Functional verification of the POWER4 microprocessor and POWER4 multiprocessor systems

by J. M. Ludden
W. Roesner
G. M. Heiling
J. R. Reysa
J. R. Jackson
B.-L. Chu
M. L. Behm
J. R. Baumgartner
R. D. Peterson
J. Abdulhafiz
W. E. Bucy
J. H. Klaus
D. J. Klema
T. N. Le
F. D. Lewis
P. E. Milling
L. A. McConville
B. S. Nelson
V. Paruthi
T. W. Pouarz
A. D. Romonosky
J. Stuecheli
K. D. Thompson
D. W. Victor
B. Wile

This paper describes the methods and simulation techniques used to verify the microarchitecture design and functional performance of the IBM POWER4 processor and the POWER4-based Regatta system. The approach was hierarchical, based on but considerably expanding the practice used for verification of the CMOS-based IBM S/390 Parallel Enterprise Server™ G4. For POWER4, verification began at the abstract, high-level design phase and continued throughout the designer and unit levels, the multi-unit level, and finally the multiple-chip system level. The abstract (high-level design) phase permitted early validation of the POWER4 processor design prior to its commitment to HDL. The designer and unit-level stages focused on ensuring the correctness of the microarchitectural components. Multi-unit-level verification, performed on storage and I/O components as well as on the processor, confirmed architectural compliance for each of the chips and subsystems. Finally, system-level verification tested multiprocessor coherence and system-level function, including processor-to-I/O communication and validation of multiple hardware configurations. In parallel with design and functional validation, verification of reliability functions, performance, and degraded configurations was also performed at most of the levels in the hierarchy.

Introduction

The sophistication of the POWER4 processor and system design required a major logic verification effort. Verifying the POWER4 superscalar, out-of-order execution processor, which can allow more than 200 simultaneous "in-flight" instructions, required detailed test plans, a breadth of simulation technology, and a staged execution plan. The 32-way multiprocessor system configuration created further verification challenges for validating cache coherency, system integrity, and memory management.

To successfully verify this logic design, the verification team employed a wide array of methodologies, using a

0018-8646/02/$5.00 © 2002 IBM

hierarchical approach to exploit different optimal methodologies as appropriate for each level. The combination of different methods at different levels enabled each level to attack the verification problem from its own unique angle, establishing new test scenarios and checking for correctness across all functions contained in the particular level.

The hierarchical verification approach expanded the practice used on the CMOS S/390 Parallel Enterprise Server* G4 system [1]. For POWER4, verification began at the abstract, high-level design phase, continued at the designer and unit levels and the multi-unit level, and was completed at the multiple-chip, system level. The abstract, high-level design phase of the verification allowed for early validation of the POWER4 processor design, prior to commitment to hardware description language (HDL). The designer- and unit-level verification stages focused on ensuring the correctness of the microarchitectural components. Multi-unit-level verification, performed on the processor, storage, and I/O components, confirmed architectural compliance for each of the chips and subsystems. System-level verification performed multiprocessor (MP) coherence and system-level function testing, including processor-to-I/O communication and validation of multiple hardware configurations. In parallel with these efforts, verification of reliability functions, performance, and degraded configurations was performed at many of these levels of the hierarchy.

As the high-level verification test plan was constructed, known methods were considered. Methodologies used included both proven verification technologies and new, leading-edge techniques. Proven methods included the use of command-driven random techniques, architectural test-case-generation engines at both the processor and the system level, system reset verification, and disabled or degraded configuration testing. Included among the new methods were high-level verification on the abstracted model, model checking, extended coverage techniques, and asynchronous clock modeling on cycle-simulation engines.

The line count for all of the very-high-speed integrated-circuit HDL (VHDL) of POWER4 was approximately 1.5 million. The verification environment added approximately another one million lines of C/C++ code, making this a very large chip-verification model. Aside from the special requirements for performance and robustness of the VHDL-processing tools, the sheer amount of code made robust data management a necessity.

Staffing for verification was obtained from a wide variety of sources and included both experienced and novice verification engineers. Expertise came from prior IBM programs, especially in the PowerPC* family. Experts were charged with teaching new verification engineers as well as leading the unit-, element-, and system-level verification teams. These leaders defined the detailed test plans; the test plans were then reviewed by experienced system-verification leaders from other IBM sites, and additional recommendations were applied. Peer reviews performed by verification leaders outside the POWER4 team were done throughout the life of the program.

This paper describes the verification plans, methodologies, execution, and experiences of the POWER4 functional verification effort. It begins by addressing the verification challenges presented by the complex microarchitecture. It addresses the methods employed at each level of verification and concludes with the results and achievements of the effort.

POWER4 architecture and microarchitecture overview

To fully appreciate some of the challenges involved, it is necessary to point out some aspects of the general PowerPC architecture and the implemented POWER4 microarchitectural features that exploit this architecture for maximum performance.

PowerPC RISC architecture

PowerPC [2] is traditionally recognized as a reduced-instruction-set computer (RISC) architecture which adheres to a basic philosophy of keeping the hardware design simple. The intent is to place more responsibility on software than is done by traditional complex-instruction-set computer (CISC) architectures such as the IBM S/390* or the Intel x86. Despite this notion of keeping things simple, the architecture presents many challenges to the verification process:

- The ordering of performance-critical instructions is often the responsibility of software and is accomplished by inserting context-synchronizing instructions into the instruction stream at the proper point. This increases the difficulty of verification because it is possible to create illegal instruction streams which lead to unpredictable hardware behavior.
- A weakly consistent memory model is assumed; the order in which loads and stores execute with respect to each other and to accesses by other processors can vary significantly. In general, loads and stores can execute out of order with respect to other processors and one another (on the same processor), provided that all accesses are consistent and properly aligned accesses are atomic.
- Address translation responsibility is shared between software and hardware. PowerPC translation uses a two-step approach which separates the "effective-to-real" translation mechanism into an "effective-to-virtual" address translation and then a "virtual-to-real" address translation.

- Logical partitioning (LPAR) in PowerPC provides software (known as the Hypervisor code) with a mechanism to run multiple operating systems on a single Regatta system. At most, each processor resides in a single partition. It is the responsibility of the Hypervisor software to establish the proper real address partitions to make this possible. However, it is the responsibility of hardware to prevent code running in one partition from interfering with code running in another partition.
- Concurrent modification and execution (CMODX) in the PowerPC architecture allows for the modification of one processor's instruction stream by another processor without the need for software synchronization between processors.

POWER4 microarchitecture implementation

In addition to the PowerPC architecture, the sophisticated microarchitecture, or implementation, of POWER4 [3] presented significant challenges to the verification team. Some of the more challenging features requiring verification in order to ensure adherence to the PowerPC architecture included

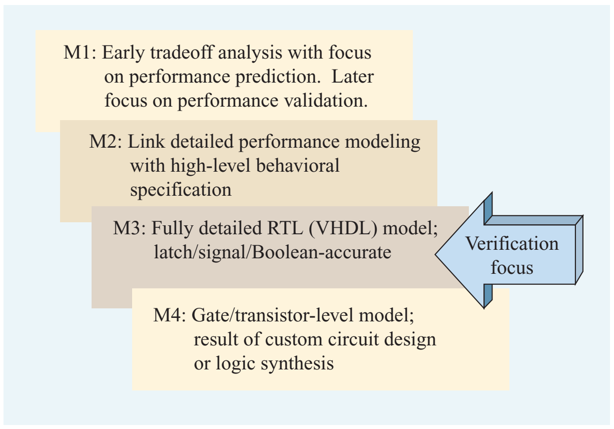
- Two superscalar, out-of-order processor cores per chip.
- Three separate buffer and caching mechanisms devoted to address translation.
- Performance-oriented data-access queues for simultaneous tracking of load misses and storage-access ordering.
- Aggressive prefetch for both instructions and data.
- Instruction cracking and microcode sequencing.
- Speculative execution for up to 16 unresolved branches.
- A three-tiered branch-prediction algorithm.
- Dual IEEE-compliant floating-point execution units.
- Three levels of cache: Each processor core has its own L1 instruction cache and L1 data cache. The L2 and L3 caches are shared by both cores and contain both instructions and data.

Overview of the design and verification flow

Logic design foundations

IBM has a successful history of more than twenty years in exploiting the benefits of a synchronous and mostly level-sensitive scan design (LSSD) rules-driven design style for functional verification [4]. The advantages of this methodology manifest themselves primarily as

- Design specification in a high-level (resistor-transistor-logic-level) hardware design language.
- Complete separation of functional verification from timing verification.
- Cycle-based simulation.



M1: Early tradeoff analysis with focus on performance prediction. Later focus on performance validation.
M2: Link detailed performance modeling with high-level behavioral specification
M3: Fully detailed RTL (VHDL) model; latch/signal/Boolean-accurate
M4: Gate/transistor-level model; result of custom circuit design or logic synthesis
Verification focus

Figure 1

The four main POWER4 model abstraction levels that were built.

- Simulation coverage analysis using properties of the high-level language specification.
- Formal Boolean equivalence proof of gate/transistor-level implementation vs. high-level design.

From the start it was the clear goal for POWER4 functional verification to base 90% of the functional verification work on the design specification in HDL. This requirement, however, had to be balanced against the ambitious goals of the project:

- A totally new, highly complex microarchitecture which demanded an earlier start of verification in order to be able to identify major microarchitectural problems before major effort had been committed to HDL coding.
- Aggressive use of custom design at the transistor level to achieve the clock-frequency goal of more than 1 GHz. This required the availability of an advanced formal equivalence-checking tool (Verity [4]) which supports all custom design styles necessary to achieve the frequency goal. This was an absolute prerequisite for decoupling functional verification from the transistor level and thus for doing effective verification at all.

Overview of design modeling levels

During the development cycle, several modeling abstractions of the design were used; these are shown in **Figure 1**.

The M1 model represents a traditional "timer" model. It models the machine strictly from a transaction viewpoint, interprets instruction traces of typical applications and benchmarks, and predicts machine performance. The modeling on this level was begun during the earliest days of the project. The model was maintained

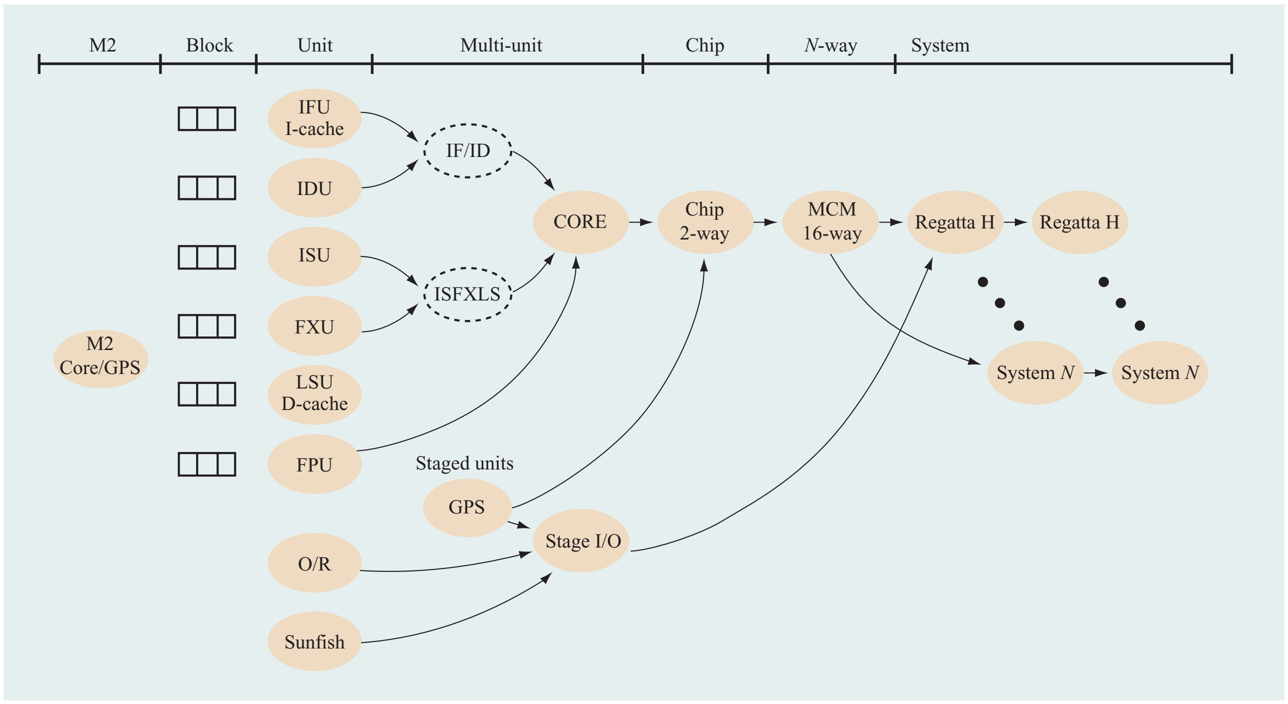


Figure 2

POWER4 verification hierarchy progressing from the M2 model to a 32-way Regatta H multiprocessor system M3 RTL model.

and adjusted even after the chip hardware existed, although its use shifted from performance prediction to performance analysis.

The M2 model is a new, abstract microarchitectural model; it is discussed in the next section.

M3 in this scheme is the name of the traditional RTL model, specified in VHDL. The VHDL model is the complete, detailed behavioral specification. It is accurate down to every latch, is Boolean-function-accurate with respect to the transistor-level implementation, and is the main focus of the verification process.

Finally, M4 is the "real" thing: the physical netlist that connects the custom transistor macros with gate-level netlists that were the output of logic synthesis.

The verification methodology began at the M2 modeling level, but most verification was performed at the M3 level. A hierarchical approach was used at the M3 level, starting with designer's blocks and progressing to system simulation, as shown in **Figure 2**. This approach optimized the bug-discovery rate by finding the greatest portion of bugs at the lowest possible level, where simulation-debug-fix iterations are the fastest.

Design modeling levels: M2

Today's HDL models are aimed at describing implementations and fail at capturing a designer's real intent. It was necessary to raise the level of abstraction and create a model that captures the design at the microarchitectural level. Many verification tasks could be improved with such a "high-level" model: simulation (speed), formal verification (model size, easy separation of control logic), coverage (obvious structures to instrument for coverage models), and test program generation (project-specific reference model for focused test generation).

At the beginning of the POWER4 project, it was decided that such a microarchitectural high-level model (the M2 model) was necessary for the processor core because the machine is so complex that it was not reasonable to predict the performance of the processor using only the traditional methods for performance modeling. The convergence of needs from the performance and verification teams made the existence of an additional model in the design process a necessity.

The M2 model was the primary design vehicle for the processor core during the first year of the project. A C/C++ modeling framework was created to enable a core team of microarchitects to write an efficient, concise microarchitectural model. The lower levels of the design hierarchy used C/C++ code, which supports model partitioning, simulation control flow, built-in elements such as latches, and performance-related constructs.

Furthermore, the system allowed the designers to use VHDL for the structural specification of the upper levels of the hierarchy. Not only is netlist-type information the obvious domain of an HDL, but early partitioning on this level was also a task for high-level physical design. Sharing the same source between physical and high-level performance/functional design was highly desirable.

The overall experience of microarchitectural modeling on the POWER4 project was very successful. Not only were reliable performance measurements derived from the model, but the verification process benefited in major ways:

- A machine-readable, executable specification proved early on that the processor in general "hangs together." Many fundamental flaws in the microarchitecture (such as hang situations) were found during high-level design.
- Simply running benchmarks for performance prediction through the model proved to have unexpected positive side effects on early verification. This code would normally have been run on the RTL model much later in the project.
- The verification infrastructure and the verification team got an early start with an executable model that was available one year earlier than the actual HDL model.

As a new addition to the methodology, the M2 approach had its problems, the most important one probably being that there is no "science" that defines the appropriate abstraction level for a "microarchitectural" model (as opposed to RTL, for instance). Over time a dynamic compromise was found, in which the focus shifted from performance modeling to functional verification over the course of the high-level design phase.

Design modeling levels: M3

The hardware description language of choice for POWER4 was a subset of VHDL, the industry-standard HDL, and the primary choice in the IBM Server Group. The definition of a subset had a basis in the tools as well as in the methodology decisions the project had made. TexVHDL, named after the cycle-based simulator TexSim* used in POWER4, limits VHDL to its synthesizable subset, which guarantees that the RTL specification can be mapped unambiguously to a Boolean network or a finite-state machine, which is the basis for cycle simulation and formal verification.

One of the methodology decisions in the POWER4 project was to require the RTL specification to be latch-accurate to the point of not allowing any automatic latch inference by tools. This meant that the logic designers had to instantiate the exact latch library elements for every state-holding element, as well as the LSSD scan path that connects the latches for manufacturing test. This VHDL style was appropriate because a large part of POWER4 was a custom-engineered chip. The advantage of having all of these physical aspects available on the RTL level was that the designers were able to verify pervasive chip-wide functions early in the project cycle, before the availability of any physical netlist. The Verity Boolean-equivalence check guaranteed the accuracy of the VHDL-level simulation results.

It was a key logic design decision for POWER4 to use a more advanced interpretation of the LSSD design rules, which allowed the separation of strict master–slave pair latches into two distinct and separate latch elements. This design style allowed for aggressive cycle stealing, which was required for the ambitious cycle-time design point.

For the cycle-based simulation approach, the existence of two discrete latches required two simulator cycles per POWER4 machine cycle. This would have divided the simulation throughput by a factor of 2 if it had not been for special provisions (referred to as "latch partitioning") in the model build programs. Latch partitioning allowed a standard master–slave latch to be modeled as a single state-holding element. For formal verification, this splitting of the latch pair would normally have meant a doubling of the state variables for these algorithms. This could have prohibited most of the use of these tools on POWER4. For this reason, a novel state-folding algorithm was devised that allowed us to automatically eliminate one level of latches. Formal verification was thus able to process dramatically larger design partitions than would have been possible otherwise.

New verification tools

Simulators

Cycle-based simulation

The synchronous design style used by IBM, allowing the separation of timing verification from functional verification, evaluates the state of the logic at the end of each machine cycle. Such a zero-delay evaluation of the Boolean logic gates between state elements is organized in a rank-ordered fashion such that the overhead of event-driven algorithms is eliminated. This type of cycle-based simulation has the additional advantage that performance and memory requirements scale at most linearly with model size.

The principles behind the cycle-based simulator TexSim are described briefly in [4]. It is a compiled-code cycle simulator that is optimized to take advantage of the superscalar, pipelined nature of PowerPC machines. Boolean functions of the model were mapped to single-cycle Boolean instructions of the host processor.

The POWER4 project created several requirements that induced a number of innovations:

- *TexVHDL language processing:* The POWER4 project brought language-processing technology to a level that was able to support the complexity of the TexVHDL subset so that it could be processed into a form usable by cycle-based simulation. Mapping TexVHDL to a Boolean netlist level requires algorithms akin to logic synthesis, but the required turnaround of a full-chip compile for verification has to fall within a matter of minutes, which requires radically different implementations.
- *Parallel instance models:* The largest system that had to be simulated was larger than anything ever before simulated in RTL: a 32-way SMP, 16 instances of a 180-million-transistor chip plus its memory hierarchy and some I/O chips. We packed replicated units tightly by using the vectored, 32-bit word of the host processor for the evaluation of up to 32 separate instances of identical blocks. This optimization has the effect that for up to 32 "parallel instances" of the POWER4 processors, the model grows (and therefore the performance degrades) only sublinearly.
- *Multi-value cycle-based simulation:* Concluding that the multi-value evaluation of logic is orthogonal to the event-driven execution of HDL models, we added a multi-value (0, 1, x, z) feature to the second-generation TexSim, not only making power-on-reset (POR) simulation possible (the POWER4 model would have far exceeded the capacity of any event-driven simulator), but also bringing it to within about three to four times the speed of two-value-cycle-based simulation.

Verification acceleration

An important component of the massive verification "horsepower" needed to verify the POWER4 systems was a special-purpose hardware-verification accelerator called AWAN. It uses a massive network of Boolean-function processors, each of which is loaded with up to 128 000 logic instructions. Typically, each run through the sequence of all instructions in all logic processors constitutes one machine cycle, thus implementing the cycle-based simulation paradigm.

Throughput of the AWAN verification is limited by model load, setup, results analysis, and most significantly by the amount of interaction between the engine and the computing host. The importance of spending most of the run time in the engine at full speed limits the amount of interaction the verification control environment can have with the AWAN model. The raw-model performance of the POWER4 chip running on AWAN exceeds 2500 cycles per second, with the POWER4 chip containing 174 million transistors. For POWER4 the ideal application of AWAN was in pervasive verification (see the section on special verification topics). It was also used to run exerciser code that previously had been run in the hardware bringup laboratory.

Boolean-equivalence checking

For POWER4, traditional verification along with functional formal verification was used to validate the functional behavior of the RTL description of the design, expressed in VHDL, against a higher-level reference model (i.e., a set of correctness properties). This RTL description then became the reference model against which to verify the transistor-level model. The Boolean-equivalence checker Verity was used to ensure identical logical behavior between the two models. This comparison implicitly validated the transistor-level implementation with respect to all results obtained from the functional verification of the RTL.

The ability to verify the correct functional behavior of the pre-hardware design by simulating an RTL model and then using Boolean-equivalence checking to provide closure to the verification methodology has been a key part of IBM methodology for twenty years. Verity was the latest and best Boolean-equivalence checking tool within IBM [5, 6], but it stands on the shoulders of the tools that preceded it.

The ability of Verity to handle all of the circuit styles required to implement a 1GHz microprocessor was critical to the successful closure of the POWER4 verification methodology. The transistor circuit styles supported for the POWER4 included static, two-phase domino, pass-gate, pseudo-n-MOS, and various custom storage elements for register files. Verity verified the logical function of almost every transistor on the POWER4 chip, both custom and synthesized. The unverified parts were array cores and analog parts such as those used for chip I/O; verification was used to provide coverage on these components. Additionally, Verity performed consistency checks which verified that the rules governing a specific design style were observed. For example, "floating net" conditions were detected in which the logical state of a net would be undefined.

The use of Verity for the equivalence checking of POWER4 sequential designs requires identical state encoding for the circuit implementation and the RTL. The state-holding elements on both sides must be identified and put into one-to-one correlation. This is referred to as latch correspondence, and it was achieved by using a variety of techniques such as scan-chain traversal, naming conventions, and connectivity analysis. Automatic identification of latches in a flat transistor-level representation of the implementation was achieved by using pattern-matching techniques.

The task of verifying the entire POWER4 chip, encompassing the core, all units, and all macros, was accomplished by using a hierarchical approach. Verity has a well-defined hierarchical verification methodology in which a hierarchical partition of the CMOS implementation can be verified against a similar partition in the high-level hierarchical RTL specification. Verity can use and test functional boundary conditions expressed as logical constraints by the designers. Verity ensures that the constraints obeyed by a generating macro satisfy the constraints that a receiving macro expects. This was all brought together by a rigorous audit methodology, which ensured that all macros were verified using Verity, and the hierarchical verification methodology ensured that the functional boundary conditions were satisfied. In this manner, a bottom-up approach made the verification of the entire chip feasible. At the chip level, all that was left to be verified was the interconnect between the units and the core (a total of 250 000 comparisons) and the boundary conditions (2000 constraints). That verification goal was achieved in less than 24 hours on an RS/6000* Workstation Model 270 using 1.5 GB of memory. In total, 414 custom macros, 448 synthesized designs, eight units, the core, and the chip were verified.

Functional formal verification

Functional formal verification (FFV) is the process of proving the correct functionality of a model. There exist numerous FFV techniques ranging from theorem proving, in which a user manually guides a mechanized proof system to a desired conclusion, to automated techniques that exhaustively explore the behaviors of the design under test (DUT). We employed the latter exclusively for POWER4 because of their relative ease of use. The domain of application was to assess the correctness of the RTL implementation itself.

FFV requires three components: the DUT, a random "irritator" which provides the set of possible input stimuli to the DUT (i.e., it encodes the set of input assumptions of the DUT), and a set of correctness properties. Given these three components, the tool set automatically proves that the composition of the DUT and its irritator cannot violate the properties, or provides a simulation-like trace exhibiting a failure. In most cases, the property can be specified in the language of the DUT by the addition of checkstop logic, and the irritator can be expressed by nondeterministic HDL.

Note that unlike FFV, cycle simulation cannot prove the absence of design flaws, and cannot yield 100% state coverage on even medium-sized blocks. FFV provides exhaustive coverage implicitly, but the penalty of this exhaustive search is that FFV uses exponentially increasing resources (with respect to design size) to attempt to complete a proof. This typically limits application to the block level and mandates manual reduction work for larger blocks. Also, one must spend the effort required to build a block-level verification environment to enable FFV.

The tool that was used for FFV on POWER4 was RuleBase [7], developed by the IBM Haifa Research Laboratory as a user-friendly extension to an enhanced version of SMV (Symbolic Model Verifier) licensed from Carnegie Mellon University. It is a symbolic model checker, which exhaustively and symbolically enumerates the set of reachable states of the DUT. RuleBase also incorporates several reduction techniques which automatically reduce design size before SMV is called, alleviating the degree of exponential blowup. RuleBase utilizes a set of unique languages: EDL for writing the irritator, and Sugar for writing properties.

After discussions with design and verification project leaders, we initially identified several primary design components for targeted FFV. The choice was based on complexity, rate of change, difficulty of verification using traditional verification methods, and areas that have experienced late bugs on previous projects. These efforts were often very fruitful and yielded a significant number of extremely complex bugs in a fairly short period of time. Nevertheless, size barriers limited what could be accomplished with FFV, so traditional verification and FFV complemented each other well in the effort of attaining as high a degree of coverage as possible.

We applied FFV to some extent on approximately 40 design components throughout the processor and found more than 200 design flaws at various stages and of varying complexity. At least one bug was found by almost every application of FFV. In most cases, FFV began significantly later than verification. It is estimated that 15% of these bugs were of extreme complexity and would have been difficult for traditional verification. In some cases, a late bug found in verification or in the laboratory was recreated and its correction verified efficiently with FFV.

The application of FFV by the designers themselves, using the FFV team as consultants, tended to be very fruitful. For example, less time was spent transmitting design information between team members; therefore, FFV could readily be applied as soon as logic was available or modified, and the degree of coverage could be much greater because of the reduced potential for omitted rules or miscommunication. The process of formal specification alone tended to benefit the design components dramatically. A significant percentage of bugs found were merely cases the designer had overlooked. FFV benefits on POWER4 included finding many complex bugs early, enhancing the block-level specification, ensuring that a block was ready for the next level of simulation, and reproducing traditional simulation bugs and laboratory bugs.

Coverage

Coverage analysis was a monumental task in meeting our strategic verification goals, and coverage tools provided the leverage to tackle the mountains of data that were generated during verification.

The goal of coverage is to reduce the risk of a hardware design bug escaping detection in a verification environment. While verification teams have used basic coverage measurements for years, the POWER4 verification team actively sought state-of-the-art technology to use on this project. Progress is no longer measured with a single arbitrary gauge; instead we rely on an arsenal of tools to address the diverse needs of our multi-tiered effort. Coverage tools were at the foundation of our process and provided our verification teams with the feedback necessary to deliver quality products on time.

POWER4 verification relied primarily on four coverage tools: Bugspray, Comet, Abacus, and Covet. These tools supported a verification methodology that allowed us to specify events to be monitored during simulation run time. Each of the tools has provisions for collecting, storing, and reporting the data from simulation. They differ largely in the type of data they were designed to collect and in the scope of their application. Just as we have increased the number of teams in our verification hierarchy, we have also increased the use of coverage tools. We now have pervasive use of coverage tools in all levels of verification, including unit, multi-unit, chip, and system simulation.

Bugspray provides an extension to the VHDL language which allows logic designers to add special macros directly into the design. Bugspray macros are excellent for targeting microarchitecture events such as state-machine transitions; more significantly, they enable the engineers most familiar with the design to define coverage events and to direct coverage to areas they believe to be most susceptible to bugs. Bugspray events help the verification team identify areas of weakness more quickly and easily. The events can be one of three types. They can pinpoint events that should never be seen, enumerate difficult, hard-to-create scenarios, or identify unique scenarios that are worth saving.

Bugspray was most useful at the lower levels of verification for two reasons. First, the events monitored by Bugspray are typically easier to encounter in these environments. They generally have fewer timing inhibitors on primary inputs because artificial stimuli are used instead of the RTL model to drive those inputs. Second, the additional Bugspray macros have a negative impact on both model size and speed, both already at a premium in the higher levels of verification.

Comet is a general-purpose coverage tool that can be used throughout the verification hierarchy. Comet tracers can parse important microarchitecture events such as dispatch and completion times and the number of rename registers in use. Comet tracers can also parse protocol events such as the number of times an intervention response appears on the POWER4 fabric bus and the number of transitions between L2 MESI[1] states. Finally, Comet can be used for the simpler task of counting the number of signal transitions coming from a chip or card I/O.

The use of Comet was targeted to the upper levels of the verification spectrum, from core to chip, and from *N*-way to system simulation. The strengths of Comet were most apparent in the analysis of timing relationships between events. The analysis is actually handled outside the model under test and also outside the coverage model itself. One must first conceive of a trace analyzer that is powerful enough to establish relationships among what would otherwise be considered unrelated events. The trace analyzer filters the data, logging interesting events or scenarios, and then passes that information into Comet, where it is added to a database of similar occurrences. Comet coverage models are then prepared which take the collection of logged events and prepare an analysis of all related coverage events. The end result is a list of interesting coverage events that have not been previously encountered.

Abacus, as its name implies, was originally a simple set of counters used for delineating features of the PowerPC architecture. Abacus was created several years ago to help us gauge the effectiveness of our test cases. Abacus simply parses a test case for its only input. The test case is broken down into architecturally significant segments which include instruction-level monitors for counting the number of times an instruction is executed, a particular field is set to a specific value, a particular instruction produces an interrupt, etc.

Abacus remains a staple of coverage at the processor core and chip level of verification for analyzing the features of the PowerPC architecture, an application which it performs exceptionally well. The verification team used the reports from Abacus to identify test-case deficiencies and weaknesses in the test-case generators. One might say that Abacus supplied us with a "report card" on the completeness of our architectural test-case suite.

What Abacus provides for PowerPC processor architecture, Covet provides for the rest of the system architecture. Covet is a special-purpose coverage tool that perfectly complements the bus-protocol checkers that were used to verify the operation of the storage and I/O subsystems. It can analyze and report on a series of bus transactions, looking for specific events such as race conditions presented at the interface of a coherence unit. Covet's coverage model monitors the occurrence of concurrent events that stress a unit. For example, Covet will analyze the many address collisions in the system at

[1] The IBM "MESI" cache consistency protocol specifies four possible cache-data states: modified, exclusive, shared, or invalid.

the interface of each cache controller to ensure that coherence and data integrity are maintained.

Covet was used exclusively by *N*-way and system verification. In many respects, Covet and Comet were similar in scope and purpose. Whereas Comet is a general-purpose coverage tool using any data as input, Covet is much better suited for use with trace data generated for Coherency Monitor Lite (CML), a bus monitor that checks for coherency violations anywhere in the system. Because of Covet's close ties to CML, it is perfect for tracking and analyzing coherency events in the large *N*-way models. The verification teams use the Covet coverage reports to target additional areas of testing that may be naturally hard to come by. Covet gave us the advantage of monitoring how well we were stressing the system at the bus level.

Designer- and block-level verification

Because the POWER4 design, from a microarchitectural standpoint, was significantly different from its predecessors, there was significant opportunity to extract many bugs from the logic at the block level early in the development cycle. Verification at the block level allows not only speedier run times because of the small model sizes, but also a more stressful environment for a given block under test than a larger chip model might provide. With very little effort, the block simulator can describe relatively complex interactions of low-level events that can stress the window conditions of a design. Trying to identify these same events with an architectural test-case generator can be extraordinarily difficult.

There were three primary tool sets or methodologies used to perform block-level verification for the POWER4 design. These were the following:

- TIMEDIAG/GENRAND.
- C/C++ interface drivers and checkers.
- Functional formal verification.

TIMEDIAG/GENRAND enabled the user to create fairly complex and stressful environments for the block under test. TIMEDIAG is a graphics-based tool that allows the engineer to define generic interface-protocol timing diagrams. The timing diagrams define potential scenarios or actions on the design interfaces and specify the expected behavior of the design. (Looping conditions, random values, complex expressions, and start-up specifications were all supported by the tool.) These diagrams can then be used individually or in various combinations by the GENRAND tool to create reasonably complex scenarios for the design under test. This verification methodology was particularly attractive to a portion of the design community that did not carry C/C++ programming as part of their skill set. The graphical waveform capture of TIMEDIAG allowed this group to define test cases of varying complexity in a very intuitive way.

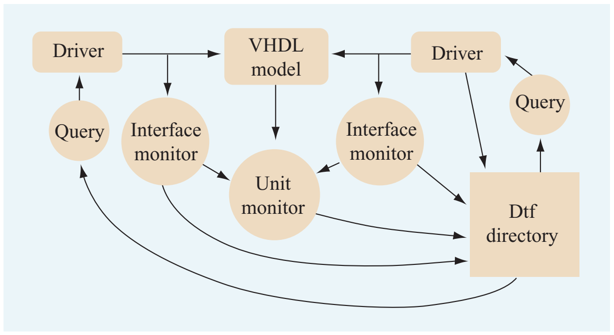


Figure 3

The random-command-driven verification environment used in GPS-unit and multi-unit verification.

C/C++ code was generated to irritate interfaces and check results of the design under test. Some of this code utilized random pattern-generation techniques to improve overall coverage. Checking code was written to monitor both outputs and internal functions of the RTL model. The checking code was written in a modular format to allow the checking to migrate up into the larger verification models. Again, the size of the design under test at the block level allowed quick turns of the models and led to the overall efficiencies of "wringing out" bugs at the lowest level of the simulation model hierarchy.

Unit-level verification

Unit and multi-unit verification using random-command-driven methodology

The unit level is viewed by some as a "sweet spot" in the verification hierarchy. It provides significant return in terms of quantity and quality of bugs removed. Unit-level verification was performed on functional units, such as the instruction decode unit, the L1 data cache, and the L2 cache.

There were definitely tradeoffs regarding the extent to which every functional unit had to be verified in a standalone unit-verification environment. A considerable investment in resources was required in order to design each unit-verification environment. A typical unit environment using the random-command-driven (RCD) methodology, as shown in **Figure 3**, consists of one or more interface drivers, interface monitors, and unit monitor checkers within the RTL model, and of course a significant amount of C/C++ RTX code which serves as the central command center and directs cycle-by-cycle command changes and monitors for test-case failures.

Driver, unit monitor, interface monitor, and checker

The following is a description of the RCD methodology that was used on most units and used exclusively on POWER4 storage subsystem (GPS) multi-unit verification. When performing unit verification, we needed a means for driving the interfaces of the design under test (DUT) instead of relying on actual RTL from interconnected units. Two methods were used to generate the necessary stimuli:

- Start with a test case that is generated by hand or use a test-case generator to generate test cases. A program extracts the transactions that affect the unit. The driver uses this information to provide the necessary stimuli to stress the unit. This approach was typically used on units that were close to the core complex.
- A driver under the influence of a parameter file and a random-number generator initiates random transactions to the DUT. The drivers are not restricted by what the real unit RTL would normally do, but by what the interface protocol allows. This has the advantage that it is easier to write and maintain. In addition, the driver can provide some unique sequences that the real hardware may exercise only under some extremely rare condition. Since the driver can create the transactions "on the fly," it can more easily react to the current state of the unit under test and drive the unit to some complex "corner" case. In contrast to the pre-generated test patterns, the dynamic test generator has *a priori* knowledge of the unit state space before it generates the test case for the next verification cycle. This approach was more appropriate for testing storage units, since these units typically have straightforward transaction requirements.

To ensure that the DUT behaves according to specification, we used interface monitors and a unit checker. An interface monitor verifies a particular interface of the unit and is also responsible for ensuring that the interface protocol is correct. The unit checker checks for proper unit behavior across all interfaces and within the unit. It also updates the reference state of the unit as a result of the unit's responses to the stimuli. This reference state can be used by the drivers to enhance stimuli sent to the unit in order to arrive at some hard-to-reach corner case. The checking is done in real time, cycle by cycle. Since checking is made against some ground rules rather than by predicting results, checking is straightforward. When a discrepancy is detected, the monitor flags the error, prints out pertinent information concerning the error, and terminates the test case. This method of verification increases the run-time efficiency of the environment by minimizing the number of cycles run between a failure and its detection. Furthermore, it was useful to totally isolate the driver from the interface monitor and checker/monitor combination. In this way, we were able to move the interface monitors and checkers up to the next level of the verification hierarchy to help detect and isolate design defects faster at run time.

Unit and multi-unit verification used an internally developed software library that supplied a foundation for writing the driver, unit monitor, interface monitor, and checker. The library support included

- Hierarchical facility management.
- A synchronization point to read/write the hardware facilities (signals).
- Error message handling, such as temporarily ignoring a fail error message for a known failure condition in order to allow further testing.
- Random-number and parameter-biasing management.

The library allowed the verification engineers to concentrate on their primary job of verifying the DUT, rather than concerning themselves with low-level simulator application interfaces and file-management chores.

RCD verification of the cache, fabric, memory, and I/O

The storage subsystem (GPS, consisting of cache, fabric, memory, and I/O) verification was accomplished by applying the RCD methodology. A model of a typical unit-verification environment in the GPS subsystem is shown in Figure 3.

Before the start of any simulation run, a well-thought-out address pool was generated. This was critical, as a truly random address generator would not stress the cache design well. We needed to select addresses according to the design of the particular cache that we exercised (e.g., to target a specific number of congruence classes and banks). Once a set of addresses was established, all transactions used those addresses throughout the simulation run. This helped ensure high resource contention.

In addition, cache preloading was used to stress address and cache-line traffic and contention. By applying cache preloading, some of the more interesting areas of the GPS state space were reached without using a significant amount of simulation time to initialize the caches from the power-on system state. The advantages included shorter test cases and assistance in targeting specific conditions (such as causing castouts from a cache more readily). Two methods were used to generate cache preloads: handwritten sequences and random generation. Handwritten sequences were used in performance-verification tests to generate preloads for the specific address sequences used in that test.

Random generation, the dominant method used in unit verification, used a Monte Carlo progressive-constraintbased system. Each type of preload (combination of cache states and levels of caches) was enumerated. For each enumeration, constraints on that preload were formed. Constraints can include physical congruence-class requirements and cache-coherency rules. The following process was then evaluated at the start of the test case. For each address to be preloaded, a preload depth was chosen for the maximum number of preloads for that address. Successive valid random preloads were chosen until the desired depth was reached or no more preloads were valid (based on the constraints). For each test, the bias for each preload state was chosen randomly. This "fixed for each test" bias gave greater variation in preload ratios. Additionally, both valid and invalid preload data were allowed. For invalid cases, the data was noncoherent random data.

While random transactions were being entered into the model, a detailed trace of transactions and reactions to those transactions was logged in an activity file. This log file proved to be indispensable in isolating and debugging failures. Detailed facility trace data during the failure time was provided to the designers to help locate the design fault. By using this log file, the majority of design fixes could be corrected and verified within one day.

Dynamic test-case generation

The dynamic test-case framework (Dtf) is a C++-based library for the generation of model-directed random dynamic test cases. It is the most advanced application of the RCD verification methodology applied at the unit level in POWER4 and was used throughout the GPS multi-unit environment. Its primary purpose is to use the information gathered in monitoring the validity of the test to assist in randomly driving that test to expose bugs in the logic faster and more often. To provide and control this means of information passing, Dtf provides a common interface for storing and querying state information and a common interface for using that information to run the test case.

The conduit for monitor feedback travels through the Dtf directory and queries. The Dtf directory is a class for storing and organizing state information. A particular entry in that directory can be associated with an arbitrary number of states. A Dtf directory query can process any first-order Boolean combination of states across any of the existing Dtf directories.

The Dtf command selector provides a common interface for the software drivers which use information from Dtf directory queries in generating pseudorandom behavior. Each type of command is encapsulated in a Dtf command "factory," which generates everything needed to execute a specific command of that type. The selector chooses randomly from the command factories in that driver according to specified bias weights. That selection is done with a Poisson distribution, which gives a more realistic pattern of event arrivals than a uniform distribution. In addition to relative weights, the selection of a command is influenced by a set of predicate functions associated with that command. These predicates both indicate when the command can be executed and provide information to the command from the applicable directory queries.

This selection process was a rather simple model for selection. Frequently, however, more interesting models were desirable. This need led to the development of the Dtf command sequence, which allowed the use of an arbitrary hierarchical Markov model to control the selection of commands. Typically, a command sequence would be used to model a situation that, in the past, had led to troublesome behavior. Through the use of predicates that use directory queries related to the activity of other drivers, a command sequence can even be used to coordinate complex behavior among multiple drivers.

In addition to its use in GPS, the Dtf library was used to design a new unit simulation environment in the core of POWER4. The previous simulation environment had allowed several bugs to pass that were later caught in the laboratory. The environment included two major drivers, one for the ISU (instruction sequence unit)-to-IFU interface and one for the CIU (cache interface unit)-to-IFU interface. For those pieces, we wrote new drivers that used Dtf to bias the control of those operations. These new drivers paid immediate dividends, finding several bugs in the hardware.

Sunfish and Outrigger unit verification

Sunfish, an external bridge I/O chip, and Outrigger, an external memory controller chip, were verified by the POWER4 team and were treated much as another unit of the POWER4 chip. At the unit level, Sunfish and Outrigger verification used the RCD methodology, as shown in Figure 2, as well as a significant amount of deterministic testing. The deterministic test cases, called unit verification programs (UVPs), were standalone test cases with specific register-setting values. In the deterministic testing mode, the UVP drivers replaced the random drivers in the model and were used to test specific boundary conditions, recreate failures found by other methods, and test specific conditions that were difficult to hit with RCD. In addition, an orthogonal method was applied in both the deterministic and random methods. The concept of this method originated from [8], but all variables (commands) were allowed to vary under the control of the orthogonal matrix, instead of varying only one variable at a time while holding all others constant. Although this method was not exhaustive in testing all combinations, it provided the best coverage for a given amount of resource.

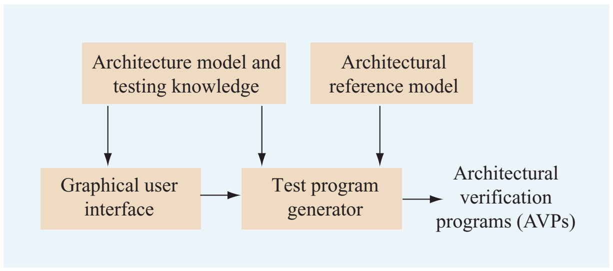

Architecture model and testing knowledge
Architectural reference model
Graphical user interface
Test program generator
Architectural verification programs (AVPs)

Figure 4

Genesys/Genie/GenesisPro pseudorandom-instruction-level test-case generators comprising four elements.

Multi-unit and chip verification

Genie/Genesys/GenesysPro pseudorandom test-case generators

The processor core (functional units and L1 cache) multi-unit-level and chip-level (incorporating both the core and GPS) verification strategy for POWER4 was based on architectural verification programs (AVPs). The AVPs were generated by model-based test generators (Genesys/Genie/GenesysPro) according to a comprehensive plan which covered all of the PowerPC architecture specification. Major components [9] of these generators are depicted in **Figure 4**.

Genesys

Genesys provides an efficient method to generate PowerPC AVPs. It generates pseudorandom AVPs according to its user's directives. Missing or incomplete directives were completed randomly by the tool, but the random selections were biased toward the generation of interesting cases. The random capabilities of Genesys assisted in finding design defects beyond the limited scope of manually written AVPs.

Genesys allows its user to select the instructions which appear in the AVP, and their order. For each instruction, both syntactic and semantic selections are available; the syntactic selections assign values to the instruction fields, while the semantic selections dictate the occurrence of events (via biased generation functions). The whole generation process is run under the control of many global parameters, which may be changed or overridden by more specific directives. The user can set initial values for the design resources (such as registers and memory).

Genie

Genie is a multiprocessor (MP) version of Genesys capable of generating false-sharing and true-sharing test cases with either deterministic (both unique and nonunique) results or nondeterministic results that can be checked by a run-time coherency monitor/protocol checker (CML).

GenesysPro

GenesysPro incorporates the learning of almost ten years of pseudorandom instruction-level test-case generator development into an entirely new tool. It is intended for both uniprocessor and multiprocessor verification. While the underlying principles and structure of the Genesys model-based test generator are unchanged, many new features have been implemented which allow the users even more powerful methods of writing directed test cases. This is necessary to keep up with the ever-increasing sophistication of microarchitectures such as the POWER4.

Since GenesysPro was in the experimental stages of development for the POWER4 project, its usage was limited to a handful of advanced test-case scenarios that were difficult or impossible to accomplish with Genesys or Genie.

Static and random test-case methodology at the processor core multi-unit level

At the uniprocessor core level, two major verification phases were defined: static and random. Both phases relied almost exclusively on test-case generators for the creation of AVPs.

For the static phase, libraries of AVPs were developed which provided broad coverage of both the architecture and the implementation-specific features of POWER4. A partial set of these tests (approximately 150 000) was used to regress each model to assess the quality of the VHDL delivered. This subset would run overnight on the simulation farm. The complete static regression consisted of approximately 500 000 tests and was run against major VHDL deliveries. It required about three days of run time on the simulation farm to complete.

In general, the complete regression "bucket" consisted of different types of test cases. These were run both with and without random external interrupts:

- StarterSet—one instance of the same instruction per test.
- JumpStarterSet—10, 50, or 100 instructions of the same type per test.
- Complex Every Others (CEOs)—All possible pairs of instructions (for those that can be generated).
- IVPs to cover events that were not targeted by the above tests. IVPs (implementation verification programs) were implementation-specific tests such as specific sequences that AIX* developers identified as interesting and sequences that were identified by lead architects and designers.
- Tests that found previous bugs.

For the random phase, a degree of randomness in the definitions ("defs") was necessary to guarantee their quality; they had to be specific enough to target the specified tasks, but random enough that different AVPs were generated each time. The nature of the random tests varied considerably from def file to def file. Short, single-instruction AVPs were used initially, followed by increasingly more complicated mixing of random instructions. In general, the AVPs were developed to stress the interaction of the various units on the processor core.

The above tests were run on both core- and uniprocessor-level models (see below). The core model allowed for faster simulation performance than the chip model, since it used a C++ behavior to represent the storage-control element instead of the actual GPS VHDL. The added benefit of this behavior was that it allowed for more robust stimuli to be applied to the portions of the core that interfaced with the GPS, i.e., the IFU and LSU.

UNI and dyadic chip-level verification

Two models were utilized to stress the interaction of the processor core and the GPS:

- UNI model: Combined the GPS and a single processor core.
- Dyadic model: Combined both processor cores with the GPS and chip pervasive logic. This model contained the complete POWER4 VHDL.

This section describes the major verification challenges and methodologies which were utilized in these two models to verify the architecture and the implementation.

For the UNI model, the methodology was similar to the core verification effort. In general, many of the same tests that exercised the core model with the L2 cache behavioral model (C++) were repeated on the UNI model. This accomplished two main goals.

First, it allowed the verification team to verify that the core VHDL actually performed correctly with the GPS VHDL under a wide array of tests and stimuli. This step was necessary to validate what was done on the core model using only the L2 cache behavioral. While the L2 behavioral allows for more robust irritation than is possible with the real GPS VHDL, it was possible that there were subtle differences which would have to be accounted for by repeating the tests on the UNI model. This also presented a wide variety of stimuli to the GPS VHDL from the core for the first time.

Second, with the incorporation of the real GPS VHDL into the model, the verification team could make use of Coherency Monitor Lite (CML). CML is valuable in verifying certain "ugly ops" [3], or instructions which would otherwise be difficult to verify; examples of this include tlbie, tlbiel, sync, lwsync, eieio, and several cache instructions such as dcbt, dcbtst, dcbf, and icbi. Verifying that these instructions have been properly executed by the RTL model can be difficult using solely AVPs and RTX checkers. An example of a bug that was detected on the UNI model by CML was the case of a noncacheable load that was sent out twice to the GPS by the core. Because of the manner in which the PowerPC architecture uses noncacheable memory pages for interaction with I/O devices, this is not allowed. No other checkers in the environment detected this error.

On the dyadic model, special focus was applied to multiprocessor (MP) test scenarios. The primary tool used for stimulus on the dyadic model was Genie. All tests run on the dyadic model were checked by CML for MP problems and protocol violations. Some tests were deterministic and relied on CML as an additional means of checking the results, while other tests were nondeterministic and relied completely on CML for determining the correctness of the results.

In particular, bugs that were found on the dyadic model related to the reservation mechanism by employing code that mimics actual "test and set" operations as they were performed by an operating system or applications. In addition, CML detected consistency bugs where two processors were reading and writing partially overlapping bytes of memory simultaneously but not returning a consistent result. The incorporation of CML into the UNI and dyadic models is a direct result of the escape analysis of the POWER3 processor in which such bugs were found only by CML in system simulation or by the hardware testing on the test floor. This demonstrates the continual evolutionary nature of processor-verification techniques and the value of analyzing previous escapes to laboratory and higher levels of simulation. Whereas the most common types of escapes to the hardware laboratory on POWER3 were related to load ordering (mostly consistency), not a single load-ordering problem was found in the laboratory on POWER4 hardware. The GenesysPro tool was used on the dyadic model to verify the CMODX-type scenarios described previously. While this tool was experimental for the POWER4 program, it allowed the verification team to recreate problems previously seen only in the laboratory and to verify fixes. Furthermore, it allowed the verification team to execute a directed test plan to cover other related scenarios and increased the level of confidence that time-to-market goals would be achieved.

Regatta[2] system verification

System simulation is the highest hierarchical level of verification. The main objective for system simulation is to

[2] Regatta is the code name given to the MP system comprising the POWER4 along with all other system chips.

verify interactions among chips using actual chip VHDL for the processor, memory, and I/O chips. Building a 32-way POWER4 system using VHDL for a full 32-way system is neither practical nor the best approach to verify a system. The challenge for system simulation was to come up with several different, smaller configuration models that effectively represented the 32-way structure without the full 32-way model. A 32-way system consists of four multichip modules (MCMs) connected using the MCM-to-MCM bus structure. Each MCM contains four POWER4 chips interconnected using a chip-to-chip bus structure. In addition, each POWER4 chip has memory and I/O buses providing connections to L3 cache, memory controller, and I/O chips. Three categories of models were used:

- Chip-to-chip models based on one MCM, with and without I/O.
- MCM-to-MCM models, with and without I/O.
- Combination models, using both chip-to-chip and MCM-to-MCM models, with and without I/O.

Two eight-way models, one chip-to-chip and one MCM-to-MCM, were the workhorses of system simulation. Most of the functional logic bugs were found using these eight-way models. Without using parallel-instance models, the system-model sizes would have been too large to run in our batch system simulation environment. Only limited testing was done on the larger models because the model environment was too big to provide the testing level achieved using the eight-way model. The system-verification environment used the same RTX checkers as those written for lower hierarchical levels. On the models which included the I/O chips, HDL PCI behaviorals were developed and compiled into the models to create I/O traffic. Additional code was developed to support the I/O chips at the system level.

Regatta system-verification test generation and checking

Two internally developed test-case generators were used by system simulation—MultiProcessor Test Generator (MPTG) and Genie/DmaInjector. A third test-case generator, SysGen, was used to a much lesser extent. MPTG and Genie were both used for *N*-way without I/O testing. All three generators were used for I/O testing.

MPTG is a system-based test-case-generation program, designed to generate comprehensive, system-level test cases for the memory hierarchy of MP systems. MPTG generates test cases that focus on causing interactions among all of the chips (including I/O) in the system. The MPTG test-case generator has two separate parts: a generic test-generation engine and a system-specific machine model. The machine model includes protocol tables and preload rules for all caches, and thus provides the MPTG tool with detailed knowledge of the memory hierarchy of the system under test. The user can create test cases that precisely target certain cache-coherency scenarios. This is done in MPTG by using load/store combinations with various address streams to address the cache directories.

MPTG test specifications are created and written through the MPTG graphical user interface. There were mechanisms to provide precise control over MPTG to create tests with varying degrees of deterministic and random events. The test specification is loaded into the MPTG generic test-generation engine and applied to the system-specific machine model to generate a "bucket" of test cases. The test specification can exploit this model to enumerate tests that cover various combinations of parameters in the protocol tables of the machine model. Because of the nature of true sharing, which was the focus of MPTG test cases mentioned in the beginning, MPTG cannot predict the final results of testing, and MPTG tests must be dynamically checked by using CML.

CML detects the types of bugs that typically arise in the interaction of components of a complex MP system. Types of rules verified by CML include MP cache coherence, memory and cache consistency, synchronization, reservations, external interrupts, and ordering and propagation of I/O traffic at several levels of the I/O hierarchy. CML consists of three components—tracers, checkers, and the plotter:

- Tracers are software state machines that monitor various interfaces or components of the system during simulation and produce trace files of all relevant traffic.
- Checkers: CML contains hundreds of checkers, most of which are based on architectural rules. The checkers run after simulation has completed.
- Plotter: The CML plotter is a debug tool that provides a device-vs.-cycle-time plot of all system activity captured by the tracers.

Regatta **N*-way system simulation testing***

N-way testing refers to Regatta MP models that do not include I/O chips. A large number of MPTG tests written for previous PowerPC *N*-way systems were ported so that they could be used to verify the Regatta system. In addition, a large number of new MPTG tests were created specifically to verify the Regatta system. The POWER4 microarchitecture includes a new L2 cache design with three slices, a new L3 cache design with shared and private modes, and a new interprocessor connection architecture with both chip-to-chip connections and MCM-to-MCM connections. Previous MP systems usually had all of the memory behind one (or two) memory controller(s), with the individual processors connected on a common system bus. For the Regatta system, a memory controller can be attached to each POWER4 chip.

The basic philosophy for verifying an MP system is to have two or more processors access the same memory word or cache line during a test. For the Regatta *N*-way systems, tests were created which selected the memory addresses in a controlled manner so that all interesting combinations of processors and targeted memory controllers could be exercised. Processor transactions were chosen randomly, based on weights in MPTG test cases. Specific tests had to be written for each system configuration, since the mapping of address to memory varied depending on whether the L3s behind each chip were in shared or private mode and whether there was a single MCM or multiple MCMs in the system model.

Another challenge for Regatta verification was related to the new L2 architecture, which had a six-deep store queue on each of three L2 slices. Two processor cores share the same L2 cache. For stressing the L2 store queues, addresses were selected to target stores to the lower or upper 64 bytes of targeted cache lines on selected L2 slices. The tests were created to use byte, halfword, word, doubleword, quadword, or random combinations of all sizes of stores only, or stores in random combination with other transaction types.

The Regatta architecture defined a new data-prefetching mechanism in which core accesses to sequential cache lines are detected and used to cause core data prefetches. These data prefetches are converted into L2 data prefetches and L3 data prefetches. A series of new tests were written to cause data prefetches to be generated in both forward and backward directions using the dcbt and dcbtst instructions, with a flag indicating data prefetching in a forward or backward direction.

The Outrigger Memory Controller was new for the Regatta system, and tests were written for the system environment to target it and the L3 controller located in front of it. The memory controller, for performance purposes, maintains a "partial line" store queue which allows multiple processor or I/O writes to a cache line to be "accumulated" before writing to the memory dimms. To support ECC, the data must be read from the memory dimms, overlaid with the "new" partial write data, and then written back to the dimms with newly generated ECC. This made partial writes slow, since both a read and a write memory access were required. Tests were written to stress the memory and L3 controller by varying the number of processors sharing cacheable and noncacheable shared addresses, the number of L3s and L3 slices to which the addresses belonged, and the weighting of stores versus loads.

Covet was used to evaluate whether the tests were examining all of the desired sequences. New tests were written to hit certain scenarios which were not exercised by the initial tests. One of the lessons learned from experience with testing the Regatta system is that with such a complex and large model, it was difficult to run test cases long enough to examine everything. It took a long time to cause L3 castouts, and it was difficult to align castouts with other accesses to the same 512-byte region. To facilitate verification and cause L2 castouts, the L2 cache configuration was changed from eight-way set-associative to direct-mapped for certain tests.

Regatta I/O system simulation testing

The I/O system model consisted of the *N*-way model plus the actual I/O chip VHDL. Key simulation challenges included the numerous system configurations, ensuring the compatibility of I/O chips, and interrupt structure changes.

The I/O system structure consisted of a POWER4 processor and an I/O bus called GX, each of which can connect to the Sunfish I/O chip. The Sunfish supports up to four intermediate I/O buses called RIOs, which connect to a RIO-to-PCI bridge chip called Speedwagon. PCI behaviorals (called Monty) were added in a nonsymmetrical manner to stress different PCI bus-loading characteristics.

Test-case generation was provided by the same two tools used in the *N*-way system testing and SysGen. MPTG, described previously, has knowledge of all processors, system memory, I/O behavioral commands, and memory-mapped I/O, including processor MMIOs (processor accesses to I/O addresses), Monty DMAs (PCI accesses to main memory), and external interrupts from Monty. The test cases contain memory initialization data, processor instruction sequences, and Monty command sequences for DMAs and interrupts. Unlike the other generators, MPTG also use TCE (Translation Control Entry, used in I/O address translation) tables, which is a key component of I/O testing.

DmaInjector could read Genie test cases and create an internal image of memory, which it then used as the target memory range for DMA. By using unused bytes in cache lines accessed by the processor, DmaInjector also generated expected results for its DMA transactions. The DMA transactions also caused cache-line thrashing between processors and I/O devices.

SysGen was used primarily to validate AS/400* features of the I/O structure.

The external interrupt testing for Regatta focused on the new Sunfish interrupt controller. The Sunfish chip introduced a new interrupt bus protocol that was based on PowerPC interrupts. One of the Sunfish chips was configured to be the "primary" Sunfish and acted as the interrupt bus arbiter and interrupt presenter. The other Sunfish chips were configured to act as routers to the primary Sunfish. Test plans and configurations were managed to cover the various interrupt modes in the new protocol.

The external interrupts were generated through the PCI behaviorals and sent to the interrupt presenter, and the interrupted processor sent configuration stores out to I/O to turn off the interrupt. The interrupt handler was designed to work with defined bus unit identifiers (BUIDs) that closely represented the anticipated product-level initialization. The handler read the external interrupt request register (XIRR) from the interrupt presenter and used the BUID information gathered there to determine the activity required to turn off the interrupt (such as configuration stores to I/O). The end of interrupt (EOI) came when the handler wrote back to the XIRR. The interrupt handler also supported nesting of interrupts. I/O interrupts were set up with a variety of priorities. This nesting required the interrupt presenter to manage multiple outstanding interrupts to the same processor. Interprocessor interrupts (IPIs) were also supported. Within the interrupt handler, IPIs were treated similarly to external interrupts minus the configuration stores to I/O. The test-case generator(s) were designed to allow for IPIs and external interrupts. The interrupt-handler code was included in these tests. A variety of test cases were generated with these interrupts. Other I/O activity (MMIOs and DMAs) also contributed to stress the Sunfish I/O bridge in conjunction with the interrupts. Various external interrupts were set up both as directed (an interrupt sent to a specific processor) and as global (an interrupt sent with a specific server number that the interrupt presenter decodes to mean one of several processors which the presenter is allowed to choose).

The interrupts were checked using two methods. Coherency Monitor Lite (CML) checked that the nested interrupts occurred with the correct priorities. CML also checked that the global interrupts (defined with a specific server number) were directed to the correct processors. This verified the LPAR portion of the interrupt protocol. Additionally, CML checked basic interrupt functionality such as ordering protocols (one example is that an EOI must not pass an MMIO store to a device) and made sure that every I/O interrupt was matched with an interrupt acknowledgment and an EOI. The second method for checking interrupts was done by using individual processor counters for the interrupts. This method worked well to identify missing, extra, or misdirected interrupts.

Special verification topics

Pervasive RAS/debug

In this paper, the term *pervasive verification* refers to a plethora of essential items in the POWER4 chip that provide reliability, availability, and serviceability (RAS) functionality. Typical features include scanning the state of the processor, array built-in self-test (ABIST), logic built-in self-test (LBIST), array display/alter, interface alignment procedure testing, I/O shorts tests, internal serial communication port (SCOM) (JTAG), debug (trace array logging, trace stop on error, and trace stop on trigger), and error injection/detection/correction/reporting. While these features provide customer RAS capability, they are also invaluable features for use in debugging problems in the laboratory. Most debugging occurs without logic analyzers.

Most of the POWER4 pervasive verification was implemented at the chip and system levels. The chip model was used for most pervasive features because of its simulation speed, while the system model focused on validating initialization and debug functions. The system models closely resembled the bringup configuration used in the laboratory. The models were built using real card and board data to validate chip-to-chip interconnections and proper chip-input pullup and pulldown resistors. Both the AWAN and MVLSIM simulators were used.

The service processor in the system controls the initialization and performs debug procedures through a serial interface (JTAG) to each chip in the system. A service processor command library was developed which was used by both simulation and the bringup laboratory. Test cases used the library and were common between simulation and the laboratory. A number of significant achievements were realized as a result of this strategy. A complete POR using full scan sequences was executed successfully before the chip was taped out, simulation was able to match LBIST signatures with the real hardware, full ABIST ran to completion using fault injection and fuse repair, and a complete set of tests were developed and run in simulation that could be run out of L2 to help the sort at wafer test. All of these significantly reduced the time spent on chip bringup in the laboratory.

Performance verification

Many sophisticated microarchitecture features of POWER4 exist in the design. Simply verifying that these features operate from a functional perspective, however, is inadequate. The performance of the total design must be verified. As the microarchitecture was developed, various high-level performance models (see the overview section) were used to enable design tradeoffs. These high-level features could be latency projections on paper or complicated cycle-approximate models of all of the important microarchitectural features. The goal of performance verification was to verify that the high-level models matched the implementation, thus giving confidence to performance projections based on those models.

Several methods of simulation were used to verify individual components of the system as well as the entire system. The system was broken into two main portions: the processor core and GPS storage subsystem. Specific tests were developed for each model. These tests included very specific sequences, synthetic instruction sequences, and application code segments.

Verification of the processor core itself was based mainly on correlation of specific instruction sequences between the M1 and M3 models. The same instruction sequence would be executed on both models. The timing of specific events would then be compared between the two models. Furthermore, the actual run time of specific instruction sequences was verified. One of the most difficult aspects in model correlation was the use of free-running counters that controlled resource allocation. If these counters did not match exactly between the models, stall conditions due to resource allocation would come out of sync. This would cause a rapid degradation in the event correlation and make valid comparisons impossible.

For verifying storage subsystem performance, we found it most useful to drive the processor interfaces by a variety of methods. The most simple method was based on command text files, which provided a method to verify latencies and bandwidths throughout the system. Design features, such as arbitration mechanisms, and overall latencies were tested by these methods. To complete the feedback system with the core, a C++-based processor behavioral was used that provided only the core functionality needed. This behavioral provided an ideal system performance assuming a perfect processor core. The feedback system was needed for streaming-data technical workloads. In these workloads, the rate at which data is returned determines the requesting rate. This simple core behavioral allowed the isolation of design performance issues to the storage subsystem.

Use of the above techniques enabled significant improvements in both implementation performance and modeling accuracy. On the basis of design defects found and design changes achieved through more accurate modeling and projections, these tests encouraged significant improvements in performance—in some cases as high as 100% for large-memory daxpy (a technical workload from the LINPACK suite of benchmarks). The performance findings also improved sustained L2 bandwidth by 30% by identifying a bug caused by an incorrect signal used for pacing.

Asynchronous interface verification

As core processor frequencies continue to increase faster than DRAM and I/O technology, designers look to asynchronous interfaces between memory and I/O to communicate independently of the processor clock frequency. Asynchronous interfaces can be categorized into two general categories—pseudo and true asynchronous. Pseudo asynchronous interfaces are the most prevalent; these are defined by two logical blocks clocked at integer multiples of each other. POWER4 uses this type of interface in several areas. For example, the L3 cache, FBC, and GX operate at $1/n$ (where $n = 1, 2, 3, 4$, etc.) of the processor frequency. True asynchronous interfaces must operate correctly on noninteger multiples. The memory controller (Outrigger) uses such an interface so that the memory subsystem can operate at optimal frequencies that are independent of the slower DDR memory access times.

The goal of verification was to prevent hardware asynchronous interface escapes by using both model checking and simulation techniques. Real hardware interfaces were subject to jitter, meta-stability, and combinatorial logic switching. Current hardware simulators, both cycle-based and event-driven, do not correctly model the unpredictable behavior seen in the laboratory. Additionally, timing analysis across such interfaces cannot prove the correctness of the design. Previously, the primary method of verification was manual inspection of the design language. This has proven to be susceptible to omissions in asynchronous crossings and difficulties in accurately mapping the design language into circuit diagrams. This problem becomes more complex as designers use prepackaged libraries.

The first step in verifying asynchronous interfaces was to identify all instances in which a signal crosses from one clock domain to another clock domain. This was done by monitoring the "rate" at which a latch output would change relative to the other latches. This can uniquely identify the clock domain to which a particular latch belongs. The next task was to map the primary inputs or latch inputs to any given latch. Since the clock domain of each latch was already known, and the path from any latch to the sources for that latch was known, the path could be identified as synchronous or asynchronous. Also, the path could be analyzed to check whether asynchronous design guidelines had been followed. Once all asynchronous crossings had been identified, during cycle-based simulation the output of an asynchronous latch could be randomly modified to simulate both jitter and the effects of combinatorial logic switching. Verification coverage metrics were developed to ensure that all asynchronous interfaces had been verified.

Verification IT infrastructure

Sim farm

A massive simulation infrastructure was required to simulate a processor and system of this size and sophistication. The infrastructure begins with the "sim farm," which consists of thousands of IBM pSeries* workstations and servers running the AIX operating system in Austin, Texas, and Rochester, Minnesota. The network is connected using 100Mb Ethernet at each site. The processing capability enabled us to average over a billion cycles per day.

Job submission

The simulation team created roughly 10 000 definitions and specifications in order to test and target different parts of the microarchitecture, parameter combinations, and coverage events. An internal job-submission tool enabled us to resolve several challenges in managing the numerous defs across thousands of systems. First, some of the defs were more "interesting" than others, so one would wish to generate more tests from those defs than the others. Second, the defs were created over more than a year, so when a new interesting def was created, it was important for that def to receive priority. Third, we wanted to submit jobs such that each def would meet a threshold criterion (defined as the number of cycles or number of tests before ceasing to generate tests from that def) equally with other defs. For example, we wanted all 10 000 defs at 60% of their criteria at the same time, rather than 5000 defs at 3% and 5000 defs at 97%. This feature enabled us to manage risk more effectively by making it easier to quantify the risk of taping out the processor before the retirement criteria were met. The submission tool created and sent jobs to a batch system which dispatched the jobs to the sim farm.

Results database

An internal database collected the results of failures and passes. Handling the results from the large sim farm required that the database process updates approximately once a second. An update consists of replicating and distributing the results in several predefined categories and immediately updating summarized views of the data to provide very fast queries.

Failure and bug management

Running this number of tests was straightforward when all tests were passing, but if a bug existed in some heavily exercised logic, tens of thousands of failures could be generated within hours. In addition to a record of the failure, other debug information, such as the simulator output, the RTX output, the parameter file, and the test case, was sent to a distributed-results processor. An internal failure-tracking tool provided an API and a GUI for the simulation and logic designers to view the failures. The GUI allowed sorting and collapsing on any field, which was useful when a common bug caused thousands of instances of the same failure. The user then selected and checked out the failures for debugging, preventing others from duplicating effort by debugging the same failure. If the failure was a design bug, the tool saved the test case in a regression bucket.

A separate internal bug database was used to track individual bugs. Every step in the bug discovery, debug, and fix validation was logged by the verification engineer and logic designer working to fix that particular bug. Similar bugs across different versions of the chip could be grouped together to follow the progress of related bugs and to note trends that might require a refocusing of effort.

Model build

When a design change was made, we decided in daily debug meetings which ones we wanted to incorporate in a new unit, multi-unit, or system model. On normal drops, the unit would build first and regress the change. The multi-unit core and/or GPS would build and regress next. The process was automated such that given a bill of materials, the code would assemble the correct VHDL revision levels, check to make sure promotion criteria were in place, compile the corresponding checkers in the RTX, and run a short regression test. Assuming that the test was successful, the code would promote the model and load it on a model server which automatically replicated the data across several RS/6000 servers. In this way, numerous requests for the RTL models and RTX could be met in a timely fashion. The chip model build followed a similar process, but many more models were built to handle the different types of simulation at the chip level. This investment in model-build automation paid huge dividends in reducing mistakes which were prone to happen during manual combining of the thousands of VHDL files, the correct version of initialization and parameter files, and the hundreds of C++ checker, driver, and monitor files that make up the RTX.

Controlling the verification process

Critical aspects in controlling the overall verification process were management of the hierarchical approach, bug analysis, effective communication, and the hardware readiness assessment. The hierarchical approach was managed with significant overlap to meet the aggressive development schedules. While bug discovery and removal are fastest at the lower levels, small teams were charged with "trailblazing" the next level of the hierarchy to ensure that the environment was stable.

Special attention was paid to bugs that were found in the larger verification models, and in most cases these discoveries fed improvements back into the lower-level test plans. For example, if a problem was found on the chip model in the FXU logic, the FXU unit environment was analyzed and improved. This feedback mechanism extended to the learning acquired from parts in the bringup laboratory. When a bug escaped the complete verification cycle and survived into hardware, the verification team not only worked to reproduce and debug the problem, but sought ways to enhance the verification environment so that future releases of the hardware would not be exposed to similar bugs.

In order to be successful in a program of this size, communication and status tracking were essential parts of our process. It was clear very early in the program that a daily meeting was appropriate and necessary to discuss technical problems, issues, daily direction, and focus items for the team. During the front end of the development cycle, this meeting was used primarily to focus the failure-debug team. As the development cycle progressed, the meeting turned into more of a daily verification tactical meeting. Issues were discussed such as bug escapes to the next level of the verification model hierarchy, utilizing coverage data to improve the test plans, and tracking project checkpoints. As the tape-out date approached, there was an additional daily meeting which served as a forum for discussing late changes that were being introduced into the design. This gave the development team an opportunity to discuss verification, timing, physical design, and other risks associated with the change, and led to a more informed choice as to whether or not to include the change.

With a program of this size, the verification team faced a challenge in assessing when the design was ready to commit to hardware. Readiness to build parts was based on the following criteria:

- Execution of a static test plan.
- Exercising a random test plan for a sufficient number of cycles.
- Architectural coverage metrics.
- Microarchitectural coverage metrics.
- Bug curve characteristics.

These criteria were not only used before tape-out but were continually monitored throughout the entire verification cycle and used to dynamically alter areas of the test plan. In some cases, the bug curve and escape rate pointed to inefficiencies in the verification techniques applied to a particular area of the design. In such cases, different approaches or techniques were adopted to increase the robustness of the overall verification effort. Other tape-out criteria, such as the need to physically test circuits and technology, were also applied.

Hardware results for POWER4

The results of the simulation investment and effort were apparent when the POWER4 hardware arrived in the laboratory in the first quarter of 2000. The first-pass chip successfully booted AIX and Linux** on multiple system configurations, including a 32-way system—an incredible accomplishment given the size and complexity of this completely new design effort. Some simulation escapes were found, but these were effectively handled with software workarounds or degraded performance modes. The resulting functionality of the chip enabled progress on the rigorous hardware-verification phase of the project, enabling the POWER4 system to track to the original aggressive schedule that had been set more than four years before.

POWER4 results—Methodology analysis

The POWER4 simulation effort caught more than 96% of the bugs before the chip was committed to hardware. The result exceeded our initial goal of fewer than 6% escapes to the laboratory, which we believed was necessary to meet the aggressive hardware-testing functionality requirements and schedule.

An analysis of simulation bug discovery demonstrates that the hierarchical methodology and tools were very effective. For all functional bugs found in the logic contained in the core, 92% were found in block, unit, and multi-unit sim. The chip-level simulation found roughly 5.6%. Fewer than 3% were found in the system simulation and in laboratory hardware, where the debug cycle requires significantly more time than at the lower hierarchical levels of simulation.

Similar results were found in the logic contained within the GPS. Block, unit, and multi-unit simulation accounted for 91% of the bugs. System-level simulation found 6.3%, which was expected since the GPS contains the interfaces to the I/O and memory controller chips. Fewer than 3% escaped to laboratory hardware.

Since the investment in rigorous checkers resulted in such a high discovery rate at the block, unit, and multi-unit levels, it proved critical to reuse the checkers at the chip and system levels of simulation. In particular, the instruction-by-instruction-level checking developed for the core model, the GPS RTX checkers developed for the multi-unit model, and the CML checkers proved to be the most valuable, since they effectively discovered bugs across several hierarchies of simulation. In addition to detecting problems at or near the point of failure, these checkers allowed for rapid isolation down to a single unit for most failing test cases without the need for creating a costly AET (a signal trace file). The productivity gained by these checking/tracing mechanisms was invaluable throughout the course of the program at multiple levels of verification.

The POWER4 project invested a great deal of design and verification effort in RAS functionality. The benefits of such a heavy investment in RAS and debug facility design and verification resulted in the smooth bring-up of hardware upon initial delivery from fabrication and packaging. Furthermore, it allowed for substantial improvements in the efficiency of debugging for mainline hardware and software functions in the laboratory over prior PowerPC designs. These improvements occurred in spite of the fact that RAS bugs actually accounted for nearly 20% of the overall bugs found on POWER4. This high bug rate is due to the vast number of functions supported by RAS.

POWER4 results—Laboratory escape analysis

The "postmortem" analysis of the bugs that escaped to the hardware laboratory was a critical piece of work necessary for improving the quality and time-to-market for each new product. The POWER3 microprocessor team provided a detailed escape analysis. As a result of this effort, many of these types of bugs were avoided completely in POWER4 because of improved verification environments, test cases, and checking.

Simulation escapes for POWER4 were carefully reviewed not only by POWER4 team members, but also by IBM developers from other projects. The results of the POWER4 escape analysis have resulted in architectural changes, logic redesigns, new test-case development, new simulation environments, and new checkers. Some specific examples include the following:

- Some instruction corruption problems related to changes introduced to meet timing objectives. These bugs were seen in the laboratory and recreated, primarily using formal verification.
  - *Analysis:* These bugs placed a greater emphasis on the importance of designing for verification. Certain areas were redesigned to simplify the verification process.
  - *Action:* This led to increased formal verification, enhanced coverage analysis, a dynamic "command-driven random" simulation environment (Dtf) for that unit, and a logic redesign of that portion of the chip.
- A move to machine state register (MTMSRD) instruction that caused a processor checkstop.
  - *Analysis:* This instruction has historically proven to be difficult to verify because of synchronization requirements imposed on some instances of the instruction.
  - *Action:* A change to the PowerPC architecture was made to define a new version of this instruction that allows it to be context-synchronizing in less performance-critical sections of code. This eliminated the need for a subsequent context-synchronizing instruction, which made verification of the MTMSRD more deterministic. This demonstrated not only the idea of designing for verification but also that ease of verification is an important consideration at the architectural level. This change maintained binary compatibility with previous PowerPC designs while greatly simplifying the verification task.
- One bug that resulted in a dropped store or a load hang in the laboratory. This was not seen until performance improvements allowed the bug to surface more easily. The root cause was the core's completing a "no-op" instruction too quickly.
  - *Analysis:* This example demonstrates that no cases should be taken for granted in verification. While early completion of the no-op was harmless in the overwhelming majority of cases, adding a checker specifically for this simple situation detected the resulting rare error much more easily.
  - *Action:* A checker was implemented that checks for completion of instructions earlier than considered possible.

- Some bugs involving moving cache lines between processors across different levels of the cache hierarchy that resulted in either a system hang or stale data. The problems resulted from the multiple-level cache hierarchy.
  - *Analysis:* In verifying a system this complex, designing for verification can help by adding specific modes used only for verification that stress areas of the design beyond normal usage. In this case, a direct-mapped mode of the caches was utilized during verification to produce inter-cache traffic more rapidly. While at first glance adding these extra modes can be seen as adding extra complexity to the design, they were extremely beneficial for locating hard-to-find bugs.
  - *Action:* Simulation was biased to use the direct-mapped mode setting, and the cache preloader was enhanced to make it easier to produce these kinds of situations in simulation.

As these examples show, one cannot take any part for granted in validating a design. Designing for verification, though requiring an investiment up front, pays dividends in a shortened and more thorough overall verification effort.

Summary

Overall, the POWER4 team believes that these simulation results were exemplary in light of the many technical and schedule challenges presented. While verification is an ever-evolving "art," the methodologies employed in POWER4 verification have demonstrated their effectiveness at removing functional problems from what may be the most complex microprocessor and SMP system ever designed.

The initial step to success was recognizing the architectural, implementation, and new design challenges. The fundamental choice of cycle-based simulation and the capital investment in the sim farm infrastructure provided the foundation to simulate more than a billion processor cycles per day across a size range from small to extremely large models. The hierarchical approach to verification enabled us to effectively use numerous test-case generators, simulators, checkers, formal methods, coverage models, and tests at the appropriate levels. This approach was validated by finding more than 90% of the bugs at the block, unit, and multi-unit levels. Checkers designed for reuse enabled more effective chip- and system-level simulation, where isolation of failures improved debug time. While most bugs were found at the lower hierarchical levels, it is currently not feasible to remove 100% of the bugs at these levels. Continued investment is needed to improve chip- and system-level simulation, simulation accelerators, and hardware test tools while developing new verification technologies to improve effectiveness at the block, unit, and multi-unit levels.

In addition to the tools and methodology, the magnitude of the project presented management and control challenges. Key factors to success were managing the overlap of each hierarchical simulation effort, effective communication, and analyzing bugs and bug trends and taking action based on the findings.

By extending their experience from prior machines such as the POWER3 system, the verification team successfully utilized and integrated their methods in verifying the POWER4 system. The methodologies used on POWER4 and the Regatta system will be extended and enhanced on subsequent products.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Linus Torvalds.

References

1. B. Wile, M. P. Mullen, C. Hanson, D. G. Bair, K. M. Lasko, P. J. Duffy, E. J. Kaminski, Jr., T. E. Gilbert, S. M. Licker, R. G. Sheldon, W. D. Wollyung, W. J. Lewis, and R. J. Adkins, "Functional Verification of the CMOS S/390 Parallel Enterprise Server G4 System," *IBM J. Res. & Dev.* **41,** 549–566 (July/September 1997).
2. C. May (Ed.), E. M. Silha, R. Simpson, and H. S. Warren, Jr. (Ed.), *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Morgan Kaufmann Publishers, Inc., San Francisco, 1994, pp. 321–324, 392.
3. J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM J. Res. & Dev.* **46,** 5–25 (2002, this issue).
4. J. Darringer, E. Davidson, D. Hathaway, B. Koenemann, M. Lavin, B. Lee, J. Morrell, S. Ponnapalli, K. Rahmat, W. Roesner, E. Schanzenbach, and L. Trevillyan, "EDA in IBM: Past, Present and Future," *IEEE Trans. Computer-Aided Design, Integrated Circuits & Systems* **19,** 1476–1497 (December 2000).
5. V. Paruthi and A. Kuehlmann, "Equivalence Checking Combining a Structural-SAT Solver, BDDs, and Simulation," *Proceedings of the IEEE International Conference on Computer Design*, Austin, TX, September 2000, pp. 459–464.
6. A. Kuehlmann, A. Srinivasan, and D. P. LaPotin, "Verity—A Formal Verification Program for Custom CMOS Circuits," *IBM J. Res. & Dev.* **39,** 149–165 (January/March 1995).
7. I. Beer, S. Ben-David, C. Eisner, and A. Landver, "RuleBase: An Industry-Oriented Formal Verification Tool," *Proceedings of the 33rd Design Automation Conference*, 1996, pp. 655–660.
8. William Diamond, *Practical Experiment Design for Engineers and Scientists*, John Wiley & Sons, Inc., New York, 1997.
9. Aharon Aharon, Dave Goodman, Moshe Levinger, Yossi Lichtenstein, Yossi Malka, Charlotte Metzger, Moshe Molcho, and Gil Shurek, "Test Program Generation for Functional Verification of PowerPC Processors in IBM," *Proceedings of the 32nd Design Automation Conference*, 1995, pp. 279–285.

Received July 15, 2001; accepted for publication November 26, 2001

John M. Ludden *IBM Server Group, Burlington facility, Essex Junction, Vermont 05451 (ludden@us.ibm.com).* Mr. Ludden is currently a Senior Engineer; he has been involved in design verification since joining IBM in 1990 after receiving a B.S. degree in electrical engineering from Rochester Institute of Technology. Mr. Ludden initially was involved in the verification of I/O and storage control subsystems for IBM S/390 mainframes. Since 1993, he has developed an expertise in the area of microprocessor verification, having leadership experience in the application of pseudorandom test-case generation as it applies to superscalar, out-of-order microprocessors. In particular, Mr. Ludden has applied such techniques to an x86 and several PowerPC microprocessors, including the POWER3 and POWER4. He has worked closely with the IBM Haifa Research Laboratory in Israel for more than eight years to improve test-generation capabilities in order to verify complex uniprocessor and multiprocessor design features. Mr. Ludden has received several informal recognition awards for his work on mainframes, microprocessors, RS/6000 workstations, servers, and SP supercomputers. He is actively involved in the IBM verification advisory team (VAT) and has been responsible for reviewing the verification plans for several other microprocessor designs within IBM in recent years.

Wolfgang Roesner *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (wolfgang@us.ibm.com).* Dr. Roesner is an IBM Distinguished Engineer; he is the technical leader for Server Group verification tools development. He received the degrees of Dipl.-Ing. and Dr.-Ing. at the University of Kaiserslautern in 1980 and 1983. Dr. Roesner developed simulators and hardware design languages at IBM in Boeblingen, Germany, later joining the POWER processor development team, where he co-developed the TexSim simulation system. His verification tools have been used on all IBM CMOS microprocessor projects, and since 1996 he has been responsible for the strategy of verification tools development. Dr. Roesner has received three IBM Outstanding Achievement Awards and one IBM Corporate Award for development in hardware design language processing and simulation.

Gerry M. Heiling *8113 Asherton Cove, Austin, Texas 78750 (gheiling@swbell.net).* Mr. Heiling received his B.S.E.E. degree from the University of Minnesota in 1967. He was employed at the Space Physics Division of Boeing from 1967 to 1969. In 1970, he received his M.S.E.E. degree from the University of Minnesota. That same year, Mr. Heiling joined IBM at Rochester, Minnesota, where he worked as an analog/digital circuit design engineer on a number of IBM products including System/3, System/36, System/38, and AS/400. From 1983 to 2001 he held various IBM management positions. He and his teams designed various analog, digital, and mixed-signal chips used in I/O, communication, and memory controller applications for IBM AS/400 and RS/6000 products. From 1996 to 2001, Mr. Heiling was a design manager in the RS/6000 processor group working on memory controllers/subsystems, L3 memory caches, and I/O controllers. He also managed the pre-silicon verification of the memory cache, I/O, and fabric portion of the POWER4 chip. Mr. Heiling holds ten patents with IBM. He is retired from IBM and resides in Austin, Texas.

John R. Reysa *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (reysa@us.ibm.com).* Mr. Reysa joined IBM in 1987. He holds a B.S. degree in electrical engineering from Texas A&M University and an M.S. degree in electrical engineering from the University of Texas at Austin. He is a Senior Engineer in the POWER4 simulation group. Before assuming his current position, he managed the POWER3-II processor development effort. Prior to that, he managed the POWER3 simulation group, all in Austin, Texas. Mr. Reysa has received an IBM Outstanding Innovation Award for his work on RS/6000 simulation.

Jonathan R. Jackson *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (jrj1@us.ibm.com).* Mr. Jackson is currently an Engineer in the IBM Server Group. He joined IBM in 2000. He received a B.E. degree with a double major in electrical engineering and computer science from Vanderbilt University in 1998 and an M.S. degree in electrical and computer engineering from Carnegie Mellon University in 2000. He currently works on storage subsystem verification for POWER4.

Bing-Lun Chu *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (bchu@us.ibm.com).* Mr. Chu received his B.S. degree in engineering science from the State University of New York at Stony Brook in 1973 and his M.S. degree in electrical engineering from Cornell University in 1974. He joined IBM at Poughkeepsie in 1978, working on storage subsystem design for System/370. Later, he focused on SMP storage subsystem functional verification. Mr. Chu pioneered the use of a random driven/concurrent checking methodology that was proven to be extremely successful in several IBM products including the POWER4 storage subsystem, the zSeries server, and its predecessors. He is currently the team leader for POWER4 and its follow-on storage subsystem verification.

Michael L. Behm *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (mbehm@us.ibm.com).* Mr. Behm is currently a Senior Engineer; he joined IBM in 1978. He received a A.S. degree in electronics technology from Lincoln Technical Institute in Allentown, Pennsylvania. He has spent five years in manufacturing test, three in the S/370 engineering laboratory, eight in S/390 core/vector processor verification, and the past seven years in POWER3/POWER4 core/chip verification. Mr. Behm's current assignment is core/chip architectual and u-architectual coverage verification.

Jason R. Baumgartner *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (jasonb@austin.ibm.com).* Mr. Baumgartner joined IBM at Austin in 1995. He has spent most of his career as a development engineer in verification tool groups, currently working in functional formal verification (FFV) application/research/development. In addition to involvement in tool development, he is the technical leader of FFV deployment within the Austin Server Group. He has authored publications on several key new FV technologies, each of which has shown great utility in reducing FV complexity for industrial designs. Mr. Baumgartner received his B.S.E.E. degree from the University of Florida in 1995 and his M.S.C.E. degree from the University of Texas at Austin in 1998. He will be completing a Ph.D. in computer engineering from the University of Texas at Austin in 2002. He holds numerous patents in the area of FV, and has reached the Fifth IBM Invention Plateau.

Richard D. Peterson *IBM Server Group, 3605 Highway 52 North, Rochester, Minnesota 55901 (petersn@us.ibm.com).* Mr. Peterson joined IBM in 1983; he is currently a Senior Engineer. Since 1992, he has been working on processor and system verification for IBM eServer, iSeries, and pSeries machines. Mr. Peterson received a B.S. degree in electrical engineering from the University of Texas in 1982, and an M.S. degree in computer engineering from the National Technological University in 1996. In 2001, he received an IBM Division Award for his contributions to AS/400 CEC verification.

Jamee Abdulhafiz *IBM Microelectronics Division, 11400 Burnet Road, Austin, Texas 78758 (abduljj@us.ibm.com).* Mr. Abdulhafiz is currently a Senior Engineer and verification leader for a POWER4 derivative microprocessor. He has worked in verification since 1995 and was system verification team leader in the Enterprise Server Group. His previous experience included logic design from 1986 to 1994. Mr. Abdulhafiz received a B.S. and an M.S. degree in electrical and electronic engineering from the University of California at Los Angeles in 1986. He received an IBM Outstanding Technical Achievement Award for his work in RS/6000 MP verification in 1998 and an IBM Invention Achievement Award for his high-speed data access system patent submission in 1991.

William E. Bucy *IBM Server Group, 11400 Burnet Road, Austin Texas 78758 (bucy@us.ibm.com).* Dr. Bucy joined IBM in New York in 1977, working in the Mid-Hudson Valley facilities from 1977 until 1991 as a Quality Assurance Engineer/Manager in the fields of semiconductor components, printed circuit cards and boards, and subassemblies. In 1985 he moved into development and contributed to the success of the IBM 200Mb Fiber Channel for data communication. In 1991, Dr. Bucy moved to IBM Austin, where he helped establish the system verification team for the RISC microprocessors. His most recent management accomplishments were in the functional system simulation of POWER3 and POWER4 products. Dr. Bucy received his B.S. degree in chemistry from West Virginia University in 1972 and his Ph.D in chemistry from the University of South Carolina in 1976. He is a member of Phi Beta Kappa.

John H. Klaus *IBM Server Group, 3605 Highway 52 North, Rochester, Minnesota 55901 (klaus@us.ibm.com).* Mr. Klaus is currently a Senior Engineer; he joined IBM in 1981. He received a B.S. degree in electrical engineering from the Illinois Institute of Technology in 1981, and a master's degree in computer engineering from the University of Minnesota in 2000. Mr. Klaus has worked on the development of S/390 channels and I/O subsystems for 13 years; for the past six years he has been responsible for the hardware design verification of I/O chips on the AS/RS systems. He received an IBM Team Award for his work on the S/390 Parallel Sysplex Team and an IBM Excellence Award for his work on the Condor series of AS/RS machines.

Danny J. Klema *IBM Server Group, 3605 Highway 52 North, Rochester, Minnesota 55901 (klema@us.ibm.com).* Mr. Klema is an Advisory Engineer in the System Pervasive Verification group. Before assuming his current position, he was team leader for printer attachment verification for AS/400 systems. He has also designed seveal chips and cards used in communications and I/O attachment for the S/36 and AS/400 systems. Mr. Klema received an IBM Excellence Award for his work in system verification for the Condor products. He has spent his entire career at IBM in Rochester, Minnesota, joining IBM in 1982. He holds a B.S. degree in electrical engineering from the University of North Dakota.

Tien N. Le *IBM Microelectronics Division, 11400 Burnet Road, Austin, Texas 78758 (tnle@us.ibm.com).* Mr. Le received a B.S. degree in electrical engineering and a B.S degree in mathematics from California State Polytechnic University in 1974. In 1976, he received a B.S. degree in computer science and an M.S. degree in mathematics from the same institution. Mr. Le joined the IBM Office Products Division, working on system verification, printer designs, and manufacturing automation. In 1987 he joined the RS/6000 processor group, working on memory controllers/subsystems, caches, and I/O controllers. He pioneered a test-case-generating expert system that won a KBS Excellence Finalist Award in 1992. Mr. Le received a multiple-instance object creation patent in 1994 and three IBM Invention Achievement Awards. He is currently a technical team manager responsible for verifying the memory flow controller portion of the Sony–Toshiba–IBM joint-venture-designed processor in Austin, Texas.

F. Danette Lewis *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (danettel@us.ibm.com).* Ms. Lewis is an Advisory Hardware Engineer in the POWER4 pervasive verification group. Before assuming her current position, she was a key member of the POWER4 fixed-point unit verification team; prior to that, she did PCI verification. Ms. Lewis has spent her entire career at IBM in Austin, Texas, joining IBM in 1980. She holds a B.S. degree in electrical engineering from New Mexico State University.

Philip E. Milling *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (pmilling@us.ibm.com).* Mr. Milling is currently a Senior Engineer; he joined IBM in 1977. He received a B.S. degree in electrical engineering from the University of South Alabama in 1975 and an M.S. degree in electrical engineering from Colorado State University in 1980. Mr. Milling worked on the team which developed the IBM Personal Computer in Boca Raton, Florida, in 1981. In 1994, Mr. Milling moved to Austin, Texas, to support development of new RS/6000 PowerPC-based systems. Since 1997, Mr. Milling has worked primarily in system simulation, most recently supporting verification of POWER4-based systems. He has 24 invention disclosures published and holds six patents. Mr. Milling has received seven formal awards from IBM for his work on the IBM Personal Computer, the IBM POWER Personal Computer, and POWER4 systems.

Lawrence A. McConville *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (mccon@us.ibm.com).* Mr. McConville is currently a Staff Engineer; he joined IBM in 1992 and worked in hardware qualification prior to joining system simulation in 1996. He received a B.S. degree in electrical engineering in 1990 and an M.S. degree in electrical engineering in 1991, both from the University of Nebraska at Lincoln. Most recently, he received an IBM Excellence Award for his work on the Condor series of AS/RS machines.

Bradley S. Nelson *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (bsnelson@us.ibm.com).* Mr. Nelson received his B.S. degree in computer engineering from Texas A&M University in 1997. After graduation he joined the RS/6000 processor memory subsystem group. He worked as a verification team leader for the POWER4 memory controller and L3 data cache. During that time, he has pioneered new methodologies to verify asynchronous interfaces. He is now working as a verification project manager for the POWER5 memory subsystem design.

Viresh Paruthi *IBM Enterprise Systems Group, 11400 Burnet Road, Austin, Texas 78758 (vparuthi@us.ibm.com).* Mr. Paruthi received the B.Tech.(H) degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, in 1995, and an M.S. degree in computer engineering from the University of Cincinnati in 1997. He then joined the IBM Server Group, where he supported and applied the Verity Boolean equivalence checker to the POWER4 processor project. Later he assumed a development role, contributing to the development and enhancement of Verity's core algorithms. His current interests include functional formal and semiformal verification and abstractions in addition to Boolean equivalence checking. Mr. Paruthi has published numerous conference papers in the area of formal verification.

Travis W. Pouarz *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (tpouarz@us.ibm.com).* Receiving a B.S. degree in electrical engineering from Duke University in 1998, Mr. Pouarz joined the IBM Server Group in Austin, Texas, to work on the development and application of Verity formal Boolean equivalence checking for the POWER4. He continues this work for the POWER4 successors and a number of other IBM high-performance microprocessors, as well as development for future formal verification tools. He is the company's Verity expert for high-performance full-custom circuitry and rigorous hierarchical methodology.

Audrey D. Romonosky *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (romonosk@us.ibm.com).* Ms. Romonosky is currently a Senior Engineer; she joined IBM in 1978. She received a B.S. degree in electrical engineering from Pennsylvania State University in 1978 and an M.S. degree in computer engineering from Syracuse University in 1985. Ms. Romonosky worked on the logic development of S/390 and ES/9000 for 14 years and in Server Group system development since 1993. She has worked in system verification since 1996 on POWER3- and POWER4-based systems. Most recently, she received an IBM Excellence Award for her work on the POWER3 Nighthawk system.

Jeff Stuecheli *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (jeffas@us.ibm.com).* Mr. Stuecheli has worked in the IBM Server Development Group in Austin since 1995. He received his B.S. degree in computer engineering from the University of Texas in 1997. He is currently enrolled in the Ph.D. program at UT Austin, researching computer architecture. His work at IBM includes system, RAS, storage subsystem, and, most recently, performance verification. Mr. Stuecheli has filed patents in both verification techniques and storage subsystem architecture.

Kent D. Thompson *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (kentthom@us.ibm.com).* Mr. Thompson is a Staff Software Engineer in the POWER4 pervasive verification group. Before assuming his current position, he was a key member of the development team for the Engineering Support Processor. This tool was used for bringup and verification of all previous POWER and PowerPC chips. Mr. Thompson has spent his entire career at IBM in Austin, Texas, joining IBM in 1982. He holds a B.S. degree in computer science from St. Edwards University.

Dave W. Victor *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (dvictor@us.ibm.com).* Mr. Victor is currently a Senior Engineering Manager in charge of i/p series processor verification. He received his B.S. degree in electrical engineering and applied physics from Case Western Reserve University in 1988 and his M.S. degree in electrical engineering from Syracuse University in 1990. He joined IBM in 1988 as a chip designer for zSeries machines in Poughkeepsie, New York. Since then, he has been a logic designer and microarchitect on various processor, memory, I/O, and graphics chips for i, p, and zSeries machines. In the early days of the POWER4 program, he was a designer and microarchitect on the instruction sequencer unit (ISU) in the processor. In February of 2000, he assumed the role of department manager, responsible for processor core and chip verification of POWER4 and follow-on design points. Mr. Victor has filed 11 patents, one of which was recognized with a supplemental patent award. He has authored four technical publications, including a recent article in the *IBM MicroNews* on memory controller design. He received an IBM Outstanding Technical Achievement Award for chip design leadership in October 1997.

Bruce Wile *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12603 (bwile@us.ibm.com).* Mr. Wile, a Senior Technical Staff Member, is the IBM Server Group verification leader. In this role, Mr. Wile is responsible for verification methodology utilization on current projects, while also focusing on future technologies and verification investments. Mr. Wile has been in the verification field for 17 years and has worked on multiple server programs at IBM. He was the team leader for the S/390 Parallal Enterprise Server G4, as well as for previous generations. Mr. Wile received his B.S. degree from Pennsylvania State University in 1985. During his time with IBM, he has received multiple awards for his verification work, including an IBM Corporate Award, an IBM Outstanding Innovation Award, and an IBM Outstanding Technical Achievement Award. Mr. Wile also holds multiple patents in the verification field.