▶ Migrating SAP BusinessObjects Business Intelligence platform with POWER7

Migrating Oracle to POWER7

This section provides you with key documents and links to assist with IBM Power Systems in an Oracle environment. This section includes information specific to POWER7 in an Oracle environment. IBM is not aware of any POWER7 specific Oracle issues at the time of this writing. Most issues that show up on POWER7 are the result of not following preferred practices that apply to all Power Systems generations.

This section also includes description about Oracle Non-Real Application Clusters (NON-RAC) and Real Application Clusters (RAC).

Table C-1 lists important documents that you should become familiar with.

Table C-1 Power Systems and Oracle information.

Title	Resource	Description
Must read first - Oracle and IBM references		
Oracle Readme/Release Notes Main Page: The Readme/Release Notes are dynamic documents updated online with new information.	http://www.oracle.com/tech/network/indexes/documentation/index.html?ssSourceSiteId=ocomen	Main Oracle page from which you can select an Oracle product for specific information about a platform (installation information, AIX patches, tuning, implementation suggestions, and so on).
<i>Oracle 11gR2 Readme/Release Notes.</i>	http://docs.oracle.com/cd/E11882_01/relnotes.112/e10853/toc.htm	Oracle Database Release Notes 11g Release 2 (11.2) for IBM AIX on Power Systems (64-bit) Part Number E23560-03.
<i>Oracle 10gR2 Readme/Release Notes.</i>	http://docs.oracle.com/cd/B19306_01/relnotes.102/b19074/toc.htm	Oracle Database Release Notes 10g Release 2 (10.2) for AIX Part Number B19074-15.
Minimum Software Versions and Patches Required to Support Oracle Products on IBM Power Systems.	https://support.oracle.com (registration required)	My Oracle Support (MOS) ID 282036.1.
<i>Oracle Architecture and Tuning on AIX V2.20</i>	http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100883	AIX tuning reference guide available on Techdocs.
<i>Oracle Real Application Clusters on IBM AIX – Best practices in memory tuning and configuring for system stability.</i>	http://www.oracle.com/tech/network/database/clusterware/overview/rac-aix-system-stability-131022.pdf	RAC tuning and configuration guide.
<i>Oracle Real Application Clusters (RAC) and Oracle Clusterware Interconnect Virtual Local Area Networks (VLANs) Deployment Considerations.</i>	http://www.oracle.com/tech/network/database/clusterware/overview/interconnect-vlan-06072012-1657506.pdf	RAC and VLAN deployment guide.

Managing Raw Disks in AIX to use with Oracle Automatic Storage Management (ASM).	https://support.oracle.com (registration required)	MOS ID 1445870.1.
<i>Oracle Database Cross Platform Migration to AIX.</i>	http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP101822	Cross-platform migration reference guide available on Techdocs.
Must read for Oracle 11g		
<i>Oracle Database Online Patching on AIX.</i>	http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102085	AIX fixes and Oracle patches that must be used with the Oracle Database Online patching feature.
<i>Oracle's USLA HEAP patches available on AIX.</i>	http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102066	Addresses that are increased per-process memory consumption.
<i>Oracle Database 11.2.0.3 available on AIX.</i>	http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102016	11.2.0.3 certification information.
<i>Oracle DB & RAC 11gR2 on IBM AIX: Tips and Considerations.</i>	http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP101176	11gR2 planning and implementation
<i>Oracle DB & RAC 10gR2 on IBM AIX: Tips and Considerations.</i>	http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP101089	10gR2 planning and implementation.
Additional IBM Information		
Review Recent ICC Flashes on Techdocs – the Technical Sales Library.	http://www.ibm.com/support/techdocs/atmastr.nsf/Web/Techdocs	Technical sales support database.
<i>Guide to Multiple Page Size Support on AIX 5L Version 5.3</i>	http://www.ibm.com/systems/resources/systems_p_os_aix_whitepapers_multiple_page.pdf	Useful information about large page configuration.
Large page size.	Review “Oracle large page usage” on page 189 for specifications	Larger page size often provides performance improvements with Oracle.
Existing environment statistics capture.	Consider creating a historical comparison baseline to aid in solving a perceived change in performance, for example, when a new system seems slower than the old one.	Determine what is most applicable to capture in your environment (that is, AWR or PerfPMR types of reports).
<i>Java Performance on POWER7 - Best Practice.</i>	http://www.ibm.com/systems/power/hardware/whitepapers/java_perf.html	Information about migrating Java applications from POWER5 or POWER6 to POWER7.

Oracle 11gR2 preferred practices for AIX V6.1 and AIX V7.1 on Power Systems

This section is a summary of preferred practices for stand-alone Oracle 11gR2 instances on AIX V6.1 and AIX 7.1 on POWER7 Systems. Except for references to symmetric multithreading, quad-threaded mode (SMT4 mode), all of the preferred practices apply to POWER6 as well. Although the primary focus is on stand-alone Oracle 11gR2 with file system-based storage, stand-alone ASM is also addressed. These preferred practices apply to Oracle 10.2.0.4 as well, and where they differ from those practices for Oracle 11gR2, patches specific to Oracle 10.2.0.4 are noted.

Detailed Oracle/AIX preferred practices documents are available on IBM Techdocs at <http://www.ibm.com/support/techdocs/atsmastr.nsf/Web/Techdocs>, including *Oracle Architecture and Tuning on AIX v2.20*, found at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100883>

The following sections describe memory, CPU, I/O, network, and miscellaneous settings. In addition, these sections list the AIX levels that are required for Oracle 11gR2, the Oracle patches for 11gR2 on AIX V6.1 and AIX V7.1, and the Oracle patches for 10.2.0.4 and 11gR2.

Memory

The specifications for kernel settings and Oracle large page usage are described in the following sections.

Kernel settings

Kernel settings are listed in Table C-2. These values are commonly suggested ones.

Table C-2 Kernel settings for Oracle

Parameter	Proposed value	AIX V6.1 default	AIX V6.1 restricted	AIX V7.1 default	AIX V7.1 restricted
minperm%			No		No
maxperm%	90	90	Yes	90	Yes
maxclient%	90	90	Yes	90	Yes
strict_maxclient			Yes		Yes
strict_maxperm			Yes		Yes
lru_file_repage			Yes	N/A	N/A
lru_poll_interval	10	10	Yes	10	Yes
minfree	960	960	No	960	No
maxfree	1088	1088	No	1088	No
page_steal_method			Yes		Yes
memory_affinity			Yes		Yes
v_pinshm			No		No
lgpg_regions			No		No
lgpg_size			No		No

maxpin%	80	80	No	90	No
esid_allocator	1 ^a		No		No

a. The default value of 1 for **esid_allocator** enables terabyte segment aliasing, reducing addressing lookasides. This value can be set to 1 in AIX V6.1 and is suggested for Oracle.

In general, you should use AIX V7.1 defaults for Oracle.

Two noticeable changes from AIX V6.1 to AIX V7.1 are:

- ▶ The elimination of the **lru_file_repage** tunable
- ▶ The default value of 90 for **maxpin%**, which is increased from 80% in AIX V6.1

Oracle large page usage

Specifications for Oracle large page usage are:

- ▶ AIX V6.1 and AIX V7.1 support three or four page sizes, depending on the hardware: 4 KB (default), 64 KB, 16 MB, and 16 GB. All four page sizes are supported by Power Systems from POWER5+ firmware level 240-202 onward.

Page sizes 64 KB and 16 MB benefit Oracle performance by reducing kernel lookaside processing to resolve virtual to physical addresses. Oracle 11g uses 64 KB pages for data spaces by default.

- ▶ LOCK_SGA = FALSE:
 - This setting is the default, which means that the SGA is not pinned in memory.
 - In general, you should not pin SGA.
 - Automatic Memory Management (AMM) uses 64 KB pages for SGA if memory is available.
 - *This value is the preferred one*, as it has been found that 64 KB pages yield nearly the same performance benefit as 16 MB pages and require no special management.
 - Oracle 10.2.0.4 with Oracle patch 7226548 also uses 64 KB pages for the SGA.
- ▶ LOCK_SGA = TRUE:
 - The AIX parameters to enable pinned memory and 16 MB large pages are:
 - **vmo -p -o v_pinshm=1** (allows pinned memory and requires reboot)
 - **vmo -p -o lpgg_size=16777216 -o lpgg_regions=number_of_large_pages**, where **number_of_large_pages=INT[SGA size -1]/16MB]+1**
 - AMM can be used with pinned 16 MB pages, provided the formula for calculating the number of large pages is modified so that the number of large pages=memory_max_target+1.
 - The capabilities that are required to allow Oracle to use 16 MB large pages (implement as root) can be set by running the following command:


```
#chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE oracle
```

For more information, go to http://www.ibm.com/developerworks/forums/servlet/JiveServlet/download/747-313057-14417535-361419/Oracle_Tuning_on_AIX.pdf and click metalink note: 372157.1
 - With Oracle 10.2.0.4, patch 7226548 is also required to use 16 MB pinned pages.
- ▶ Using 64 KB pages for data, text, and stack regions (applies to Oracle 10.2.0.4 as well):
 - 64 KB page size for data, text, and stack regions is useful in environments with a large (for example, 64 KB+) SGA and many online transaction processing (OLTP) users. For smaller Oracle instances, 4 KB is sufficient for data, text, and stack.

- 64 KB page use for data, text, and stack is implemented separately from 64 KB pages for the SGA, and is done through an environment variable that is exported on behalf of the Oracle user:

```
$ export LDR_CNTRL=DATASIZE=64K@TEXTSIZE=64K@STACKSIZE=64K oracle
* LDR_CNTRL=DATASIZE=64K@TEXTSIZE=64K@STACKSIZE=64K
export LDR_CNTRL
```

CPU

CPU specifications are as follows:

- ▶ SMT mode: POWER7 supports SMT4 mode, which is the AIX default. AIX and Oracle performance support encourages starting with the default.
- ▶ Virtual processor folding: This is a feature of Power Systems in which unused virtual processors are taken offline until demand requires that they be activated. The default is to allow virtual processor folding; do not alter this setting without consulting AIX support.
- ▶ Specific to POWER7 SMT4 mode: Certain Oracle 11G parameters, including **DB_WRITER_PROCESSES** and **PARALLEL_MAX_SERVERS**, are partly derived from **CPU_COUNT**, and **CPU_COUNT** is equal by default to the number of logical CPUs. **CPU_COUNT** automatically adjusts to changes in virtual processor count and to SMT mode, up to three times the value on startup.
- ▶ When you migrate from single-threaded platforms to Power Systems, or from POWER5 or POWER6 to POWER7 with SMT4 mode, the value of **CPU_COUNT** also increases, affecting **DB_WRITER_PROCESSES**, **PARALLEL_MAX_SERVERS**, and other dependent parameters. Queries that are sensitive to a degree of parallelism might change behavior because of the migration to POWER7. Reviewing the **PARALLEL_MAX_SERVERS** parameter after migration, but set **DB_WRITER_PROCESSES** to the defaults.

I/O

I/O specifications are as follows:

- ▶ If ASM is not used, use max interpolicy striping (also known as *pp spreading* or *poor man's striping*) when logical volumes are created. To get the most benefit from spreading physical partitions across the LUNs, use a small physical partition size, for example, 32 MB or 64 MB.
- ▶ Async I/O is used even with Concurrent I/O (CIO):
 - With AIXV 6.1 and AIX V7.1, start with the asynchronous I/O defaults. With AIX V6.1, there is a new implementation of AIO. AIO kernel extensions are loaded at system boot (always loaded), AIO servers stay active if there are service requests, and the number of AIO servers is dynamically increased or reduced based on demand of the workload. The **aio_server_inactivity** parameter defines how many seconds of idle time before an AIO server exits. AIO tunables are now based on logical CPU count, and it is usually not necessary to tune **minservers** , **maxservers** , and **maxreqs** .
 - In AIX V6.1, there are two tunables for minservers and maxservers, **aio_minservers/aio_maxservers** for older threads, and **posix_aio_minservers/posix_aio_maxservers** for POSIX threads. Oracle uses older threads.
 - Increase **aio_maxservers** or **posix_aio_maxservers** Only by running **ioo -p -o** (the default is 30 per logical CPU) if you run **pstat -a | fgrep aio** or **ps -k | fgrep aio** shows that you are continually using **maxservers** .
 - Oracle parameter (**init.ora**) should be set to **disk_async_io = TRUE** .

- ▶ Buffered file I/O on JFS2:
 - The default is `filesystemio_options=ASYNC`.
 - In this case, all data spaces, redo log file systems, and control file systems are using the kernel buffers rather than writing directly to disk.
 - In this case, it does not matter whether redo log file systems and control file systems are 512 byte or 4 KB block size file systems.
 - The best performance for Oracle on AIX or Power Systems is, however, usually achieved by using CIO (though there are exceptions).
- ▶ Concurrent I/O (CIO) on JFS2:
 - Set the Oracle parameter `filesystemio_options=SETALL` or mount the file systems (other than dump devices) with the `CIO` option. It is not necessary to use both `SETALL` and mount file systems with the `CIO` option, although no harm is done either way. Metalink note: 272520.1 indicates that mounting with CIO is needed at the time of the writing of this guide.
 - If you use CIO with `SETALL`, `CIO mount`, or both, you *must* create separate file systems for redo logs and control files (or a single file system for both), with an `agblksize` of 512 rather than the default 4 KB.
 - The `ioo` parameter `fs_fastpath` accelerates CIO. It is enabled by default in AIX V6.1 and AIX V7.1.
- ▶ IBM mount advice for database files:
 - Data files: Use the CIO parameter `filesystemio_options=SETALL`, with a default of `agblksize` (4 KB). Use `mount` with no options.
 - Redo logs: Create the logs with `agblksize` of 512 and `mount` with no options. With `SETALL`, use direct I/O for Redo logs.
 - Control files: Create the files with `agblksize` of 512 and `mount` with no options. With `SETALL`, use direct I/O for Redo logs.
 - Archive logs: Run `mount -o rbrw`. Do not use CIO; use the `jfs2 rbrw` option.
 - Dumps: Run `mount -o rbrw`.
 - The `mount` option `noatime`, suggested for Oracle 10g, is no longer required.
- ▶ IOO tunables `j2_nBufferPerPagerDevice` and `j2_dynamicBufferPreallocation`:
 - Do not change these tunables unless there is a high delta in `vmstat -v` external pager file system I/Os that are blocked with no `fsbuf`. If this value is high, first increase `j2_dynamicBufferPreallocation` from 16 (16 KB slabs) to 32 and monitor the system. If increasing this tunable does not help, then consider raising the value of `j2nBufferPerPagerDevice`, which is the starting value for dynamic buffer allocation.
 - For information about these parameters, see the help pages available at http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmd_s3/ioo.htm. Do not change AIX V6.1 or AIX V7.1 restricted tunables unless directed to do so by IBM AIX support. In AIX V6.1, `j2_nBufferPerPagerDevice` is a restricted tunable, and `j2_dynamicBufferPreallocation` is not.
- ▶ ASM considerations for stand-alone Oracle 11gR2:
 - For identifying, renaming, and securing ASM raw devices, see *Managing Raw Disks in AIX to use with Oracle Automatic Storage Management (ASM)* in Table C-1 on page 186.
 - ASM uses asynchronous I/O by default, so `filesystemio_options=ASYNC` (the default) is appropriate.

- In the stand-alone usage of ASM, unlike RAC, hdisks and hdiskpower devices do not need to have Small Computer System Interface (SCSI) reservation disabled.
- The following initialization parameters must be adjusted for ASM:
 - Add 16 to the value of processes.
 - Add an additional 600 KB to the value of large pool size.
 - Add to shared pool size the aggregate of the values that are returned by these queries:
 - `SELECT SUM(bytes)/(1024*1024*1024) FROM V$DATAFILE;`
 - `SELECT SUM(bytes)/(1024*1024*1024) FROM V$LOGFILE a, V$LOG b WHERE a.group#=b.group#;`
 - `SELECT SUM(bytes)/(1024*1024*1024) FROM V$TEMPFILE WHERE status='ONLINE';`
- For disk groups using external redundancy, every 100 GB of space needs 1 MB of extra shared pool, plus 2 MB.
- For disk groups using normal redundancy, every 50 GB of space needs 1 MB of extra shared pool, plus 4 MB.
- For disk groups using high redundancy, every 33 GB of space needs 1 MB of extra shared pool, plus 6 MB.

For more information, go to:

http://docs.oracle.com/cd/E18283_01/server.112/e16102/asminst.htm#CHDBBIBF

Network

This section outlines the minimum values that are applicable to network configurations.

Kernel configurations

The following values can be considered as starting points for Oracle:

- ▶ Set `sb_max` \geq 1MB (1048576), which must be greater than the maximum `tpc` or `udp send` or `recvspace` (if you are using RAC and an especially large `udp_recvspace`, you might need to increase `sb_max`).
- ▶ Set `tcp_sendspace` = 262144.
- ▶ Set `tcp_recvspace` = 262144.
- ▶ Set `udp_sendspace` = `db_block_size` * `db_file_multiblock_read_count`.
- ▶ Set `udp_recvspace` = 10 * (`udp_sendspace`).
- ▶ Set `rfc1323` = 1.
- ▶ Ephemerals (non-defaults suggested for many connecting hosts or a high degree of parallel query; also to avoid install-time warnings) should be set as follows:
 - `tcp_ephemeral_low`=9000
 - `tcp_ephemeral_high`=65500
 - `udp_ephemeral_low`=9000
 - `udp_ephemeral_high`=65500

Jumbo frames are Ethernet frames larger than the standard MTU size of 1500 bytes. They can be up to 9000 bytes. They are used to reduce the number of frames to transmit a volume of network traffic, but they work only if enabled on every *hop* in the network infrastructure. Jumbo frames help reduce network and CPU processing impacts.

Miscellaneous

Miscellaneous specifications for Oracle are described in this section:

- ▶ `ulimits` (run `smitt chuser` or edit `/etc/security/limits` to create a stanza for Oracle) should be set to `-1` (unlimited) for all values except `core`.
- ▶ The maximum number of `PROCESSES` allowed per user (run `smitt chgsys` to set) should be set to `maxuproc >= 2048`. 16 KB is a commonly suggested value for Oracle.
- ▶ Environment variables should be set as follows:
 - `AIXTHREAD_SCOPE=S` (set in Oracle profile)
 - `LDR_CNTRL=DATAPSIZE=64K@TEXTPSIZE=64K@STACKPSIZE=64K` (set in the Oracle profile and the Oracle listener's environment)
 - There is an option to use `ldedit` to edit the Oracle binary files so that they use 64 KB pages directly. Run the following command:

```
# ldedit -btextpsize=64k -bdatapsize=64k -bstackpsize=64k
$ORACLE_HOME/bin/oracle
```

Using `ldedit`: Whenever a patch is applied or an Oracle relink is performed, the `ldedit` command must be run again.

- ▶ Disk and adapter resources
 - To set `hdisk`, run `lsattr -El hdisk<>`.
 - The queue depth might vary among 8, 16, 20, and 24, depending on the storage vendor. A queue depth of 2 on SAN devices usually indicates a driver mismatch, but is the default for Hitachi HDS on AIX; increase this value to 8 as a starting point. Queue wait and queue overflow that is detected by running `iostat -D1` might indicate a need to increase queue depth.
 - `max_transfer` might need to be adjusted upward, depending on the largest I/O requested by Oracle; a typical starting point for Oracle on AIX is `0x100000`.
 - To set FC Adapter, run `lsattr -El fcs<>` and `fcstat -e`.
 - `max_xfer_size` must be at least equal to `max_transfer` at the `hdisk` level.
 - `num_cmd_elems` might need to be increased if `fcstat -e` reports a persistent non-zero value for *No Command Resource Count*.
 - If `fcstat -e` reports a persistent, non-zero value for *No DMA Resource Count*, contact support.

Migrating Sybase ASE to POWER7

Sybase ASE is widely used. In the past two years, many migrations of Sybase ASE to POWER7 were completed or are in progress. Two published white papers provide comprehensive documentation for general preferred practices and for migration to POWER7:

- ▶ *Optimizing Sybase Adaptive Server Enterprise (ASE) for IBM AIX*, available on the IBM Techdocs website and on the Sybase website:
 - <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101429>
 - <http://www.sybase.com/detail?id=1096191>

This paper is a joint IBM and Sybase publication that was originally written in 2006 and has been updated twice since then, most recently in Spring, 2012. The authors of the current version are Peter Barnett (IBM), Mark Kusma (Sybase), and Dave Putz (Sybase).

This publication directly addresses three main areas in which POWER7 differs from POWER5 and POWER6: SMT4 mode, the virtual processor folding algorithm, and overall throughput.

Kernel tuning parameters are presented in detail, particularly regarding memory and I/O, and closely resemble preferred practices for Oracle.

The section *Processor Considerations* addresses the implications of SMT4 mode for allocation of Sybase ASE engines, the implications of the POWER7 virtual processor allocation algorithms on sizing ASE in a shared pool environment, and general preferred practices for aligning Sybase ASE resources with entitlement and virtual processor count.

The same section of this paper addresses the non-traditional technique of allocating more Sybase ASE engines than there are AIX virtual processors or cores. In particular, you should not exceed a ratio of two engines per virtual processor, even with SMT4 mode enabled.

Suggestions to reduce SMT mode to 2 or even ST mode for Sybase ASE and suggestions to turn off virtual processor folding are noted. Most of clients get the best performance from Sybase ASE on AIX/POWER7 with SMT4 mode and virtual processor folding enabled.

Appendix A of this paper presents an iterative tuning exercise with Sybase ASE 15.5, AIX V7.1, and POWER7. Results are presented for the TPC-C benchmark and other workloads, which indicate that the SMT4 mode provides superior or equal performance when compared to SMT2 and ST modes.

- *Migrating Sybase ASE to IBM Power Systems*, available at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102105>

The subtitle of this paper explains its purpose: It presents the case for POWER7 as the optimal platform for Sybase ASE, presents guidelines and preferred practices for migrations, and includes a field guide to migrations and proofs of concept.

Many sections of this paper are useful for systems and database managers in planning and carrying out migrations of Sybase ASE to POWER7.

Part I includes a description about Cross Platform Dump and Load (XPDL), the most widely used ASE migration technique, including preparation and post-migration cleanup steps. Other migration techniques are also described. The implications of upgrading ASE from Version 12.5 to Version 15.x, as part of a platform migration, are also described, and some preferred practices are offered. Specific scenarios and case studies are cited, including scale-ups and consolidations. Shared processor and virtualization concepts are reviewed in the context of Sybase ASE.

In Part II, the “Field guide to migrations and proofs of concept”, the first section on requirements gathering, presents a detailed list of configuration information and performance data to be collected in preparation for a migration to POWER7.

In Part III, commonly encountered problems in migrating Sybase ASE to POWER7 are identified and described in some detail.

A list of configuration and performance data to collect when you troubleshoot a performance issue is provided.

Several performance scenarios are described and compared: The case of spinlock contention is contrasted with a *healthy* CPU-bound workload. Likewise, the case of idling engines because of overallocation is contrasted with a *healthy* I/O-bound workload.

Implementing Sybase IQ to POWER7

Sybase IQ is a specialized data server that is optimized for *ad hoc* queries and reporting on data warehouses and data marts.

AIX kernel tunables for Sybase IQ are the same regarding virtual memory, I/O, network, and ulimits as for Sybase ASE and Oracle.

However, there are some special environment variables and scheduler tunables that should be considered, based on joint work between Sybase and AIX development teams. These items were originally developed for POWER6, but are also applicable to POWER7. Some of these are *firm suggestions* that we urge you to follow, and others are *soft suggestions*, as they are helpful with specific workloads, and can be used at your discretion.

Environment variables

The environment variables are:

▶ **SPINLOOPTIME=500** (*Firm suggestion*)

If a user thread cannot get a mutex, it spins for some time, hoping to acquire the lock afterward. The environment variable **SPINLOOPTIME** controls the spin time duration. After the period, the thread either goes to sleep or stays runnable (see **YIELDLOOPTIME**).

▶ **YIELDLOOPTIME=0** (*Firm suggestion*)

It is more efficient for a runnable thread to resume than to wake up from sleep and run. After the spin time is exhausted, the thread can go to sleep or give up CPU but stay runnable. The environment variable **YIELDLOOPTIME** defines the number of times a thread gives up the CPU before it sleeps.

▶ **AIXTHREAD_SCOPE=S** (Same as **AIXTHREAD_MNRATIO=1:1**) (*Firm suggestion*)

Setting **AIXTHREAD_SCOPE=S** means that user threads created with default attributes are placed into system-wide contention scope. If a user thread is created with system-wide contention scope, it is bound to a kernel thread, and it is scheduled by the kernel. The underlying kernel thread is not shared with any other user thread.

▶ **AIXTHREAD_MNRATIO=8:1** (the default if **AIXTHREAD_SCOPE** is defaulted to **P**) (*Soft suggestion*)

Setting **AIXTHREAD_SCOPE=S** means that user threads created with default attributes are placed into system-wide contention scope. Defaulting to **AIXTHREAD_SCOPE** or specifying a ratio greater than 1:1 for **AIXTHREAD_MNRATIO=M:N** means that kernel threads might be shared by M user threads. M>N might be more efficient for highly threaded processes.

▶ Sybase IQ start script **start_iq** ships with deprecated values for **AIXTHREAD_MNRATIO**, **SPINLOOPTIME**, and **YIELDLOOPTIME** (*Firm suggestion*)

Sybase IQ start scripts set both **AIXTHREAD_SCOPE=S** and **AIXTHREAD_MNRATIO=4:1**. The latter overrides the former. The former is considered a starting point for systematic testing. IQ start scripts also set **SPINLOOPTIME** and **YIELDLOOPTIME** to 40 without any explanation.

▶ **AIXTHREAD_MUTEX_FAST=ON** (*Firm suggestion*)

Setting the variable to **ON** forces threaded applications to use an optimized mutex locking mechanism, resulting in increased performance.

▶ **MALLOCOPTIONS=multiheap:8,considersize** (*Firm suggestion*)

Multiheap defaults to 32 heaps, but a smaller number is suggested. The number must equal the number of logical CPUs, or 32, whichever is less. Watson and Yorktown versions of **MALLOCOPTIONS** are problematic for Sybase IQ.

- ▶ **LDR_CNTRL=TEXTPSIZE=64K@STACKPSIZE=64K@DATAPSIZE=64K** (*Firm suggestion*)
Set in the *Sybase owner* profile, not in **start_asiq**, this environment variable enables the application to use 64 KB pages.
- ▶ **AIXTHREAD_PREALLOC=<size of IQ heap in bytes>** (*Firm suggestion*)
 - A mostly undocumented feature of AIX 5L V5.3.0.10+ and AIX V6.1.0.3+
 - The kernel pre-sets the heap pointer, reducing the number of system calls in malloc
 - Saves some time that is otherwise taken by threads that are waiting on the malloc mutex
 - Reported to benefit larger systems, 8-core or more
 - Probably most beneficial to IQ load phases

Special considerations for Sybase IQ with SMT4 mode

SMT4 mode provides special benefits for Sybase IQ, owing to its highly threaded architecture. The IQ configuration parameter, **-iqnumbercpus**, is determined by default by the number of logical processors that IQ detects. The **-iqnumbercpus** parameter affects the value of several downstream parameters, including **-iqgovern** and **-iqmt**. The configuration parameter **-iqmt** governs the number of threads created.

Some special considerations are:

- ▶ By default, **-iqmt** is set to $60 * (\min(-iqnumbercpus,4)) + 50 * (-iqnumbercpus - 4) + 2 * (\text{numConnections} + 2) + 1$
- ▶ On a 4-core, single threaded system with **-gm** (number of connections), it is set to 20, that is, $iqmt = 60 * (4) + 50 * (4-4) + 2 * (20+2) + 1 = 285$
- ▶ However, if SMT4 mode is on, **-iqnumbercpus** is set to 16 for a 4-core system, so **-iqmt** is 1605. Therefore, 1605 threads are allocated by IQ.
- ▶ Is IQ allocating too many threads in SMT4 mode? With POWER5 SMT2 mode, it is beneficial to set **-iqnumbercpus** to the number of cores or virtual processors. However, with POWER6 and POWER7, the best performance is generally obtained by leaving **-iqnumbercpus** at its default, even though IQ creates more threads.

Shared versus dedicated processors with Sybase IQ and POWER7

Sybase IQ is *CPU hungry* by design and scale both out and up with the addition of processors and workload. Sybase support generally favors dedicated cores over shared processor pool technology, and some enterprises report some negative experiences with IQ on shared processors, which might be because of extraneous factors.

The IBM team conducted systematic comparisons of dedicated and shared processor performance using a Sybase provided test workload to produce the *Stockmarket Benchmark*. This benchmark is similar to TPC-H in that it consists of simulated data that can be inflated. There are 18 complex queries that can be run in parallel or one at a time.

Our findings were that, on POWER7 and AIX V6.1, you can achieve equal or better performance for the 18 queries in parallel, if you allocate +1 virtual processors to the target dedicated cores and:

- ▶ Give the IQ LPAR slightly higher entitlement than its neighbors
or
- ▶ Give the IQ LPAR a higher *weight* setting than its neighbors
or
- ▶ Arrange the shared pool so that neighbor LPARs have lower peak demand

The reason for allocating +1 virtual processor in relation to the target dedicated CPU count is that the shared processor allocation algorithm is continually recalculating load and thus *averages down* slightly from the number of virtual processors. Thus, if you allocate four virtual processors and sustain a 100% load, the actual consumption is about 3.9 processors on average. Thus, to attain a peak allocation of the power of four processors, five virtual processors must be assigned, and so on.

You can achieve equal or better performance of each query that is run sequentially by using the configurations that are outlined in “Implementing Sybase IQ to POWER7” on page 195) and by turning off virtual processor *folding*. Turning off virtual processor folding is not standard practice, but it leads to improved performance in some cases.

Finally, our results suggest that IQ 15.2 can attain dedicated-level performance with a 1:4 entitlement to virtual processor ratio, providing that the other configuration suggestions are followed.

Migrating SAS to POWER7

This section provides a starting point for performance optimization from a system-wide perspective to create an enhanced environment for SAS 9 on IBM POWER processor-based servers that run AIX V6.1 and AIX V7.1. To verify that the version of SAS and the product suites you are running are supported at the version of AIX you are running, review the information found at:

<http://support.sas.com/resources/sysreq/hosts/unix/>

The preferred performance settings are as follows:

- ▶ Technology levels: Use the latest version of AIX V7 (currently Version 7.1) with the latest maintenance levels and fix packs, or AIX 6 V6.1. For more information, go to:
<http://www.ibm.com/support/fixcentral>
As of the publication of this guide, Version 7.1 TL01 with fix pack or Version 6.1 TL07 are considered to be the latest technology levels with fix packs.
- ▶ IBM HMC: Keep the HMC and microcode up to date.
 - Search for HMC update bulletins and microcode at:
<http://www14.software.ibm.com/webapp/set2/subscriptions/pqvcמיד>
 - VMM: Both the AIX V6.1 and AIX V7.1 default VMM parameters are mostly tuned for optimal SAS performance, except for one parameter. If needed, you can change the other values by using the AIX `vmo` command. Start with the following parameter values:
 - `nokilluid=10`.
 - `minfree=use greater of 960 or (128 * number of logical processors)`.

- `maxfree=(minfree + j2_maxPageReadAhead * number of logical processors)`.
- The following values are designated as restricted values in AIX V7, and you should not change them unless directed to do so by AIX support. These values are already in line with SAS recommendations.
 - `lru_file_repage` (Version 6.1 only).
 - `minperm%` (Version 6.1 only).
 - `maxperm%`.
 - `maxclient%`.
 - `strict_maxclient`.
 - `strict_maxperm`.
- I/O: Tune I/O at the file system layer by using the AIX `ioo` command to enable Enhanced Journaled File System (JFS2) to perform efficient caching. Start with the following parameter values:
 - `j2_dynamicBufferPreallocation=256`.
 - `j2_nBufferPerPagerDevice=2048`.
 - `j2_maxPageReadAhead=1024`.
 - `j2_minPageReadAhead=16`.
 - `j2_nPagesPerWriteBehindCluster=64`.

For higher workloads, increase the value of `j2_maxPageReadAhead` up to 2048.

- Network: Tune the network parameters by using the `no` command. If SAS applications (such as Scalable Performance Data Server (SPDS), Scalable Performance Data Engine (SPDE), or SAS/CONNECT) are being used, set the following parameter value:


```
tcp_nodelayack=1
```
- Maximum user process: If the maximum number of processes for a single user exceeds 2000, increase the value of `maxuproc` to prevent SAS processes from abnormal shutdown or delay. Increase the `maxuproc` setting by using the AIX `smit` or `chdev` commands, for example:


```
chdev -l sys0 -a maxuproc=<new value>
```
- User limits: Increase user-process resource limits for SAS users and database instances as appropriate (for example, `unlimited` or some tuned value for all resources. In the `/etc/security/limits` file, set `-1` for all resources.

Default AIX resource limits: The default AIX user-process resource limits might be too low for SAS power users or large enterprise-class deployments of SAS. When SAS processes end because of attempts to exceed these resource limits, the system administrator typically sets all of the user process resource limits to `unlimited` (a numeric value of `-1`) for users who run SAS. The problem with this approach is that the increased multi-threading and scalability support in newer versions of SAS, coupled with an `unlimited` setting for user-process resource limits, allows other users to potentially exhaust system resources, such as processor, memory I/O, and paging space. Before you increase the user-process resource limits, such as memory, to high values, consider the potential consequences.

- Paging space: Configure the paging space to include at least the following items:
 - Place paging spaces on dedicated disks to eliminate I/O contention.
 - Use multiple paging spaces that are spread across multiple disks.
 - Make the primary paging space hd6 a little bigger than the secondary paging spaces.
 - Ensure that the paging space is sufficient to support the number of concurrent SAS processes, because the number of SAS processes can be dynamic, depending on application workload.
- Volume groups (VGs): Use the AIX Scalable or Big volume group.

The scalable VG implementation provides configuration flexibility regarding the number of physical volumes (PVs) and logical volumes (LVs) that an instance of the new VG type can accommodate. The configuration options allow any scalable VG to contain 32, 64, 128, 256, 512, 768, or 1024 disks and 256, 512, 1024, 2048, or 4096 LVs. You do not need to configure the maximum values of 1024 PVs and 4096 LVs when you create the VG to account for potential future growth. You can increase the initial settings later, as required.
- Disk layout: Minimize disk contention between SAS temporary space and data spaces by considering the following items:
 - Avoid disk contention by placing SAS temporary-space file systems and SAS data file systems on physically separate disks.
 - Use multiple storage-server controllers to further separate and isolate the I/O traffic between SAS temporary and data spaces.
 - Use multiple mount points for SAS file systems. Place the operating system, SAS installation, SAS user, SAS temporary space, and SAS data file systems on separate physical disks.
 - Consider creating multiple SAS WORK areas that are used by groups of SAS users.
 - Create separate JFS2 log files on separate physical disks for each SAS file system.
 - Spread the I/O workload across many physical disk spindles rather than across fewer, larger-capacity disks. Determine the sizing, based on the quantity of disks rather than on disk capacity. Do not wrap logical unit numbers (LUNs) around the same spindle sets.
 - Do not share disk spindles with a relational database management system (RDBMS).
- Release-behind mechanism for JFS2: This feature allows the file system to release the file pages from file system buffer cache when an application reads or writes the file pages. This feature helps when the SAS application performs a great deal of sequential reads or writes, and after the file pages are accessed, they are not accessed again soon. This feature can be configured on a file system basis. When you use the `mount` command, enable release-behind by specifying one of the following flags:
 - Release-behind sequential read flag (`-rbr`),
 - Release-behind sequential write flag (`-rbw`),
 - Release-behind sequential read and write flag (`-rbrw`).

- Host bus adapters (HBAs): Use enough HBAs from storage to the host server to provide the required application bandwidth.
 - Consider high-performance storage channels, such as Fibre Channel (FC) technology instead of slower mediums.
 - If possible, use dynamic multipathing to spread the I/O load across multiple adapters.
- Redundant Array of Independent Disks (RAID): Implement storage system RAID striping across multiple physical disks.
 - Use RAID10 or RAID5, depending on the level of redundancy and total capacity instead of the usable capacity that is needed for each file system type.
 - Use Logical Volume Manager (LVM) striping instead of concatenation.
- LVM striping: When you choose the disk stripe or segment size, or array stripe size, AIX file systems are aligned on a 16 KB boundary.
 - A strip is the size of data to be written to each physical disk in the array. A stripe is the size of the full write across all the physical disks in the array. For example: strip size x number of disks = stripe size.
 - The AIX LVM stripe size that you can select from the `smit lv create` panel is the single strip size (not stripe). It is the size of data to be written to each of the array disks; it is not the full stripe size across all the physical disks. Consider using an LVM stripe size of 64 KB or 128 KB. Stripe sizes of 256 KB or 512 KB show better I/O performance in the case of SAS 9 workloads.
 - Synchronize SAS BUFSIZE with the storage-system stripe size and the AIX LVM stripe size (if you are using LVM striping) and VMM read-ahead increments.
 - Synchronizing I/O sizes streamlines I/O processing and reduces the number of I/O requests to the storage subsystem.

Disk storage tuning

From a high level, the AIX I/O stack contains several layers that an I/O must traverse. At each layer, AIX tracks the I/O. Some of the layers have specific queues that are useful to consider tuning. The I/O stack layers are:

- ▶ Application
- ▶ File system (optional)
- ▶ LVM (optional)
- ▶ Subsystem device driver (SDD) or SDD Path Control Module (SDDPCM) (if used)
- ▶ hdisk device driver
- ▶ Adapter device driver
- ▶ Interconnect to the disk
- ▶ Disk subsystem
- ▶ Disk

In this section, the focus is on tuning the middle layers, which consist of SDD, hdisk, and adapter device drivers. The goal is to improve simultaneous I/O capability and realize efficient queue handling. See Table C-3 for some of the parameters that can affect disk and adapter performance. In general, SAS applications benefit from careful consideration and tuning of these parameters.

Table C-3 Disk and adapter I/O tuning parameters

Parameter	Description
max_xfer_size	FC adapter maximum I/O that is issued.
max_transfer	Disk maximum I/O that is issued.
queue_depth	Disk maximum number of simultaneous I/Os. The default is 20 but can be set as high as 256 for IBM Enterprise Storage Server® (ESS), IBM System Storage® DS6000™, and IBM System Storage DS8000®.
num_cmd_elems	FC adapter maximum number of simultaneous I/Os. The default is 200 per adapter but can be set up to 2048.
qdepth_enable	Subsystem Device Driver (SDD) data path optimizer (dpo) device queuing parameter. The default is yes; setting this parameter to no disables SDD queuing. Use this parameter with IBM Enterprise System Storage, IBM System Storage DS6000, and IBM System Storage DS8000 storage.
lg_term_dma	Long-term DMA - Memory area the FC adapter uses to store I/O commands and data.
LTG	AIX volume group Logical Track Group parameter. LTG specifies the largest I/O the LVM issues to the device driver. In AIX 5.3, the LTG dynamically matches the disk maximum transfer parameter.

Both the disk and adapter have maximum transfer parameters that can be adjusted to handle larger I/O, reduce I/O splitting, and coalesce I/O as it moves up and down the stack. In addition, both have I/O queues that can be adjusted to accept more I/Os.

If SDD is used (IBM System Storage DS6000 or IBM System Storage DS8000), evaluate the data path optimizer (dpo) device I/O queue. SDD provides a virtual path (vpath) to the storage subsystem LUN and logical disk and provides several hdisk devices through the physical paths (such as FC adapters). So, with SDD you can issue **queue_depth** times the *number of paths to a LUN*.

However, when the dpo device queue is enabled (the default is yes), any excess I/Os that cannot be serviced in the disk queues go into the single wait queue of the dpo device. The benefit of this situation is that the dpo device provides fault-tolerant error handling. This situation might be wanted for high availability applications, but for other applications, there are advantages to disabling the dpo device queue and using multiple hdisk wait queues for each SDD vpath device.

AIX limitations for I/Os: This is not an exhaustive description and does not detail all possible AIX limitations for total number of I/Os. Also, carefully evaluate the queue parameters before you implement any changes. For tuning guides specific to a particular IBM storage system such as the IBM System Storage DS4000®, DS6000, or DS8000, see Appendix B, “Performance tooling and empirical performance analysis” on page 155.

Queue information can be monitored in AIX 5L V 5.3 and later by running `iostat -D`. For AIX 5L V5.1 and AIX 5L V5.2, SAR can be used. Run `qdepth_enable=no` to use the hdisk wait queue rather than the dpo device wait queue.

You should increase `num_cmd_elems` for the FC adapter from the default (initially starts at 400). Some of these parameters require a system reboot to take effect.

Run the following commands to display and modify disk and adapter parameters and settings:

- ▶ Disk: `max_transfer` and `queue_depth`
 - `'lquerypv -M hdisk#'` displays the maximum I/O size a disk supports.
 - `'lsattr -El hdisk#'` displays the current disk values.
 - `'lsattr -R -l hdisk# -a max_transfer hdisk#'` displays the allowable values.
 - `'chdev -l hdisk# -a max_transfer=value -P'` modifies the current disk values.

Device state: The device must be in an offline or disabled state before you change any parameters, and then you must run `cfgmgr`.

- ▶ Adapter: `max_xfer_size`, `lg_term_DMA`, and `num_cmd_elems`
 - `'lsattr -El fcs#'` displays the current value.
 - `'chdev -l fcs# -a max_xfer_size=value -P'` modifies the current value.

Device state: The device must be in an offline or disabled state before you change any parameters, and then you must run `cfgmgr`.

- ▶ SDD/DPO: `qdepth_enable`
 - `'lsattr -El dpo'` displays the current value.
 - Run `datapath` to change the parameters if at SDD 1.6 or greater. Otherwise, run `chdev`. For example:
`'datapath set qdepth disable'`

Available documentation

For detailed tuning information, see the following publications:

- ▶ *Best Practices for Configuring your IO Subsystem for SAS9 Applications*;
<http://support.sas.com/rnd/papers/sgf07/sgf2007-iosubsystem.pdf>
- ▶ *How to Maintain Happy SAS Users*;
<http://support.sas.com/resources/papers/happyIT.pdf>

- ▶ *AIX V6 best practices for SAS Enterprise Business Intelligence (SAS eBI) users on IBM POWER6:*
<http://www.sas.com/partners/directory/ibm/AIXBestPractice.pdf>
- ▶ *IBM General Parallel File System (GPFS) wiki for SAS:*
<http://www.ibm.com/developerworks/wikis/display/hpccentral/SAS>
- ▶ *Understanding Processor Utilization on Power Systems - AIX:*
<http://www.ibm.com/developerworks/wikis/display/WikiPtype/Understanding+Process+or+Utilization+on+POWER+Systems+--+AIX>

Migrating SAP BusinessObjects Business Intelligence platform with POWER7

The SAP BusinessObjects Business Intelligence (SBOP BI) platform enables enterprises to realize key insights about their corporate data. By delivering interactive business intelligence (BI) reports to users using any web application (that is, Internet or intranet), SAP BusinessObjects provides the information necessary to make more informed business decisions. As an integrated suite for reporting, analysis, and information delivery, SAP BusinessObjects BI provides a solution for increasing user productivity and reducing administrative efforts. You can analyze data from any of the SAP systems and from many supported database systems (including text or multi-dimensional online analytical processing (OLAP) systems). You can publish in many different formats to many different publishing systems.

Platform support

Although the front-end reporting tools are mostly web browser or Windows application based, the server components of the SAP BusinessObjects BI platform can run on POWER servers using the AIX operating system.

The currently supported AIX versions for SBOP BI are:

- ▶ AIX 5L V5.3 TL3 SP3
- ▶ AIX V6.1 TL5
- ▶ AIX V7.1 TL1 SP1

The supported operating systems for SBOP BI components are listed in *Platform Availability Matrix (PAM)* at the SAP Service Marketplace, which is available at <http://service.sap.com/pam> (registration required; search for *SBOP BI platform*).

SBOP BI itself requires a database for its Central Management Service (CMS). For larger SBOP BI installations, run this database on a dedicated server. The supported databases for CMS are listed in an embedded Excel spreadsheet in the PAM document, and include the most common databases, such as IBM DB2.

SBOP BI can use various data sources for analysis and reporting. Besides SAP systems in general, and SAP Business Warehouse systems specifically, SBOP BI can also directly access database systems. The types of database systems that are supported as data sources are also listed in the embedded Excel spreadsheet of the PAM document.

Although the PAM document is the official source for any SAP-related platform support, the SAP Community Network topic, *Supported Platforms/PARS for SAP Business Objects* (<http://www.sdn.sap.com/irj/boc/articles?rid=/webcontent/uuid/e01d4e05-6ea5-2c10-1bb6-a8904ca76411>) provides a good overview of SAP BusinessObjects releases and supported platforms.

Sizing for optimum performance

Adequate system sizing is essential for successfully running an SBOP BI solution. Therefore, SAP includes SBOP BI in its Quick Sizer tool. Based on the number of users who are planning to use the SBOP BI applications, the Quick Sizer tool provides an SAP Application Performance Standard (SAPS) number and the memory necessary to run the applications. IBM can then provide the correct system configuration that is based on these numbers. The Quick Sizer tool is available at <http://service.sap.com/quicksizer> (registration required).

Also, the *SAP BusinessObjects BI 4 - Companion Guide* is available on the SAP Quick Sizer landing page at <http://service.sap.com/quicksizer> (registration required). To download the guide, open the web page and click **Sizing Guidelines** → **Solutions & Platforms** → **SAP BusinessObjects**. This guide explains the sizing process in detail using a sizing example.

Landscape design

For general considerations about how to design an SBOP BI landscape, see the *Master Guide* (registration required), available at:

http://service.sap.com/~sapidb/011000358700001237052010E/xi4_master_en.pdf

There are six different reference architecture landscapes available that demonstrate how to implement an SBOP BI solution that is based on the type of enterprise resource planning (ERP) and data warehouse solution you are using.

The architecture documents are in the *Getting Started with SAP BusinessObjects BI Solutions* topic on the SAP Community Network (<http://scn.sap.com/docs/DOC-27193>) in the Implementation and Upgrade section.



POWER7 and POWER7+ Optimization and Tuning Guide

(0.2"spine)
0.17"->0.473"
90->249 pages



POWER7 and POWER7+ Optimization and Tuning Guide



Discover simple strategies to optimize your POWER7 environment

Analyze and maximize performance with solid solutions

Learn about the new POWER7+ processor

This IBM Redbooks publication provides advice and technical information about optimizing and tuning application code to run on systems that are based on the IBM POWER7 and POWER7+ processors. This advice is drawn from application optimization efforts across many different types of code that runs under the IBM AIX and Linux operating systems, focusing on the more pervasive performance opportunities that are identified, and how to capitalize on them. The technical information was developed by a set of domain experts at IBM.

The focus of this book is to gather the right technical information, and lay out simple guidance for optimizing code performance on the IBM POWER7 and POWER7+ systems that run the AIX or Linux operating systems. This book contains a large amount of straightforward performance optimization that can be performed with minimal effort and without previous experience or in-depth knowledge. This optimization work can:

- ▶ Improve the performance of the application that is being optimized for the POWER7 system
- ▶ Carry over improvements to systems that are based on related processor chips
- ▶ Improve performance on other platforms

The audience of this book is those personnel who are responsible for performing migration and implementation activities on IBM POWER7-based servers, which includes system administrators, system architects, network administrators, information architects, and database administrators (DBAs).

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8079-00

ISBN 0738437530