VT-ASOS: Holistic System Software Customization for Many Cores

VT-ASOS: Holistic System Software Customization for Many Cores

Dimitrios S. Nikolopoulos, Godmar Back, Jyotirmaya Tripathi and Matthew Curtis-Maury
Department of Computer Science, Virginia Tech
dsn@cs.vt.edu, gback@cs.vt.edu, tripathi@cs.vt.edu, mfcurt@cs.vt.edu

Abstract

VT-ASOS is a framework for holistic and continuous customization of system software on HPC systems. The framework leverages paravirtualization technology. VT-ASOS extends the Xen hypervisor with interfaces, mechanisms, and policies for supporting application-specific resource management schemes on many-core systems, while retaining the advantages of virtualization, including protection, performance isolation, and fault tolerance. We outline the VT-ASOS framework and present results from a preliminary prototype, which enables static customization of scheduler parameters and runtime adaptation of parallel virtual machines.

1. Introduction

High-performance computing systems and virtualization technologies were recently brought together. Virtualization frameworks offer functionality that enhances HPC systems with capabilities such as fault tolerance, workload migration, and customization of the application execution environment, including shells, environment variables, runtime libraries and operating system components [7, 10, 5]. Virtualization facilitates holistic and targeted software development, to better support parallel applications [11]. Other well-known advantages of virtualiation that may benefit specific HPC user groups are performance isolation, security, and consolidation. While virtualization has served the server computing domain well so far, HPC systems present new opportunities and challenges for virtualization environments.

Several research groups are exploring the performance, potential, and drawbacks of current virtualization technologies for HPC systems [8]. These studies seem to converge to two conclusions. The first is that paravirtualization frameworks are more suitable for HPC environments than full system virtualization frameworks [11]. Paravirtualization reduces overhead by replacing full hardware emulation with a small set of abstractions for processor, mem-

ory, and I/O devices, and by using a direct communication interface between virtual machines and the hypervisor. The latter provides virtual machines with the capability to perform privileged operations through the hypervisor, while retaining close control over the resources allocated to them. The second conclusion of ongoing research on virtualization for HPC environments is that, though extensively studied, current paravirtualization frameworks still impose bottlenecks for HPC applications. In particular, frequent context switches between virtual machines and the hypervisor and extensive data copying, hurt the performance of applications with high demands for I/O or communication [8].

VT-ASOS, an NSF-CSR project the acronym of which stands for "Virtualization Technologies for Application-Specific Operating Systems", takes a performance-oriented approach to virtualization for HPC systems. We proposed VT-ASOS to deliver holistic optimization capabilities to system software developers on emerging many-core systems. We view VT-ASOS as a framework to enable higher performance for parallel applications, by replacing potentially suboptimal resource management policies in the system software components —user-level and kernel-level schedulers, memory allocators, communication libraries and I/O device drivers—, with optimized application-specific ones. VT-ASOS aims also at eliminating the noise introduced by system software during the execution of parallel applications, and minimize the footprint of the system software stack to the bare essentials, leaving more resources available to user-level code.

The design of VT-ASOS is based on the premise that existing paravirtualization frameworks, such as Xen, can execute parallel applications with small overhead compared to non-virtualized frameworks, at least in tightly coupled systems such as compute nodes with a few multi-core processors. In this paper, we present an outline of the design of VT-ASOS, and proceed to solidify the motivation behind this research with results from paravirtualized execution of OpenMP and MPI applications on state-of-the-art multi-core processors. The results illustrate that the unmodified Xen hypervisor supports Linux guests running HPC applications with little or no perturbation during their execution,
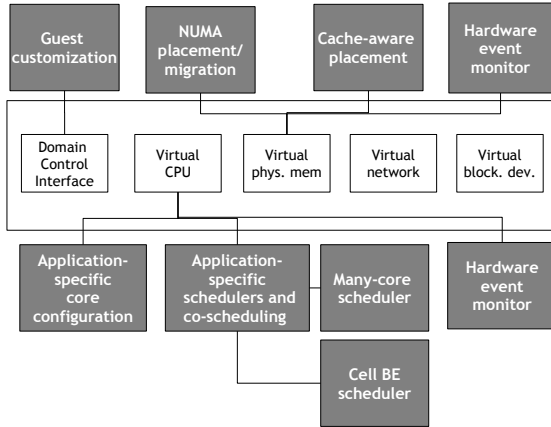
Figure 1. VT-ASOS modules.

provided that virtual machines are aware of the implications of the mapping of threads to cores, and OS-induced noise is minimized via proper control of the frequency of timer interrupts in the hypervisor. We introduce extensions of the shared-memory interface of Xen to enable adaptive parallel execution, granularity control, and phase-aware optimization of parallel codes on multi-core platforms. The extended interface enables higher performance for individual applications and provides opportunities for consolidation of applications and improved hardware utilization on HPC platforms.

2. VT-ASOS Design

VT-ASOS is a paravirtualization framework for holistic application-specific customization of system software stacks. The framework enables a user to configure the software environment of a many-core system with a minimal set of highly optimized components for executing parallel applications.

VT-ASOS extends the hypervisor of Xen [1]. The VT-ASOS hypervisor provides components that enable: i) customization of the mapping of virtual processors to cores; ii) performance isolation between disjoint sets of cores; iii) customization of parameters of CPU schedulers or replacement of CPU schedulers altogether with highly tuned alternatives for co-scheduling threads in parallel applications; iv) minimization of hypervisor and operating system interference during computation-intensive and communication-intensive parallel execution; v) implementation of application-specific policies for data placement and data transfers along the physical memory hierarchy. The VT-ASOS hypervisor integrates a runtime hardware performance monitoring infrastructure, which enables dy-

namic inference of workload properties by the hypervisor and guest virtual machines, through statistical analysis of samples collected from hardware event counters. Figure 1 summarizes the components of VT-ASOS. We outline the principal components in the following subsections.

2.1. Custom core configurations

The introduction of multi-core execution in processors affects parallel applications in subtle and non-trivial ways. Parallel applications do not necessarily scale gracefully to any number of cores on a multi-core system, whereas both performance and energy consumption are very sensitive to the mapping of threads to cores. System configurations with the same degree of active concurrency have very different performance signatures. Even applications that are immune to the mapping of threads to cores often have very different power signatures and optimized thread to core mappings can yield substantial energy savings [2]. Moving from a handful to a hundred processor cores in the near future will only exacerbate these problems.

In VT-ASOS, the application's concurrency and memory locality patterns are directly communicated to the VMM hypervisor, through the guest OS. Rather than tuning the policies at the runtime or guest OS layer, the hypervisor itself computes an assignment of guest virtual processors to cores that optimizes resource use. The hypervisor communicates continuously with the guest operating system and adjusts at runtime the degree of concurrency exposed by the guest and the mapping of guest virtual processors to cores. The adjustment of the mapping of the application to the system happens in response to either direct workload measurements conducted by the guests and communicated to the hypervisor, or through analysis of system utilization and performance metrics conducted in the hypervisor. Our current prototype follows the first approach.

2.2. Custom scheduling

The schedulers in hypervisors such as Xen, are primarily designed for commercial server consolidation scenarios, in which different guest domains are isolated from each other. In such scenarios, achieving fairness and maximizing utilization are paramount. Current schemes do not provide gang scheduling, or any synchronization-aware scheduling, both necessary features for parallel applications running on multi-core systems. Moreover, the interaction between the hypervisor's scheduler and the schedulers of the guest domains is not understood well, as is evidenced by the frequent turnover of experimental scheduling algorithms in these systems. In VT-ASOS, a guest domain's scheduling requirements (such as capacity and latency requirements and synchronization and communication patterns) are di-

rectly communicated to and enforced by the hypervisor. The hypervisor takes into account these requests to prioritize the threads of each application in strict accordance with application-specific scheduling priorities. These priorities are dictated by such events as inter-core/inter-processor synchronization, network communication, and I/O. Furthermore, VT-ASOS uses adjustable frequencies of timer interrupts to minimize operating system noise, while maintaining responsiveness in applications with a blend of compute-intensive and synchronization-intensive execution phases.

2.3. Custom memory management

Emerging architectures have widely non-uniform memory access times, even at the same level of the memory hierarchy (e.g. DRAM or an outermost-level NUCA cache). Parallel applications are sensitive to memory latency and must therefore control the placement and movement of their data in physical memory and caches. In a virtualized architecture, the underlying hypervisor must be involved in the assignment of memory to guest domains. VT-ASOS addresses these problems by exposing information about the physical layout of memory to the guest operating systems through the hypervisor, effectively enabling guest applications to tune their data placement as if they were running on a non-virtualized environment. As new multi-core architectures move towards explicitly managed memory hierarchies [4], functionality for application-specific memory management becomes essential.

2.4. Current Prototype

The current VT-ASOS prototype extends xenstore — Xen's interface for hypervisor-guest OS communication— with asynchronous notifications of scheduling events that occur in guest operating systems and a mechanism for streaming performance data from guest operating systems to the hypervisor and vice versa. The extended xenstore interface is used to dynamically adapt the concurrency and mapping of virtual processors to cores. Adaptation is implemented with application-specific algorithms. We have tested VT-ASOS with an adaptation framework for OpenMP codes, which enables dynamic concurrency throttling and control of the mappings of threads to cores at the granularity of parallel loops [2]. The VT-ASOS prototype implements also a user interface for static customized configuration of virtual machines. The interface can be used for configuring cores and memory nodes, as well as for tuning of scheduling parameters in both the guest operating system and the hypervisor. We are using this interface in particular, to tune the frequencies of the timer interrupts in Xen, and minimize hypervisor noise.
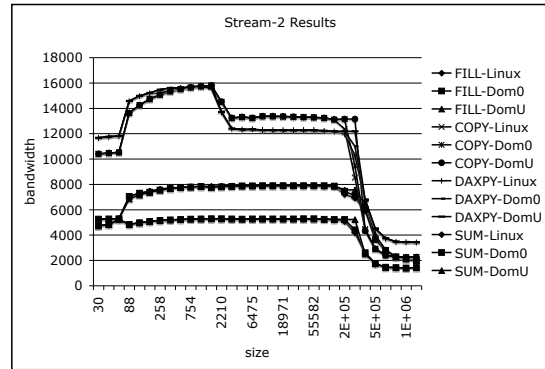

Figure 2. STREAM-2 microbenchmark. The four kernels FILL, COPY, DAXPY and SUM are executed in virtualized and non-virtualized mode.

3. Performance Analysis

We present preliminary performance results of VT-ASOS components in HPC environments based on multi-core processors, using microbenchmarks and parallel applications. Our performance analysis attempts to: i) assess the overhead imposed by Xen in operations lying typically on the critical path of HPC applications; ii) explore if VT-ASOS can support custom execution environments for adaptive parallel applications, and what are examples of static or dynamic optimizations that VT-ASOS enables to improve throughput in HPC systems.

Our experimental platform is a compute node with a quad-core Xeon 5335 processor. The processor is 64-bit and boasts 32 KB of L1-D cache per core, 32 KB of L1-I cache per core, and a 4 MB L2 cache per pair of cores sharing the same socket. In the experiments, we used Linux 2.6.18 in native mode and in paravirtualized mode as a guest OS. We used Xen 3.1.0, both unmodified and as a substrate for VT-ASOS extensions, in our paravirtualized experiments.

3.1. Microbenchmarks

Figure 2 and Figure 3 illustrate the performance of native Xen with two microbenchmark suites. The STREAM-2 suite measures the bandwidth of the system across all layers of the memory hierarchy. The EPCC suite evaluates the overhead of OpenMP constructs for spawning and managing parallelism and synchronization in OpenMP codes.

Xen imposes overhead mainly due to crossings (context switches) between the guest OS and the hypervisor,
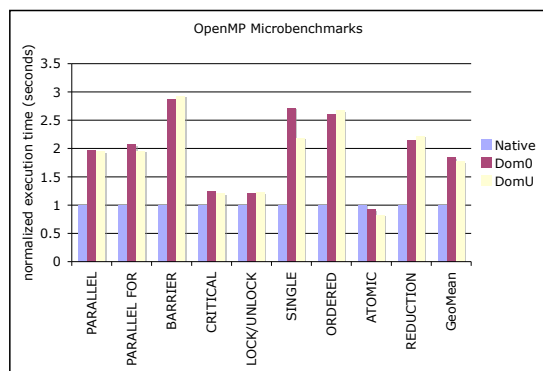
Figure 3. OpenMP microbenchmarks.

Figure 4. OpenMP NPB performance.

Figure 5. MPI NPB performance.

as well as between the guest OS and the driver domain (known as Dom0), which are needed whenever the guest OS needs to execute privileged operations. The overhead is prevalent during I/O operations, as the system needs to perform multiple crossings (guest OS to hypervisor, hypervisor to driver domain, driver domain to hypervisor, hypervisor to guest OS), in order to move data between external devices and memory [6]. Network communication and I/O not-withstanding, Xen may impose context switching and hypervisor-induced overhead for updates to page tables, guest process creation and management operations, and guest-induced context switches.

Figure 2 illustrates that Xen does not penalize the bandwidth of the memory hierarchy on the quad-core system. We observe that neither the guest OS nor the driver domain suffer penalties in raw memory performance due to paravirtualization. This result may not generalize to NUMA or NUCA systems, where memory performance is sensitive to the physical placement of data in memory. Xen's support for NUMA systems is still in an experimental stage and NUMA memory layouts are not exposed to guest operating systems.

Figure 3 illustrates that Xen imposes a 2.0x–3.0x overhead in operations that involve thread creation and management for the guest operating systems, whereas the overhead for simple synchronization operations such as locks and atomic accesses to critical sections is more affordable (no more than 1.2x). This result indicates that the implementation of OpenMP in gcc incurs excessive hypervisor calls and context switches during paravirtualized execution. We are exploring techniques for reducing this overhead, such as batch creation of threads and virtual processor co-scheduling during synchronization operations, with assistance from the hypervisor.
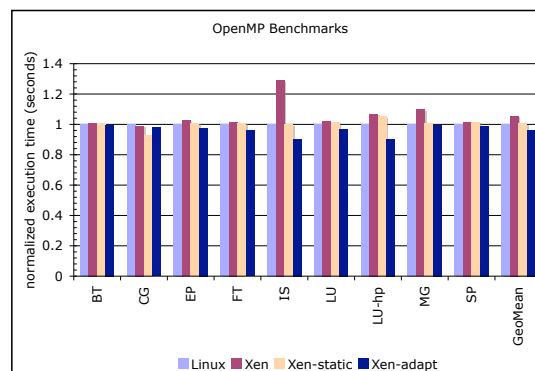
3.2. OpenMP Applications

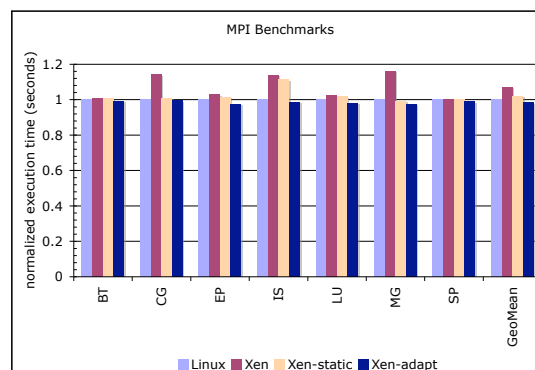Figure 4 and Figure 5 illustrate the performance of Xen with OpenMP and MPI application benchmarks from the NAS Parallel Benchmarks collection (NPB), in three settings. The first setting uses an unmodified version of Xen. The second setting uses a version of the Xen-VTASOS prototype (Xen-static), which statically customizes the mapping of virtual CPUs to cores and tunes the frequency of timer interrupts in the hypervisor, to minimize operating system noise while guaranteeing responsiveness during synchronization operations. The Xen-static configuration uses feedback collected off-line and communicated from the guest virtual machines to Xen at boot time. The third setting uses an enhancedversion of the Xen-VTASOS prototype (Xen-adapt) which uses xenstore and asynchronous

notifications between virtual machines and the hypervisor to perform phase-aware granularity control (i.e. tuning the degree of concurrency exposed from virtual machines to the hypervisor) and phase-aware mapping of threads to cores at runtime, combined with application-specific tuning of the frequency of timer interrupts.

We observe that customized static configuration of the guest operating systems retrieves the performance loss due to hypervisor interference. The overhead of Xen is noticeable in applications with relatively frequent barriers and reduction operations. The adaptive version of Xen improves the performance of parallel applications further than Xen-static and beyond native Linux, thanks to performance-aware and application-aware granularity control and mapping of threads to cores, in addition to the reduction of the operating system and hypervisor noise. Sensitivity analysis indicates that both adaptive execution and tuning of interrupt frequency contribute significantly to the performance gains in OpenMP applications. In MPI applications (Figure 5), runtime adaptation is not possible without modifying the applications and the performance gains from controlling interrupt frequency are limited and most visible in communication-intensive codes.

4. Conclusion

VT-ASOS is exploring mechanisms, policies, and interfaces in virtualization environments, to support application-specific customization of software on many-core HPC systems. Our early experience suggests that current hypervisor designs are not intrusive for well-designed parallel applications, however the scalability of their mechanisms for creating, scheduling, and synchronizing parallel computation may be limited to a few cores. Furthermore, hypervisors lack the necessary flexibility to support parallel workloads with diverse execution signatures. Our research attempts to move from passive hypervisors design to active hypervisor designs, whereby the hypervisor provides a precisely tuned and adaptive hardware/software environment for parallel applications. While VT-ASOS shares motivation and ideas with other virtualization platforms, such as PROSE [3] and resource containers [9], it also departs from these works in that it attempts to break down the abstraction barriers between applications, libraries, operating systems, hypervisors, and the bare metal.

Our current VT-ASOS research proceedsin three directions. The first is the acceleration and partial automation of the static virtual machine customization process. The second is the introduction of an interface that enables awareness of the physical memory layout in guest operating systems. The third is the development of a more adaptable hypervisor scheduler which leverages application feedback throughout the scheduling process.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[2] M. Curtis-Maury, F. Blagojevic, D. Nikolopoulos, B. DeSupinski, and M. Schulz. Prediction Models for Multi-Dimensional Power-Performance Adaptation on Many Cores. Technical report, Department of Computer Science, Virginia Tech, Jan. 2008.

[3] E. V. Hensbergen. P.R.O.S.E.: partitioned reliable operating system environment. *SIGOPS Oper. Syst. Rev.*, 40(2):12–15, 2006.

[4] T. J. Knight et al. Compilation for explicitly managed memory hierarchies. In K. A. Yelick and J. M. Mellor-Crummey, editors, *PPOPP*, pages 226–236. ACM, 2007.

[5] A. M. Matsunaga, M. O. Tsugawa, M. Zhao, L. Zhu, V. Sanjeepan, S. Adabala, R. J. O. Figueiredo, H. Lam, and J. A. B. Fortes. On the use of virtualization and service technologies to enable grid-computing. In J. C. Cunha and P. D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[6] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23, New York, NY, USA, 2005. ACM Press.

[7] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 23–32, New York, NY, USA, 2007. ACM.

[8] H. Raj and K. Schwan. High performance and scalable i/o virtualization via self-virtualized devices. In C. Kesselman, J. Dongarra, and D. Walker, editors, *HPDC*, pages 179–188. ACM, 2007.

[9] S. Soltesz, H. Pötzl, M. Fiuczynski, A. Bavier, and L. Peterson. Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. In *Proc. of EuroSys'2007*, Lisbon, Portugal, Mar. 2007.

[10] G. Vallée, T. Naughton, and S. L. Scott. System management software for virtual environments. In *Conf. Computing Frontiers*, pages 153–160, 2007.

[11] L. Youseff, R. Wolski, B. C. Gorda, and C. Krintz. Paravirtualization for hpc systems. In G. Min, B. D. Martino, L. T. Yang, M. Guo, and G. Rünger, editors, *ISPA Workshops*, volume 4331 of *Lecture Notes in Computer Science*, pages 474–486. Springer, 2006.