Verification strategy for the Blue Gene/L chip

M. E. Wazlowski
N. R. Adiga
D. K. Beece
R. Bellofatto
M. A. Blumrich
D. Chen
M. B. Dombrowa
A. Gara
M. E. Giampapa
R. A. Haring
P. Heidelberger
D. Hoenicke
B. J. Nathanson
M. Ohmacht
R. Sharrar
S. Singh
B. D. Steinmacher-Burow
R. B. Tremaine
M. Tsao
A. R. Umamaheshwaran
P. Vranas

The Blue Gene®/L compute chip contains two PowerPC® 440 processor cores, private L2 prefetch caches, a shared L3 cache and double-data-rate synchronous dynamic random access memory (DDR SDRAM) memory controller, a collective network interface, a torus network interface, a physical network interface, an interrupt controller, and a bridge interface to slower devices. System-on-a-chip verification problems require a multilevel verification strategy in which the strengths of each layer offset the weaknesses of another layer. The verification strategy we adopted relies on the combined strengths of random simulation, directed simulation, and code-driven simulation at the unit and system levels. The strengths and weaknesses of the various techniques and our reasons for choosing them are discussed. The verification platform is based on event simulation and cycle simulation running on a farm of Intel-processor-based machines, several PowerPC-processor-based machines, and the internally developed hardware accelerator Awan. The cost/performance tradeoffs of the different platforms are analyzed. The success of the first Blue Gene/L nodes, which worked within days of receiving them and had only a small number of undetected bugs (none fatal), reflects both careful design and a comprehensive verification strategy.

Introduction

Very large chips and associated verification efforts were formerly seen only in the domain of processor design, such as the PowerPC* [1]. Now, system-on-a-chip (SoC) designs can rival the complexity of processor designs, but are designed by smaller teams of system designers. An SoC design comprises a number of semiautonomous subsystems that are integrated to form a single system. Any of these subsystems can be as large and complex as an entity that, just a few years ago, would have been considered an entire chip in itself. While it is true that some components in an SoC design are pre-designed and pre-verified (e.g., an embedded processor), the complexity of SoC designs poses a challenge to a small verification team because of its large breadth. State-of-the-art verification relies on a fundamental set of concepts.

Unit-level simulation is used to efficiently verify a single component inside the chip. This allows the verification engineer to more easily write tests targeted at the interface of the unit under test without having to satisfy conditions or constraints of other units. A disadvantage of unit-level simulation is that it does not verify the interaction of the unit with other units. Unit-level simulation is generally much faster than system simulation because there is a smaller runtime overhead.

System simulation is used to comprehensively verify that the chip operates correctly in its targeted system configurations under all stress conditions. This method of verification complements unit-level simulation in that it covers the interaction of the unit with its neighbors, as unit-level simulation alone does not. There are two main disadvantages of system simulation: First, it is often difficult to create specific events from outside the chip at the interface of a unit that is deep inside the chip; second, simulations run slowly because of the large compiled simulation image.

Whether verifying at the unit or system level, two types of stimulus generators may be employed: *deterministic*

0018-8646/05/$5.00 © 2005 IBM

and *random*. If a unit is very simple, a deterministic (also known as *directed test*) stimulus should be used. The advantage of deterministic verification is that a verification engineer can write a suite of tests that either enumerate all possible states or target specific scenarios for the logic being verified, run the test suite in regression, and verify the function. In practice, however, this can be accomplished for only the simplest of units and requires extensive engineering time. For these simple units, formal verification techniques can be considered because the state–space of these units can be verified in a relatively short length of time with a computer containing a moderate amount of memory.

Random simulation relies on the test environment to generate the test scenarios in such a way that they can be generated at machine speed indefinitely, without intervention. We rely on *directed random* simulation—a hybrid of the directed and random testing strategies. This scheme retains the advantages of random verification but provides control to the test engineer to focus the random testing on a specific corner of the design space or functionality.

Coverage metrics can be used to help quantify which design functions have been reached by simulation. Code-coverage metrics that determine whether each line of Very high-speed integrated circuit Hardware Description Language (VHDL) code has been exercised provide some value, but they provide no means of determining the context in which the line of VHDL was exercised. This can lead to a false feeling of coverage. A solution is to embed assertion statements in the VHDL that enable specific functional coverage information to be logged—for instance, including an assertion statement that indicates when the L3 controller read address queue is full. Subsequently, the number of times this event occurs in the simulation can be counted, and the information can be used to tailor the random simulation if the queue is full too rarely or too often. A pitfall of this approach is that it relies on the VHDL designer to think of coverage checks and to code them. It is possible for a designer to miss some of these coverage checks. In spite of their shortcomings, it is desirable to use assertion statements, because they are flexible and can provide better information on what has been verified when compared with code coverage.

Full processor models allow the use of real software applications to run in a system-level simulation. The software is cross-compiled and then run on the processor model in the verification environment. A benefit of using a full processor model in system-level simulation is that the software team can develop system software and test it in the simulation environment before the actual hardware is running in the laboratory. Also, software-based stress tests written for verification can easily migrate into the laboratory, and there is the additional benefit that the behavior of the processor in the system can be observed. This information can be used to tailor a bus-functional processor model to more closely emulate some of the traits of the real processor.

A bus-functional processor model provides a means to drive transactions on the processor bus. The model does not contain any of the internal behaviors or intelligence of a full processor model; it simply knows how, for example, to do reads and writes on the processor bus when it is told to do so by the verification engineer. A bus-functional model can be used to more strenuously drive transactions on the processor bus because it does not have to wait for pipeline stalls or instruction fetches, like the full processor model. The verification engineer can program as many reads or writes in succession as desired.

This paper discusses the strategy and methods required for a small verification team to verify an SoC design in the context of Blue Gene/L (BG/L). In this paper, we describe the BG/L compute chip (BLC) architecture and the verification strategy, tools, and hardware platforms used to validate the chip. The subsystem or unit verification of the BLC network interfaces is discussed next, followed by random system simulation and code-driven system simulation. The paper concludes with a discussion of the verification results and a summary.

BG/L architecture

The node architecture of BG/L is shown in **Figure 1**. The BG/L design uses an SoC architecture in which each node in the 65,536-processor supercomputer consists of one BLC with memory ranging from 256 MB to 1 GB of external double-data-rate synchronous dynamic random access memory (DDR SDRAM). The BLC contains two PowerPC 440 (PPC440) embedded cores [2]. Each of these cores contains a *double-hummer* floating-point unit (FPU), which consists of two 64-bit PowerPC FPUs programmed with single-instruction multiple-data (SIMD) instruction-set extensions supporting up to 5.6 gigaflops per node at 700 MHz. Each core is attached to a private L2 prefetching cache through three separate buses: instruction fetch, data read, and data write. Although there is no coherence support between the L1 caches of the two cores, the L2 caches support coherence at L2 and beyond through a "snoop" interface. The L2 interface also provides access to all memory-mapped input/output (I/O) devices, including the torus, collective, and global interrupt networks, a shared 16-KB static random access memory (SRAM), and the Gigabit Ethernet controller. A 4-MB embedded DRAM L3 cache provides coherent interfaces to each L2 and interfaces with the DDR controller. In addition to the memory-mapped I/O (MMIO) interfaces, each BLC contains a set

of device control registers (DCRs) used for configuration and status. The torus network is used for point-to-point communications between nodes for computation, whereas the collective network is used for global communications, such as broadcast and reduction. On the BLC, there are 734 DCRs of up to 32 bits each. Other functional units in the BLC include the interrupt controller and the *lockbox*, which provides atomic test-and-set semaphore and barrier operations between the cores. The 440 CPUs, the double hummers, and the Ethernet subsystem, which includes a bus bridge and a direct memory access (DMA) controller, are off-the-shelf cores and are shaded blue in Figure 1. All other logic—approximately 900,000 lines of VHDL—had to be synthesized and verified.

Strategy

The overall verification strategy for the BLC is built upon a hierarchical approach that uses a number of different methodologies. Each of these methodologies has specific strengths and weaknesses, and the BLC verification strategy uses the methodologies in a complementary fashion to exploit strengths and compensate for inherent weaknesses. The methodologies employed for BLC verification are more or less standard [3] and include unit-level, formal protocol, random system-level, and code-driven system-level verification.

The BG/L chip unit-level strategy was to minimize the number of bugs found at the system level. Debugging chip failures at the system level is significantly more difficult and costly than debugging failures at the unit level, so we also chose to do unit-level simulation. Verification engineers and designers of a given unit collaborated on the unit-level verification solution. Unit verification was performed on the L3 controller, the DDR interface, the collective interface, the torus interface, and many other units and subunits.

Although unit simulation is critically important in the overall verification strategy, space constraints do not allow us to describe them all in this paper. Only the collective and torus unit-level testbenches are described. The unit-level testbenches are simpler than the system-level testbench and are thus usually available before the system-level testbench. This strategy allows the system-level testbench to be developed while the unit-level tests are running.

The BG/L system verification strategy targets the following:

- Chip correctness in a large system.
- Low-cost, rapid implementation.
- Support for logic design debug.

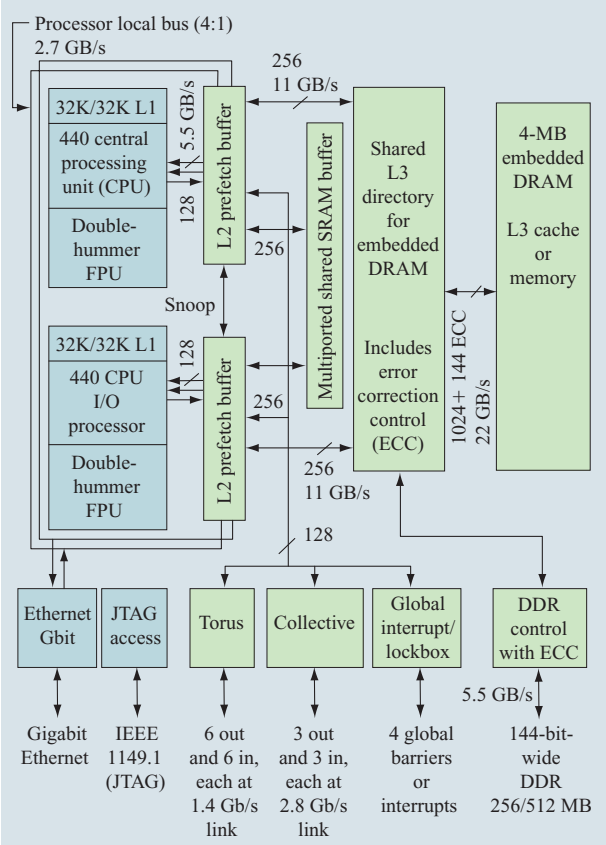

Figure 1

Blue Gene/L compute (BLC) chip architecture. Blue shading indicates off-the-shelf cores. ©2002 IEEE. Reprinted with permission from G. Almasi et al., "Cellular Supercomputing with System-on-a-Chip," *Digest of Technical Papers*, 2002 IEEE International Solid-State Circuits Conference.

- Compatibility among tools, libraries, models, and preexisting logic code.
- Efficiency.

Efficiency was critical to the verification strategy, as only six months were afforded to develop and implement the system verification strategy and validate the chip before fabrication. We emphasized strategic efficiency as *cycles per effective simulation event* and *test case per simulation engineer hour*. Effective simulation events are defined as simulation events that actually test the design, as opposed to simulation events that are expended to prepare or flush the system state in response to a desired event or scenario. Bus and interface model generators were leveraged to maximize effective simulation events, while a random test-case generation environment was used to maximize test-case generation. Strategically, the testbenches could have been written more easily in C.

However, the aggressive schedule, preexisting models, and support for multiple simulators led us to choose VHDL as the language for all testbenches.

The BLC contains two PowerPC processors, but using full processor models in every simulation would render the system-level simulation too slow. In addition, the goal was not to verify the processors themselves, but to verify their interaction with the rest of the logic. The processors, which are off-the-shelf components, had been pre-verified. The processor bus model, written in VHDL, replaces a full processor model and interfaces with the memory system via the processor bus. Creating a bus-functional model rather than using the full processor model provided the following benefits:

- It has significantly improved verification throughput. The bus model runs much faster than a full processor model because it performs far less work per simulation cycle.
- A bus-functional model can be made to appear much more random than a full processor model.
- The bus-functional model provides the opportunity to inject errors on the processor bus so that the error-handling capabilities of the logic can be tested.

For the BLC, we chose to use assertion statements embedded in the VHDL rather than code coverage because, as discussed above, more sophisticated reporting can be gained from assertions. Assertions were written for functional coverage in every unit—L2, L3, collective, torus, etc.—and to flag fatal error conditions. Verification was not considered complete until all of the functional coverage assertions were hit.

A verification aid often used in BLC verification is the reduction of maximum queue depths or cache associativity. Most controllers or state machines that manage queues have special conditions for being empty or full. With large queues or FIFOs (queues in which access becomes available according to the *first in, first out* rule) in the design, it is more difficult to cause the special full and empty conditions to occur frequently during simulation. In the VHDL, the simulator forces the maximum queue depth to be a smaller number at time 0 to help exercise these conditions. This technique is used on several queues and on the associativity of the L3 cache. The associativity of the L3 cache is randomly reduced on some simulations from eight-way to four-way or two-way.

Gate-level event simulation with back-annotated timing is an essential part of the verification effort because it can uncover uninitialized latches and dynamic timing paths that are not found by static timing tools. The benefits of gate-level simulation are discussed in more detail below.

Because of the hardware-specific programming model for BG/L, extensive development is required in order to produce the software that will run on the BG/L hardware. The software includes both the operating system and applications. To allow the development of software before the BG/L hardware is available in the laboratory, code-driven system simulation is employed. The code-driven environment uses full processor models that allow actual software binaries to be run in the simulation.

Verification platforms

Several verification platforms were selected for use in BLC verification. We used both cycle simulators and event simulators. If the only concern had been simulation speed, we would have used only cycle simulation, but cost and verifiability also entered the decision process. The IBM internally developed cycle simulator Mesa was used because it is an order of magnitude faster than event simulators. Code written for cycle simulation is easily transferred to Awan [4], the hardware accelerator, which is an order of magnitude faster than Mesa, but at a substantial monetary cost. However, cycle simulation requires a different VHDL coding style than for event simulation. Because of its nature as a cycle simulator, Mesa requires special practices to simulate transparent latches and subclock cycle events, e.g., analog behavior of the chip I/O interface. A solution is to run the fundamental clock of the cycle simulator at a higher frequency than the actual clock required by the chip. Unfortunately, adding more cycles to the cycle simulator slows down the simulation. Ultimately, a cycle simulator can never really simulate analog behavior because of the coarse granularity of the fundamental simulator cycle, which means that it cannot be used for gate-level simulation with back-annotated timing information because this requires fine-grain analog behavior.

The above shortcomings of cycle simulation adversely affect the verifiability of the design when cycle simulation alone is used to verify the design. Back-annotated gate-level event simulation is required in order to find dynamic timing issues and to find errors in scripts written to control a static timing tool. Certain issues (such as the simultaneous read and write of an SRAM or a static timing tool script that declares a path to be multicycle when it is not) cannot be detected by cycle simulation coupled with a static timing tool.

Each of the simulators—cycle for its speed and event for its accuracy—has its place. For the BLC, cycle simulation running on Awan is used for code-driven system simulation using the full processor model. The same simulation is prohibitively slow on an event simulator and is impractical. The event simulator is used to simulate BLC with the processor bus-functional model in random system simulation, including the analog

behavior of the chip I/O interfaces. The event simulator is also used for gate-level simulation.

The hardware platforms in use by BG/L are 32-bit Intel-based Linux** machines, 64-bit PowerPC-based AIX* machines, and the hardware accelerator Awan. There are 100 Intel-based machines running Linux and Cadence NC-VHDL which constitute the simulation farm for the random system-level simulation. There are tens of PowerPC machines running AIX and Mentor Graphics ModelSim**.

Torus multinode unit-level verification testbench

A multinode torus verification testbench was developed in VHDL to enable networks with different sizes and shapes to be simulated under a variety of traffic patterns. This testbench, written and maintained by one person, consists of the torus logic, a packet generator and injection unit, a packet reception and checker unit, links, and a global control unit (**Figure 2**). This testbench does not include the driver (receiver) units that serialize (deserialize) bytes onto (from) the links.

The packet generator creates packets of a given *workload*, where the workload consists of packet destinations, sizes, virtual channels, etc. These workloads can be quite flexible. For instance, they may consist of random destinations, nearest neighbor, hot-spot, broadcasts, and so on. There can be dependency between consecutive destinations to model long messages. In addition, a simulation can be flexibly configured so that each node selects a different workload with different parameters according to some probability distribution. The packet generator also creates self-checking packets, so that when a packet is received—and without additional testbench coordination between the sender and receiver—the packet checker can determine that the packet arrived at the correct destination with every byte intact. The injection unit puts these packets into the torus injection FIFOs when space is available, but it can also insert random delays between injections. The reception unit checks the torus reception FIFOs for packets, reads them out, and checks the packets for correctness, which also includes checking that deterministically routed packets arrive in the correct order. Delays can be inserted between receptions, thereby allowing reception FIFOs to fill up and cause further backlogs within the network. The link units have the capability of either passing bytes through (with some delay) or corrupting bytes to test the torus error-detection mechanism and retransmission protocol.

The error rate on each link can be flexibly controlled. The global control unit is responsible for coordinating termination, i.e., determining when all packets have been received, or warning of a potential deadlock after a suitable timeout. Upon termination, each torus unit is checked to make sure that it is left in a pristine, empty state, with all FIFOs empty, all tokens accounted for, sender link-level cyclic redundancy check (CRC) equal to the corresponding receiver link-level CRC, etc.

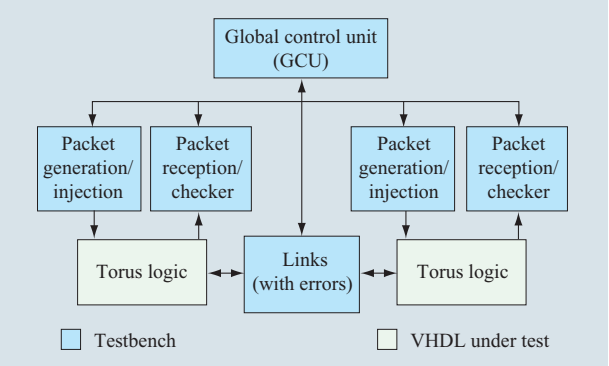

Figure 2

Multinode torus verification testbench.

Besides logic verification, this testbench was also used for performance verification for simple communication patterns. In particular, as described in [5], comparison between this VHDL verification testbench and a higher-level performance simulator caught a performance bug in one of the torus arbiters.

This testbench runs on NC-VHDL on a subset of the Intel-based simulation farm. A 2.6-GHz workstation could simulate about 200K node-cycles per hour. (A node-cycle is the number of nodes times the internal torus cycle time, which is one-fourth that of the processor.) Each node required approximately 30 MB of memory, and the largest configuration simulated was a 64-node configuration.

Assertions were placed in the torus VHDL when certain conditions (full FIFOs, corrupted packets, etc.) were hit. While most assertions were frequently hit, directed tests sometimes had to be written to hit certain assertions. In addition, after a bug was found, directed tests were typically written to recreate many times over the conditions that led to the bug, and the testbench would often be modified to more frequently include workloads that tended to create such corner cases.

The combination of thorough subunit simulation and the use of two independently created testbenches (the multinode and the random system-level), proved a powerful combination: Bugs were caught by both testbenches. When hardware arrived, a test environment similar to multinode was created and a single-node system (running in loop-back) and a large torus system were subjected to many different communication patterns. Real Message Passing Interface (MPI) applications were

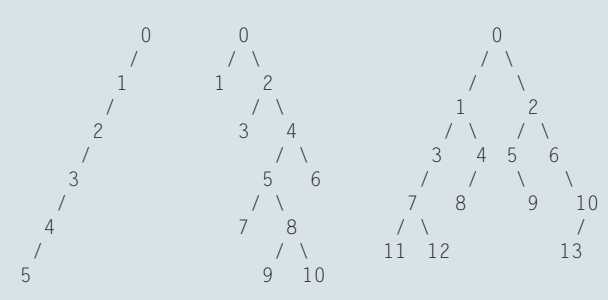

Figure 3

Three of the network shapes being simulated.

also run on this hardware. As mentioned above, the link capture units can be programmed to intentionally corrupt bytes going onto the links. This provides a hardware capability for testing error recovery similar to that in multinode link units. Running on real hardware, which provides approximately six orders of magnitude more cycles per second than can be achieved in simulation, uncovered no logic bugs in the torus.

Collective link protocol verification

The Blue Gene/L supercomputer will combine more than 65,536 collective network routers in one large system. It is very unlikely that simulations will hit all of the potential situations that may occur in a system of that size and complexity. To reduce the risk that under some strange conditions the network does not function correctly, we applied formal verification methods to the high-level design of the collective communication protocol. This effort was performed by one of the collective hardware designers. Formal verification was not required for the torus network because it was built on an existing protocol.

The collective communication protocol is special because of the usage of explicit resend requests (negative acknowledgment, or NACK) in addition to the mandatory timeouts. It was not obvious whether the addition of these protocol features would cause livelocks, or even deadlocks, because of, for example, alternating and amplifying resend requests. Another important condition that required verification is the guarantee that the senders and receivers on both sides of the link are synchronized. Synchronizing the sender and receiver means that the sender must be informed about the correct reception of the packets, which is required in order to track the availability of buffer space on the receiving side of the link. Therefore, synchronizing sender and receiver is a precondition to not generating extra packets and not dropping good packets.

Since the verification of such high-level protocol properties is extremely difficult at the register transfer level, we use the Murphi verification tool [6] and an abstracted description of the protocol to prove the correctness of the protocol. The Murphi tool basically enumerates all possible states of the abstracted protocol description. The Murphi tool can easily check for the reachability of harmful states (assertions) or for the possibility to exit any loop in the state transition matrix (livelock or deadlock). Because of the protocol abstraction, the Murphi tool can exhaustively verify the protocol.

The downside for the Murphi verification approach is the level of abstraction necessary before such an exhaustive state–space exploration can be performed successfully. Even if 64-bit systems with 64 GB of memory are used, only about 100-bit state-vectors can be used before the verification fails because of the lack of memory. Therefore, the protocol must be abstracted. Special care must be taken to implement the abstracted protocol exactly as defined and verified.

The entire formal verification approach began after a flaw in the protocol was uncovered that could have caused a temporary livelock of a link. The problem was initially found using a random-number-based, self-checking testbench. The testbench, described in the following section, had to run for several hours before the situation occurred. Since the fix of the protocol would have changed the dynamic behavior of the simulation, it would not have been clear whether the protocol change really fixed the problem or simply modified the behavior of the simulation in such a way that it no longer hit the problematic state. Using the Murphi tool, we were able not only to reproduce the situation, but also to verify that the intended correction really solved the problem and to successfully verify the correctness of the final version of the protocol.

Collective multinode unit-level verification testbench

A multinode collective verification testbench, called Forest Bench (FB), was developed in VHDL to simulate networks of various sizes and shapes under a variety of traffic patterns. FB, written and maintained by one person, is similar to the torus testbench described above. If *collective logic* were substituted for *torus logic* in Figure 2, it would accurately depict FB. As in the torus testbench, FB does not include the driver (receiver) units that serialize (deserialize) bytes onto (from) the links.

Each instance of FB consists of a global control unit (GCU) and one or more nodes. Three of the network shapes being simulated are shown in **Figure 3**.

The GCU coordinates various global phases of the simulation. The coordination uses a daisy-chained bus through all of the nodes. The daisy chain easily connects an arbitrary number of nodes and, within a node, easily connects the subunits. In all phases, the GCU collects statistics and displays the global progress of the simulation. The GCU also identifies a hung network or testbench when no progress occurs over a long interval. In a jam phase, nodes inject packets into the network but do not remove any packets. The GCU ends the jam phase when no node can inject any more packets. Similarly, in a drain phase, nodes remove packets from the network but do not inject. The GCU ends the drain phase when all outstanding packets have been received. When not jamming or draining, nodes inject and receive packets. The end phase distinguishes a failed run from a successful run, which meets all expectations and self-consistency checks.

Each FB node consists of the collective logic, the universal performance counter (UPC) logic, and the following main testbench units: DCR, send, receive, and link. In addition to other functions, the DCR unit configures test modes, such as corrupt-packet-capture and link-loopback, since FB makes aggressive use of such test modes. At the end of a run, the DCR unit also reads registers to perform collective consistency checks. Similarly, it also configures and checks the UPC logic. The link unit adds arbitrary delay to the collective links between nodes and injects errors on the links. Imitating the PPC440, the send and receive units inject packets into and receive packets from the network, respectively.

With the goal of covering all possible scenarios in the collective logic, the above and other testbench actions are driven by 95 independent pseudo-random number generators. Each generator is seeded using the same global seed modified by a unique local function. The single global seed minimizes the amount of information needed to repeat a simulation. As far as possible, a pseudo-random number generator drives each significant testbench action.

There is no per-packet communication between the send and receive units within or across nodes. Instead, the receive units replay the pseudo-random number generators of the send units in order to obtain the complete expectations for every bit of every packet header and payload. For example, for a combine packet, each receiver reproduces the contribution of each participant and performs the arithmetic operations of the reduction. The receiver halts the simulation when an unexpected bit is found.

In addition to logic verification, the testbench was used to verify the performance of some simple communication patterns.

Random system-level simulation

Unit-level verification targets the specific unit under test and facilitates finding bugs within the unit, but because the unit is just a building block, it must be tested as it works in concert with other units of the chip. System-level simulation verifies the chip as a whole when all units are connected together, and it verifies the connection of the chip to the surrounding system. In the case of the BLC, the surrounding system is a collective network and a torus network.

Testing randomly is an essential element of a simulation strategy, because random tests uncover cases that neither the designer nor verifier anticipated. For a design with any complexity, it is simply not possible for a verification engineer to list all of the events that should be tested and then test them. The key to a successful random simulation environment is to constrain the randomness to meaningful operations, while at the same time maximizing the potential randomness within this context.

The random system simulation (**Figure 4**) is controlled by three distinct classes of drivers which include a processor bus model, a collective interface model, and a torus interface model. The random environment was written and maintained by four people, two working on the memory subsystem and one each on the torus and collective networks. The testbench models are shaded blue in the figure. The sum of the three drivers is approximately ten thousand lines of VHDL. The processor bus model emulates the traffic on the processor bus and performs reads and writes to the memory subsystem and to software registers. It is instantiated three times, once for each of the two processors and once for the Ethernet DMA controller. A single instantiation of the collective interface model emulates an entire 64K-node collective network, and similarly, a single torus interface model emulates an entire torus network.

Processor bus model

In addition to having better simulation speed than a full processor model, the bus model produces a large number of test conditions in a comparatively short length of time. This benefit has two sources. One is that the stimulus can be incessant. Instruction fetches, memory latencies, pipeline bubbles, and other obstacles force a processor to stammer out bus operations. The bus-functional model has no such dependencies and can start a new bus operation as soon as the previous operation finishes. The other quality advantage derives from randomness itself. Almost by definition, random values are fresh and unexpected. Given the slowness of simulation in general and the vastness of the combinatorial space to be covered, it is important to generate as few routine, equivalent cases as possible. A processor tends to spend much of its bus time repeating the same set of operations, such as loading

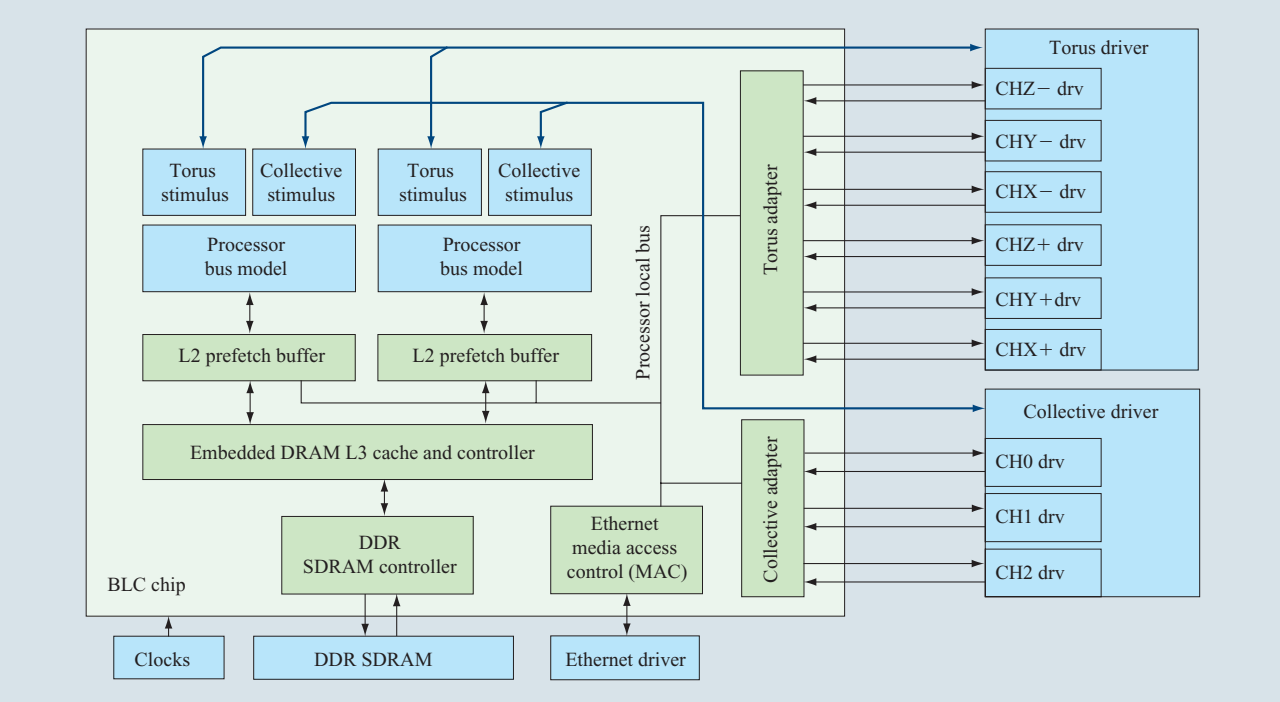

Figure 4

BLC random system simulation testbench. Testbench models are shaded blue. Blue lines indicate communication between the different testbench modules.

a code cache line. However, it is also desirable for a random bus model to reproduce at least some of the behavior of the processor. This implies that randomness should be structured to a certain extent. Left completely uncontrolled, a random bus model may not reproduce certain multi-operation sequences because the random space is so large.

Verifying a memory system is conceptually simple. The verification driver writes and reads addresses; the data read back must be the same as the data most recently written. Checker code in the simulation maintains a private memory that reflects all of the writes that have completed; read data is checked against this what-to-expect memory. In the BLC, three masters can write to memory at once: the two processor cores and the Ethernet DMA. Reads must return data from the most recent write as of the time the read began. Some simultaneous writes have an undefined result. Since badly written software could perform these kinds of undefined writes, the verification code performed them as well to ensure that they did not hang or confuse the hardware and treated the writes that resulted as undefined. A read of that write would then be ignored.

The most challenging part of creating a random testbench is controlling the stimulus to ensure that it is stressful, i.e., that it explores the entire design space and is not overly repetitive. If read and write addresses were allowed to be chosen completely at random, the test would not be particularly challenging or thorough. We added several kinds of structure to avoid this pitfall. We walked addresses forward through memory, a behavior for which the hardware has been optimized and which reflects the way actual processor addresses might typically move. We also moved them backward, a behavior that might thwart the optimized hardware. We used a relatively small number of addresses and randomly made some addresses more likely to be chosen than others—a "hot" address might be read by one core only a cycle or two after it was written by the other, and only a cycle or two before it was overwritten by DMA. Moreover, the addresses were chosen to maximize L3 contention by targeting a small number of L3 associativity classes. Randomly, we also reduced the number of ways in the L3, for example from eight-way to two-way, to further enhance the competition. This reduction in set associativity allowed us to find several bugs that would have been very difficult to find with the full associativity.

We also emulated the behavior of the processor as it loads and unloads packets to and from the collective and

torus networks. Packet loading and unloading is accomplished by a series of software register reads and writes.

In an attempt to maintain as much randomness as possible, we randomized the probabilities of certain events from test to test. For example, the delay between operations is a random range of bus cycles based on a probability, and from run to run, the range itself is changed, so that some runs tend to have more delay than others.

Random tests should be reproducible in the event they uncover a bug. The bus-functional model can completely reproduce its behavior on the basis of a single integer seed value.

Torus driver

The torus driver generates, checks, and injects faults in traffic patterns and packet structure consistent with a fully configured 64K-node system. The torus driver for the random system simulation sends data packets into six torus chip receiver input ports ($x+$, $x-$, $y+$, $y-$, $z+$, $z-$) and receives data from the six corresponding chip output ports. It also interfaces with the processor bus model via software registers, which allow the loading and unloading of torus packets.

The torus driver, written in VHDL, has two parts. As a behavior model, the first part, a low-level unit, implements the bus protocol for a pair of torus output and input ports. The second part, a high-level unit, maintains a data structure of inflight torus data packets. Whenever a torus packet can be sent on any of the eight chip ports (six x, y, z ports and two processor ports), a packet is generated on the fly with random destinations, routings, and payloads. When a packet is received by any of the testbench receivers, the inflight data packets are searched. If a packet is correctly received by the expectant testbench receiver, it is marked so. If a packet arrives at an unexpected receiver, an error is indicated. When an inflight packet is received by all of the expectant testbench receivers, it is then marked as fully received. Statistics are gathered, and the data structure is made ready to be used by the next inflight packet.

The torus data throughput is controlled by inserting a random wait interval before returning the token-acknowledge (ack) packets. For the processor interface, the torus driver waits a random interval before reading the next packet from the FIFOs.

However, at random intervals, the torus driver stops these two receive operations entirely, causing the data buffers in the chip to begin to fill up. As this happens, the torus driver switches to a heuristic mode and generates only new packets that can fill up any still-available buffers in the chip. When the buffers are as full as possible, the torus driver switches to a drain mode. At the fastest possible rate, the token-ack packets are returned, and the processor receive FIFOs are read. This drains the chip buffers at the fastest possible rate. After the buffers are totally emptied, or nearly so, the flow control changes back to the random interval mode.

One drawback of this method for verifying the BG/L torus, compared with the multinode unit-level simulation method, is that the low-level torus driver and receiver behavior have to be written into the testbench. The unit-based method simply hooked two torus units together, with little new code required, which mean that the unit-level testbench was up and running more quickly.

The major advantage of the random system simulation torus driver is its much smaller memory size and much faster simulation speed. For example, in the unit-level simulation method, to simulate the torus traffic patterns and the workload for a 64-node configuration, the unit simulator has to instantiate 64 BG/L torus units. In contrast, the torus driver described here requires only one instance of the BG/L torus driver and can simulate a 64K-node configuration.

Collective driver

The random system-level testbench collective driver component was designed to test both the collective network protocol and the collective network adapter implementation in the BG/L chip. The tester was developed independently of the actual collective network VHDL to mitigate the potential for design flaws going undetected, a potential when common functional elements are shared between the tester and device under test. The collective driver generates, checks, and injects faults in traffic patterns and the packet structure consistent with a fully configured 64K-node system.

Each BLC is a node in the system-wide collective network. The collective network adapter consists of four ports, a programmable routing table, and a queuing structure for moving traffic through the node. Traffic can be from or to the node, or a combination of both. Each port consists of a full-duplex two-byte-wide interface in which three of the ports are brought out through serialization or deserialization logic to high-speed BLC signal contacts, and the fourth port is internally connected to the BG/L processor for the node to send and receive up to eight packets through in each direction. The collective network protocol includes idle packets, sync packets, and 256-byte payload data packets. Data packets can be either combining or point-to-point (combining packets can be logically or arithmetically combined during routing). Packets are defined as either forward or reverse through the network, where reverse packets come back through the network (for example, after combining packets). Packet routing and payloads are generally pseudo-randomly generated, but can be selected from a

specific list of special patterns (e.g., "sync" pattern as data payload).

Collective network combining requires that combining packets be generated as logically related groups with common attributes, such as combining opcode and routing class. When a combining packet is randomly generated, other randomly generated packets inherit the common attributes until all of the constituent packets have been generated for a combining group. Combining "reverse" packets are always generated randomly, since they represent packets that have already been combined and are not part of a combining group. The combined packet is checked for the appropriate logical or arithmetic result.

The collective function was tested by two different testbenches; cross-coupled unit-level nodes with small collective networks stimulated by processor-generated traffic (the FB) and by the random testbench collective driver. The random testbench was significantly more efficient in terms of memory size and collective packets per second. It discovered several protocol faults that went undetected in the cross-coupled unit-level test configurations.

Code-driven system simulation

Code-driven system simulation is used in the BG/L project for logic verification, architectural validation, and performance measurements. The work was performed by one full-time worker and a number of part-time workers. Our estimate of the aggregate full-time work complement is 1.5 individuals. The BG/L advanced diagnostics environment (ADE) [7] provides complete and flexible access to all hardware facilities via software. BG/L ADE consists of a lightweight multithreaded coherence-managed kernel, runtime libraries, device drivers, system programming interfaces, and host-based cross-compilers and development tools. Through the use of runtime and compiletime options, BG/L ADE has the ability to utilize all or a subset of the hardware, ranging from a single PPC440 core with a simple SRAM memory subsystem VHDL model through the full 64K-node machine. This flexibility provides four important benefits. First, it allows functional units to be integrated into the chip simulation environment as they are developed. Second, because full chip simulations run at approximately three processor clocks (pclks) per second of wall-clock time on an event simulator and at approximately 600 pclks/s under Awan, it allows testing of specific functional units through directed tests without the need to initialize and configure other units. Third, certain functional units containing analog circuitry, such as the I/O capture units, cannot be accurately simulated with the Awan cycle simulator and are removed from the Awan simulation model. Fourth, because of the differences between the event and Awan simulation environments, different units can become available and ready for simulation at different times. For example, we can simulate with the L3 cache on an event simulator for several weeks while an Awan cycle-simulation embedded DRAM model is being developed. A simple BG/L ADE runtime flag disables L3 configuration, initialization, and use.

To assist software-driven system simulation, both internal and external logic was added to the simulation environments. Externally, the BG/L chip is enclosed in a testbench that provides required board-level functionality, including DDR chip models and clocks. This testbench also wraps network connections at the chip I/O pins, creating a $1 \times 1 \times 1$ torus network and externally wrapped collective links and global interrupts. To stress these network connections, varying network delays were added in the testbench. A version of this testbench provides a two-node simulation environment by enclosing two BG/L chips in a $2 \times 1 \times 1$ torus with all three collective links and global interrupts interconnected. Internally, a virtual universal asynchronous receiver transmitter (UART) serial port device is added to the BG/L in the processor local bus (PLB) MMIO space, which allows software to print trace, status, and debug information to the simulation logs.

BG/L ADE is used to build a suite of hundreds of test cases that exercise all functional units. Of these hundreds of tests, more than 160 are incorporated into a regression suite that is run continually during ASIC development and verification. Versions of many of these tests are used for hardware bring-up and are still in use as manufacturing test and full system on-site diagnostics test suites.

The verification environment also forms an important component of chip bring-up when hardware arrives. Test cases are easily moved back and forth between hardware and simulation environments to perform root-cause analysis of any unexpected hardware behavior. That allows us to find work-arounds, and we have even found the occasional subtle software bug when running at a six-order-of-magnitude increase in performance. For this to be effective, BG/L ADE and many of the test cases are carefully designed to achieve cycle-accurate reproducibility, including the ability to restart a test—or a particular iteration of a test—from a known state of all units, including the processors, memory system hierarchy, and networks. Of the few bugs found during bring-up, all had only a single-cycle window in a particular chip or memory system state during which they could occur. All of these are reproduced in relatively short order in simulation where the root cause is identified and the fix or work-around is identified and confirmed.

Test cases range from simple directed tests that focus on the correctness of specific DCRs to performance tests

of the full chip and to architectural studies. Directed software tests often incorporate external tool command language (TCL) scripts that are triggered when a particular device state is reached by software running on the processors. These scripts perturb that state in various ways by injecting errors, creating unusual situations, and verifying correct recovery in collaboration with the kernel or test-case software. The TCL scripting language is the interface between the user running the simulation and the event simulators ModelSim and NC-VHDL. SRAM, DDR, and latch bits are flipped. Network headers and packets are corrupted. State machines are forced into error states. Spurious device interrupts and machine checks are asserted. Transient and permanent errors are injected. In all of these cases, detection and (where possible) correction are verified.

Performance and architectural studies are at the other extreme of the test cases run under BG/L ADE in the simulation environments. Compared with the directed tests, which typically require several hours to complete, several of these studies run for days, even up to a week at a time. Performance studies measure memory system latencies and bandwidth; effectiveness of prefetching between L2 and L3 and between L3 and DDR; and instruction-path lengths for network interface injection and reception. In addition to correctness, architecture is considered in these studies. Feedback is provided to enhance device programming interfaces, interrupt delivery and handling, and MMIO layout. Interaction between the two PPC440 processors is also verified with regard to memory system coherence beyond the processor L1 caches, core-to-core interrupt delivery, and data and code-path locking.

To verify and measure the new double-hummer FPUs and their interaction with the memory system, selected computational kernels are extracted from parallel applications, including double-precision general matrix–matrix multiply (DGEMM), fast Fourier transform (FFT), and small molecular dynamics systems. Because these tests are run early in the BG/L chip development, there is time to design and incorporate memory system and instruction-set enhancements without excessive schedule impact. These tests also assist in compiler development, debugging, and back-end optimizer enhancements.

Results

Table 1 shows the number of fatal bugs found by each of the testbenches. The torus subsystem of the BG/L chip had the fewest bugs: 15 (13 + 2). The collective section had more bugs: 26 (19 + 7). The smaller number of bugs found in the torus design compared with the number found in the collective design can be attributed to the fact that subunit verification was performed on the torus and not on the collective. The subunit verification bugs that were found were not entered into the bug tracking system and consequently cannot be reported here. In the cases of both the collective and the torus, the unit-level tests found more bugs than the system-level tests. This is desirable because it is easier to debug failures at the unit level than the system level, and the difference can be explained by the unit-level tests starting earlier in the design cycle than the system-level tests. The memory subsystem had the most bugs, 72 (60 + 12), but this is to be expected, because it is the largest and most complicated subsystem on the BLC.

Table 1 Number of fatal bugs found by each testbench.

Testbench	*Number of fatal bugs found*
Torus: Random unit level (multinode)	13
Torus: Random system level	2
Collective: Random unit level (FB)	19
Collective: Random system level	7
Memory subsystem: Random system level	60
Memory subsystem: Code-driven system level	12

When BG/L hardware became available in the laboratory, the operating system and some application software were running within days. This is a great success considering the complexity of the chip. However, software running a stress test on BG/L hardware found two bugs that simulation had failed to catch. Oddly, these bugs were not hit by the operating system or application software, or at least had not yet been hit. Both bugs were in the L3 cache, and the conditions required to hit the bugs were very tightly constrained. Concerned that there might be more escapes, we created a random unit-level simulation for the L3. This random unit-level simulation allowed us to focus on the L3 without having to pass through the L2, as in the system-level simulation. We simulated both the original L3 design that contained the bugs and the corrected design. Although the simulation runs appeared to be substantially similar to the ones that had run in the random system-level simulation, this new simulation immediately detected the bugs in the older design, as well as two additional minor bugs in the new design. We did expect the L2 to have some filtering effect on our random system-level tests, but expected that this filtering would be overcome by the large number of machine hours of random stimulus.

One minor collective network interface bug was found in the laboratory relating to arithmetic logic unit (ALU) overflow error reporting. This bug resulted in some spurious overflow interrupts, but was easily worked around.

Table 2 Code-driven and random system-level cost–performance comparison.

Simulation type	*Simulator*	*System*	*Cost ($)*	*Wall-clock time*	*Processor cycles/s*	*$/cycles/s*
		Code-driven				
Code-driven system-level with full processor model	ModelSim	IBM RS/6000* 44P Model 270 POWER3*-II/AIX* 375 MHZ	10K	9 hours 17 minutes	3	3,333
Code-driven system-level with full processor model	Mesa	IBM RS/6000 44P Model 270 POWERr3-II/AIX 375 MHz	10K	28 minutes	50	200
Code-driven system-level with full processor model	Awan	Awan	200K	2 minutes	603	332
		Random				
Random system-level with bus functional model	NC-VHDL	IBM xSeries* 305 Pentium** 4/Linux** 2.6 GHz	1K		23	43

Unexpectedly, the code-driven system simulation found two bugs in the PPC440 core and one bug in the full processor model. This was surprising, considering that these were supposed to be pre-verified drop-in components.

A comparison of cost as a function of performance for the different simulators in use in the BLC running exactly the same code-driven system-level simulation is shown in **Table 2**. The major distinction between the rows of code-driven data is the simulation engine. The cost of software is not included in the cost column. From a project standpoint, the cost of ModelSim, NC-VHDL, and Mesa is an equivalent constant. The wall-clock time is large, and the $/cycle/s metric is large. However, if Mesa instead of ModelSim is run on the same hardware platform, the IBM RS/6000 Model 270, the $/cycle/s metric compares well with that of Awan. Whether one should use the Mesa system or the Awan system would depend on whether the increased wall-clock time of the Mesa system would be acceptable, considering its lesser cost compared with Awan. The software running in this simulation is a "Hello world"[1] program running on top of the operating system. If the goal is to run a substantial software application (much more substantial than Hello world) and the budget permits, Awan would be the preferred platform.

The cost/performance numbers for the random system-level simulation are also shown in Table 2. The major distinction between the code-driven and random simulations is that they are completely different simulations. The code-driven simulation uses a full processor model, whereas the random simulation uses a bus-functional model. Thus, there are far more simulation events per cycle in the code-driven simulation than in the random simulation, and this distinction must be taken into account when comparing the two.

[1]A program written by beginning students that simply writes "Hello world" to the screen.

In the past we compared the performance of ModelSim and NC-VHDL and, as one would expect, we found them to be roughly equal. Accordingly, we ignore the different event simulators, ModelSim and NC-VHDL. The real interest in comparing the two types of simulations is the cost of the xSeries 305 system compared with the RS/6000 Model 270. If one assumes the performance of these two systems to be roughly equal, and this is not an unreasonable approximation, and the cost of the xSeries 305 machine is substituted for the cost of the Model 270 in the table, this brings the $/cycles/s metric for ModelSim down to 333, far better than the ModelSim number and comparable with that of Mesa and Awan. Making the same substitution for the Mesa system brings the $/cycles/s metric from 200 down to 20, which is an order of magnitude better than the $/cycle/s of the Awan system. The Model 270 is a 64-bit system, whereas the xSeries 305 is a 32-bit system. Comparing the approaches also shows that the random simulation with the bus-functional model is an order of magnitude faster than the code-driven simulation with the full processor model, but this is expected, for the reasons discussed above. We also report that we encountered no problems running any of the simulation tools on Linux.

In the future, instead of VHDL, the random system-level testbench will be written in C++, SystemC [8], or an equivalent IBM internally developed package. Given our experience, it is felt that the task of writing complex behavioral software for a testbench can be done much more easily in C++ than VHDL, regardless of C++ simulator interface portability issues.

Summary

The complexity of system-on-a-chip logic applications such as the BG/L chip defines the challenges that must be met by state-of-the-art verification methodologies.

The high development and fabrication costs of these systems necessitate the use of sophisticated verification methodologies to help ensure successful first-pass silicon. To meet these challenges, we developed a verification strategy in which different types of verification are used together in specific ways to complement one another. This multifaceted approach forms a robust verification of complex systems. Our verification strategy employs engineered directed-test cases, unit-level simulation, and formal protocol verification to augment our core random system simulation, in which we run prodigious numbers of simulation cycles on a continuous stream of automatically generated new and varied test cases. The "art" in an effective random simulation is to direct and control the operational focus in the test space so that simulation cycles are not wasted, and to know when the simulation space has been sufficiently explored. The verification strategy described in this paper is validated by the first-pass success of the BG/L chip.

We successfully demonstrated that verifying a large SoC chip using a Linux-based commodity personal computer farm is cost-effective. Our verification strategy led other verification efforts within IBM to use Linux-based PCs for a simulation farm.

Acknowledgment

The Blue Gene/L project has been supported and partially funded by the Lawrence Livermore National Laboratory on behalf of the United States Department of Energy under Lawrence Livermore National Laboratory Subcontract No. B517552.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Linus Torvalds, Mentor Graphics Corporation, Cadence Design Systems, Inc., or Intel Corporation in the United States, other countries, or both.

References

1. 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, and B. Wile, "Functional Verification of the POWER4 Microprocessor and POWER4 Multiprocessor Systems," *IBM J. Res. & Dev.* **46**, No. 1, 53–76 (January 2002).
2. See *http://www.chips.ibm.com*.
3. J. Bergeron, *Writing Testbenches: Functional Verification of HDL Models*, Kluwer Academic Publishers, Boston, 2000.
4. J. Darringer, E. Davidson, D. J. Hathaway, B. Koenemann, M. Lavin, J. M. Morell, K. Rahmat, W. Roesner, E. Schanzenbach, G. Tellez, and L. Trevillyan, "EDA in IBM: Past, Present, and Future," *IEEE Trans. Computer Aided Design Integr. Circuits & Syst.* **19**, No. 12, 1476–1497 (December 2000).
5. N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue Gene/L Torus Interconnection Network," *IBM J. Res. & Dev.* **49**, No. 2/3, 265–276 (2005, this issue).
6. D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang, "Protocol Verification as a Hardware Design Aid," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1992, pp. 522–525.
7. M. E. Giampapa, R. Bellofatto, M. A. Blumrich, D. Chen, A. Gara, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, B. J. Nathanson, B. D. Steinmacher-Burow, M. Ohmacht, V. Salapura, and P. Vranas, "Blue Gene/L Advanced Diagnostics Environment," *IBM J. Res. & Dev.* **49**, No. 2/3, 319–331 (2005, this issue).
8. See *http://www.systemc.org*.

Received June 1, 2004; accepted for publication July 5, 2004; Internet publication April 12, 2005

Michael E. Wazlowski *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mew@us.ibm.com)*. Dr. Wazlowski is a Research Staff Member; he is responsible for high-performance memory system architecture and design and has led the verification efforts for two ASICs. He received a B.S. degree in computer engineering from the University of Massachusetts at Amherst in 1990 and M.S. and Ph.D. degrees in electrical sciences from Brown University in 1992 and 1996, respectively. Dr. Wazlowski received an IBM Outstanding Technical Achievement Award for his contributions to IBM Memory Expansion Technology (MXT). He holds several patents. His research interests include computer architecture, memory systems, and ASIC design. He is currently working on cache and memory systems. Dr. Wazlowski is a member of the IEEE.

Narasimha R. Adiga *IBM Engineering and Technology Services, Golden Enclave, Airport Road, Bangalore 560 017, India (anarasim@in.ibm.com)*. Mr. Adiga is a Staff Research and Development Engineer. In 1998 he received a B.E. degree from Karnataka Regional Engineering College, India. He subsequently joined IBM, where he has worked on the development of the PCI-PCIX exerciser/analyzer card and verification of the torus with a single-node BG/L compute chip for Blue Gene/L. Mr. Adiga is currently working on memory interface controller designs for future systems.

Daniel K. Beece *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (beece@us.ibm.com)*. Dr. Beece is a Research Staff Member at the IBM Thomas J. Watson Research Center, where he has worked in the area of VLSI system design and verification. He has received several IBM Outstanding Technical Achievement and Outstanding Innovation Awards and patents for his contributions in this area. He received a B.S. degree in physics from Cornell University in 1975 and a Ph.D. degree in physics from the University of Illinois in 1983. His research interests include VLSI design tools, design methodology, system verification, languages, and timing. Dr. Beece is currently working on system-level simulation and verification.

Ralph Bellofatto *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ralphbel@us.ibm.com)*. Mr. Bellofatto is a Senior Software Engineer. He has been responsible for various aspects of hardware system verification and control system programming on the Blue Gene/L project. He received B.S. and M.S. degrees from Ithaca College in 1979 and 1980, respectively. He has worked as a software engineer in a variety of industries. Mr. Bellofatto's interests include computer architecture, performance analysis and tuning, network architecture, ASIC design, and systems architecture and design. He is currently working on the control system for Blue Gene/L.

Matthias A. Blumrich *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (blumrich@us.ibm.com)*. Dr. Blumrich is a Research Staff Member in the Server Technology Department. He received a B.E.E. degree from the State University of New York at Stony Brook in 1986, and M.A. and Ph.D. degrees in computer science from Princeton University in 1991 and 1996, respectively. In 1998 he joined the IBM Research Division, where he has worked on scalable networking for servers and the Blue Gene supercomputing project. Dr. Blumrich is an author or coauthor of two patents and 12 technical papers.

Dong Chen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (chendong@us.ibm.com)*. Dr. Chen is a Research Staff Member in the Exploratory Server Systems Department. He received a B.S. degree in physics from Peking University in 1990, and M.A., M.Phil., and Ph.D. degrees in theoretical physics from Columbia University in 1991, 1992, and 1996, respectively. He continued as a postdoctoral researcher at the Massachusetts Institute of Technology from 1996 to 1998. In 1999 he joined the IBM Server Group, where he worked on optimizing applications for IBM RS/6000* SP systems. In 2000 he moved to the IBM Thomas J. Watson Research Center, where he has been working on many areas of the Blue Gene/L supercomputer and collaborating on the QCDOC project. Dr. Chen is an author or coauthor of more than 30 technical journal papers.

Marc Boris Dombrowa *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (dombrowa@us.ibm.com)*. Mr. Dombrowa received his Dipl.-Ing. degree in electrical engineering from the University of Hannover, Germany, in 1997. He was a very large scale integration (VLSI) designer at the IBM VLSI Laboratory in Boeblingen, Germany, from 1997 to 1998, performing memory design verification and synthesis on S/390* Enterprise memory systems. From 1998 to 2000 he was assigned to the S/390 Server Division at the IBM Poughkeepsie facility to perform custom circuit design. He moved to Blue Gene/L cellular systems chip development in 2001 and has been responsible for the high-level design, synthesis, timing, and verification of the test interface of the Blue Gene/L compute chips as well as design-for-testability transformation for the entire chip, clock-tree verification, and simulation setup for instruction program load for the chip verification teams. Mr. Dombrowa received an IBM Outstanding Achievement Award in 1998 for his S/390 contributions. He is coinventor of one patent. His research interests include computer architecture, design for test, system bring-up, diagnostics, and ASIC design. Mr. Dombrowa is currently working on the manufacturing diagnostic software as well as the system-level rack diagnostic test suite and bring-up for the Blue Gene/L cluster.

Alan Gara *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (alangara@us.ibm.com)*. Dr. Gara is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in physics from the University of Wisconsin at Madison in 1986. In 1998, Dr. Gara received the Gordon Bell Award for the QCDSP supercomputer in the most cost-effective category. He is the chief architect of the Blue Gene/L supercomputer. Dr. Gara also led the design and verification of the Blue Gene/L compute ASIC as well as the bring-up of the Blue Gene/L prototype system.

Mark E. Giampapa *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (giampapa@us.ibm.com)*. Mr. Giampapa is a Senior Engineer in the Exploratory Server Systems Department. He received a B.A. degree in computer science from Columbia University. He joined the IBM Research Division in 1984 to work in the areas of parallel and distributed processing, and has focused his research on distributed memory and shared memory parallel architectures and operating systems. Mr. Giampapa has received three IBM Outstanding Technical Achievement Awards for his work in distributed processing, simulation, and parallel operating systems. He holds 15 patents, with several more pending, and has published ten papers.

Ruud A. Haring *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (ruud@us.ibm.com)*. Dr. Haring is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received B.S., M.S., and Ph.D. degrees in physics from Leyden University, the Netherlands, in 1977, 1979, and 1984, respectively. Upon joining IBM in 1984, he initially studied surface science aspects of plasma processing. Beginning in 1992, he became involved in electronic circuit design on both microprocessors and application-specific integrated circuits (ASICs). He is currently responsible for the synthesis, physical design, and test aspects of the Blue Gene chip designs. Dr. Haring has received an IBM Outstanding Technical Achievement Award for his contributions to the z900 mainframe, and he holds several patents. His research interests include circuit design and optimization, design for testability, and ASIC design. Dr. Haring is a Senior Member of the IEEE.

Philip Heidelberger *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (philiph@us.ibm.com)*. Dr. Heidelberger received a B.A. degree in mathematics from Oberlin College in 1974 and a Ph.D. degree in operations research from Stanford University in 1978. He has been a Research Staff Member at the IBM Thomas J. Watson Research Center since 1978. His research interests include modeling and analysis of computer performance, probabilistic aspects of discrete event simulations, parallel simulation, and parallel computer architectures. He has authored more than 100 papers in these areas. Dr. Heidelberger has served as Editor-in-Chief of the *ACM Transactions on Modeling and Computer Simulation*. He was the general chairman of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS) Performance 2001 Conference, the program co-chairman of the ACM SIGMETRICS Performance 1992 Conference, and the program chairman of the 1989 Winter Simulation Conference. Dr. Heidelberger is currently the vice president of ACM SIGMETRICS; he is a Fellow of the ACM and the IEEE.

Dirk Hoenicke *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (hoenicke@us.ibm.com)*. Mr. Hoenicke received a Dipl. Inform. (M.S.) degree in computer science from the University of Tuebingen, Germany, in 1998. Since then, he has worked on a wide range of aspects of two prevalent processor architectures: ESA/390 and PowerPC. He is currently a member of the Cellular Systems Chip Development Group, where he focuses on the architecture, design, verification, and implementation of the Blue Gene system-on-a-chip (SoC) supercomputer family. In particular, he was responsible for the architecture, design, and verification effort of the collective network and defined and implemented many other parts of the BG/L ASIC. Mr. Hoenicke's areas of expertise include high-performance computer systems and advanced memory and network architectures, as well as power-, area-, and complexity-efficient logic designs.

Ben J. Nathanson *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (bjnath@us.ibm.com)*. Mr. Nathanson joined the IBM Research Division in 1985 and has worked on the parallel computers RP3, Vulcan, SP1, SP2, and Blue Gene/L. He received IBM Outstanding Technical Achievement Awards for hardware contributions to SP1 and SP2 and Research Division Awards for RP3 bring-up and verification work on memory compression hardware. Mr. Nathanson holds M.S. and B.S. degrees in electrical engineering from Columbia University and is a member of Tau Beta Pi and Eta Kappa Nu. His current focus is hardware verification.

Martin Ohmacht *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mohmacht@us.ibm.com)*. Dr. Ohmacht received his Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from the University of Hannover, Germany, in 1994 and 2001, respectively. He joined the IBM Research Division in 2001 and has worked on memory subsystem architecture and implementation for the Blue Gene project. His research interests include computer architecture, design and verification of multiprocessor systems, and compiler optimizations.

Robert Sharrar *IBM Systems and Technology Group, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (rsharrar@us.ibm.com)*. Mr. Sharrar is a Senior Engineer in server development. His responsibilities include work as a server farm administrator and verification engineer. He received a B.S. degree in electrical engineering from Lehigh University in 1984, joining IBM that same year, and has since held several positions in personal computer and server development. Mr. Sharrar has received three IBM Outstanding Technical Achievement Awards and a Division Excellence Award. He is currently working on a PCI bridge and memory controller ASIC for storage applications.

Sarabjeet Singh *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (sarabj@us.ibm.com)*. Mr. Singh is a Senior Research and Development Engineer with the Engineering and Technology Services Division of IBM, currently on assignment at the IBM Thomas J. Watson Research Center. He received a B.Tech. degree in electrical engineering from the Indian Institute of Technology in 1996 and subsequently joined IBM, where he has worked on various research projects involving all aspects of ASIC and system-on-a-chip (SoC) design. Over the past seven years he has worked on many CMOS technologies (Blue Gene/L in Cu-11 technology), up to 700-MHz clock designs, asynchronous logic design, and Small Computer System Interface (SCSI) drive controllers to HPC systems. Mr. Singh is currently working on memory subsystem microarchitecture for an HPC system based on the STI cell.

Burkhard D. Steinmacher-Burow *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (steinmac@us.ibm.com)*. Dr. Steinmacher-Burow is a Research Staff Member in the Exploratory Server Systems Department. He received a B.S. degree in physics from the University of Waterloo in 1988, and M.S. and Ph.D. degrees from the University of Toronto in 1990 and 1994, respectively. He subsequently joined the Universitaet Hamburg and then the Deutsches Elektronen-Synchrotron to work in experimental particle physics. In 2001, he joined IBM at the Thomas J. Watson Research Center and has since worked in many hardware and software areas of the Blue Gene research program. Dr. Steinmacher-Burow is an author or coauthor of more than 80 technical papers.

R. Brett Tremaine *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (afton@us.ibm.com)*. Mr. Tremaine is a Senior Technical Staff Member. He is responsible for commercial server and memory hierarchy architecture, design, and ASIC implementation. He worked at the IBM Federal Systems Division in Owego, New York, before joining the IBM Thomas J. Watson Research Center in 1989. Mr. Tremaine has led several server architecture and ASIC design projects, many with interdivisional relationships, and he has received three IBM Outstanding Technical Achievement Awards

and several Division Excellence Awards for his contributions. He received a B.S. degree in electrical engineering from Michigan Technological University in 1982 and an M.S. degree in computer engineering from Syracuse University in 1988. He holds 11 patents and has five patents pending. Mr. Tremaine has published a number of technical papers and is a member of the IEEE.

Michael Tsao *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mtsao@us.ibm.com)*. Dr. Tsao is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received B.S.E.E, M.S.E.C.E, and Ph.D. degrees from the Electrical and Computer Engineering Department of the Carnegie-Mellon University in 1977, 1979, and 1983, respectively. He joined IBM at the Thomas J. Watson Research Center in 1983 and has worked on various multiprocessor projects including RP3, GF11, Vulcan, SP1, SP2, MXT, and BG/L. Dr. Tsao is currently working on cache chips for future processors.

Arun R. Umamaheshwaran *IBM Engineering and Technology Services, Golden Enclave, Airport Road, Bangalore 560 017, India (arun_mahesh@in.ibm.com)*. Mr. Umamaheshwaran is a Research and Development Engineer. He received a B.Tech. degree in electronics and communication engineering from the University of Calicut, India. His efforts are directed toward the verification of ASICs; he was involved in the verification of two high-performance SoCs and was recognized for his efforts. Mr. Umamaheshwaran's areas of interest include digital system design, ASIC verification, and bus architectures.

Pavlos Vranas *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (vranasp@us.ibm.com)*. Dr. Vranas is a Research Staff Member in the Deep Computing Systems Department at the IBM Thomas J. Watson Research Center. He received his B.S. degree in physics from the University of Athens in 1985, and his M.S. and Ph.D. degrees in theoretical physics from the University of California at Davis in 1987 and 1990, respectively. He continued research in theoretical physics as a postdoctoral researcher at the Supercomputer Computations Research Institute, Florida State University (1990–1994), at Columbia University (1994–1998), and at the University of Illinois at Urbana–Champaign (1998–2000). In 2000 he joined IBM at the Thomas J. Watson Research Center, where he has worked on the architecture, design, verification, and bring-up of the Blue Gene/L supercomputer and is continuing his research in theoretical physics. Dr. Vranas is an author or coauthor of 59 papers in supercomputing and theoretical physics.