The IBM Blue Gene/Q Compute Chip

Ruud A. Haring
Martin Ohmacht
Thomas W. Fox
Michael K. Gschwind
David L. Satterfield
Krishnan Sugavanam
Paul W. Coteus
Philip Heidelberger
Matthias A. Blumrich
Robert W. Wisniewski
Alan Gara
George Liang-Tai Chiu
IBM T.J. Watson Research Center

Peter A. Boyle
University of Edinburgh

Norman H. Christ
Columbia University

Changhoan Kim
RIKEN BNL Research Center

Blue Gene/Q aims to build a massively parallel high-performance computing system out of power-efficient processor chips, resulting in power-efficient, cost-efficient, and floor-space-efficient systems. Focusing on reliability during design helps with scaling to large systems and lowers the total cost of ownership. This article examines the architecture and design of the Compute chip, which combines processors, memory, and communication functions on a single chip.

Like its previous generations, Blue Gene/L (BG/L)[1] and Blue Gene/P (BG/P),[2] the Blue Gene/Q (BG/Q) system is optimized for price performance, energy efficiency, and reliability in large-scale scientific applications. The system also aims to expand into applications with commercial and industrial impact, such as analytics. For the heart of the system, the Blue Gene/Q Compute (BQC) chip, this translates into design requirements to optimize power efficiency in terms of floating-point operations per second, per Watt (flops/W); to optimize the chip's networking and connectivity features; and to optimize reliability features such as redundancy, the usage of error correction and detection, and the minimization of sensitivity to soft errors.

The BQC chip is an 18.96 × 18.96 mm chip in IBM's Cu-45HP (45 nm silicon on insulator [SOI]) application-specific integrated circuit (ASIC) technology, and it contains about 1.47×10^9 transistors. The design integrates processors, a memory subsystem, and a networking subsystem on a single chip. A die photograph of the chip shows 18 processor units (PU00 to PU17) surrounding a large Level-2 (L2) cache that occupies the center of the chip (see Figure 1). The processor units are based on the PowerPC A2 processor core, with BG/Q-specific additions: a single-instruction, multiple-data (SIMD) quad floating-point unit; a Level-1 (L1) prefetching unit; and a wake-up unit.

The processor units function as 16 user processors plus one processor for operating system services. A redundant processor increases manufacturing yield. The processor units connect to the L2 cache via a crossbar switch in the middle of the chip.

The 32-Mbyte L2 cache is built from 16 2-Mbyte slices (L200 to L215), using embedded DRAM (eDRAM) for dense, power-efficient, and high-bandwidth storage. The L2 cache is multiversioned to support speculative execution and transactional memory. In addition, the L2 cache supports atomic operations. L2 cache misses are handled by dual on-chip memory controllers (MC0 and MC1) that drive the double data rate type three (DDR3) integrated I/O blocks (PHYs) located on the left and right

Published by the IEEE Computer Society

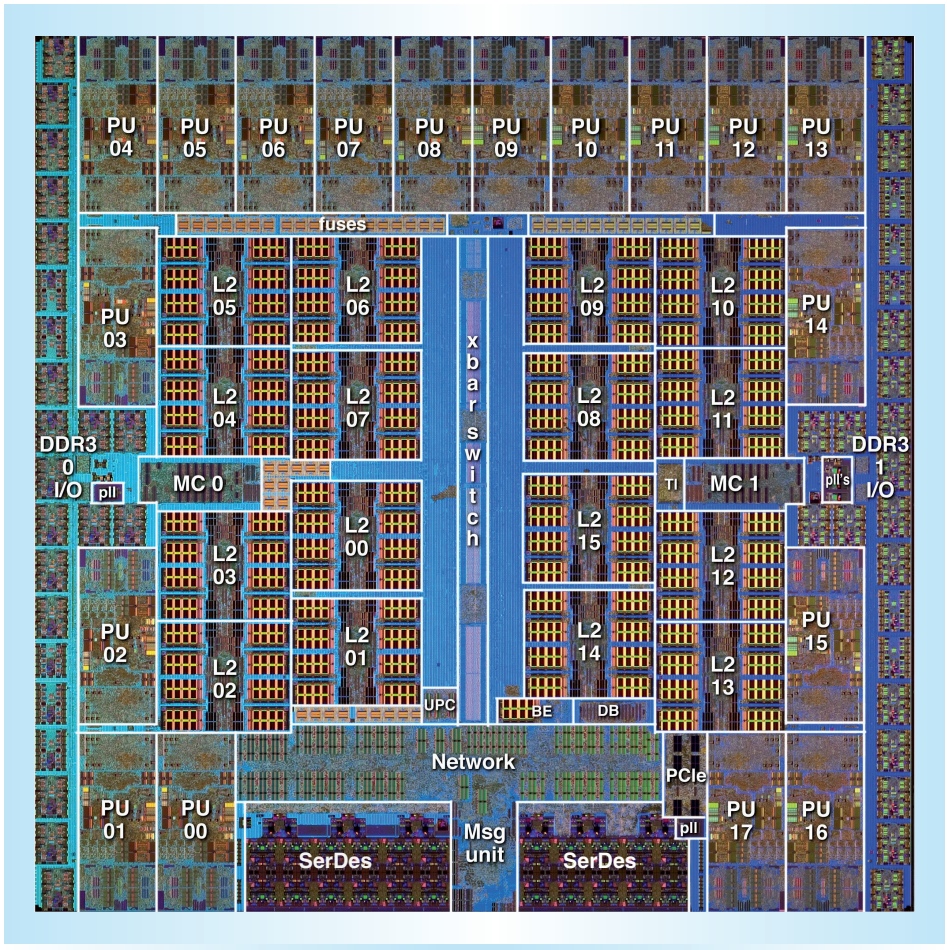

Figure 1. A die photograph of the Blue Gene/Q Compute (BQC) chip. The photograph shows a large Level-2 (L2) cache in the center of the chip, surrounded by 18 processor units based on the PowerPC A2 processor core. Integrated DDR3 memory controllers are on the left and right sides, and the chip-to-chip communication (network) logic is at the bottom side of the chip.

edges of the chip. The network and messaging unit occupies the chip's south end.

BQC chip design

The BQC chip is optimized for floating-point performance, power efficiency, chip-to-chip communication bandwidth, and reliability. The multithreaded multicore chip design contains several innovative hardware features that support efficient software execution.

Processor cores

The BQC chip's processor core is an augmented version of the A2 processor core used on the IBM PowerEN chip.[3] The A2 processor core implements the 64-bit Power instruction set architecture (Power ISA) and is optimized for aggregate throughput. The A2 is four-way simultaneously multithreaded and supports two-way concurrent instruction issue: one integer, branch, or load/store instruction and one floating-point instruction. Within a thread, dispatch, execution, and completion are in order.

The A2 core's L1 data cache is 16 Kbyte, eight-way set associative, with 64-byte lines. The 32-byte-wide load/store interface has sufficient bandwidth to support the quad floating-point unit. The L1 instruction cache is 16 Kbyte, four-way set associative.

We resynthesized the processor core to run at 1.6 GHz at a reduced voltage

(0.8 V nominal) versus the original A2 design point. The lower-voltage operation reduces both active and leakage power.

Quad floating-point processing unit

The Quad Processing eXtension (QPX) to the Power ISA is an architectural definition and set of instructions for floating-point processing in a microprocessor core, developed specifically for BG/Q. The BQC chip Quad floating-point Processing Unit (QPU) implements the QPX instruction set, as well as the scalar floating-point instructions from the Power ISA. The QPU replaces the A2 core's scalar floating-point unit as used on the PowerEN chip.

QPX instruction set architecture. The QPX architecture's computational model is a vector SIMD model, with four execution slots and a register file containing 32 256-bit quad processing registers (QPRs). We can envision each of the 32 registers as containing four 64-bit elements, whereby each execution slot operates on one vector element.

The QPX architecture contains a full set of arithmetic instructions, including fused multiply-add (FMA) instructions, a new set of permute instructions for data shuffling across execution slots, comparison, conversion, and move instructions. In addition, the QPX architecture contains a novel set of complex arithmetic instructions, which go beyond the traditional repertoire of SIMD lane-oriented computation by allowing execution slots to operate on data from adjacent slots. A collection of load/store instructions for complex data complements the complex arithmetic instructions.

Certain QPX store instructions provide a novel mechanism for the detection and indication of numerically exceptional conditions at the store interface. A Store Indicate NaN or Store Indicate Infinity exception occurs when the source operand of a Store with Indicate instruction in any of the four execution slots contains a NaN or Infinity value.

Scalar floating-point load instructions, as defined in the Power ISA, cause a replication of the source data across all elements of the target register.

Scalar floating-point move, arithmetic, rounding and conversion, comparison, and select instructions, defined in the Power ISA, are executed in execution slot 0, and operate on element 0 QPRs, which serve as both the scalar floating-point registers (FPRs) for scalar instructions and the element 0 QPRs for vector instructions.

QPU design. Each execution slot within the QPU embodies a fused multiply-add data flow (MAD) pipeline, which is the machinery that calculates the mathematical equation $\pm [(A * C) \pm B]$, where A, B, and C are the operands defined by the Power and QPX ISAs. The QPU instantiates four copies of this data flow, creating a four-way SIMD floating-point microarchitecture, as Figure 2 shows.

Each execution slot has a register file holding one element of each vector register. The register file contains the state of four threads, to support the processor core's four-way multithreading capabilities. Physically, the 2W4R register file is built out of two identical copies of a 2W2R register file.

Arithmetic operations in the MAD pipeline have a latency of six cycles for back-to-back dependent operations. Data shuffling, select, and move instructions have a latency of two cycles. Some dependent operations involving complex instructions can incur an extra cycle of latency. All data paths are fully pipelined, allowing a new instruction to be issued every cycle.

To protect the QPU's large architected register state from soft errors, we use parity protection on interleaved groups of bits in the register file. This will detect soft-error upsets, including multibit hits. When a bit-parity error is detected, the correction mechanism exploits the redundancy provided by the two identical copies of the register file. When a corrupted data value is detected in one of the operands, a recovery operation is performed, in which data from the good copy overwrites the corrupted register file entry.

The QPU's peak performance is four FMA operations (that is, eight double-precision floating-point operations) per cycle. At 1.6 GHz, an A2 processor core and QPU therefore has a peak performance of 12.8 Gflops. The aggregate peak performance of the 16 user-processors on the BQC chip is 204.8 Gflops.

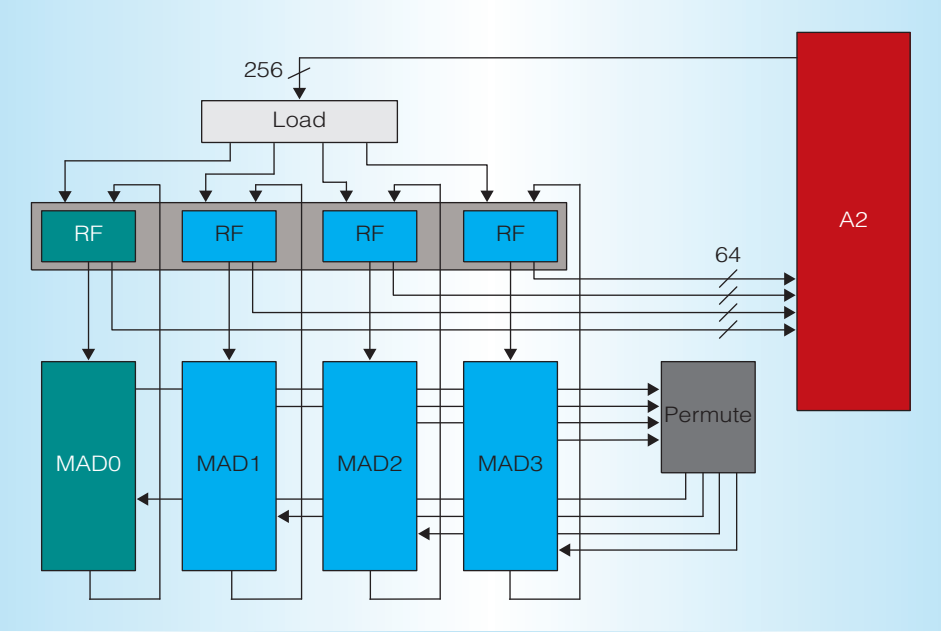

Figure 2. The BG/Q Quad floating-point Processing Unit (QPU) consists of a Quad Processing Register file, four parallel double-precision floating-point multiply-add (MAD) pipelines, and a Permute unit.

L1 prefetch unit

Another BG/Q-specific augmentation to the A2 processor core is the L1 prefetch unit. The L1P implements two types of automatic data prefetching to try to hide the latency of accesses to the L2 cache and DDR memory. Data is prefetched as 128-byte L2 cache lines (twice the L1 cache line length) and stored in a buffer capable of holding 32 prefetch lines. Full coherency is maintained for all prefetched data lines.

The first type of prefetching is an enhanced version of the stream prefetching implemented in BG/L and BG/P. Stream prefetching hardware identifies a sequence of loads from increasing, contiguous memory locations, and then prefetches additional contiguous data in this sequence.

The second type of prefetching, called list prefetching, is designed to anticipate a sequence of loads made from an arbitrary series of addresses, when that sequence of load addresses is accessed more than once.

Stream prefetching. For L1 misses, the two previous Blue Gene machines incorporate hardware that automatically detects and prefetches data accessed in a sequence of increasing contiguous memory addresses, a classic form of prefetching.[4] The BG/L chip prefetches one 128-byte line and a subsequent 128-byte line when a second L1 miss address lies within either line. Thus, BG/L prefetches streams of depth two and can maintain seven active streams within the 15 lines that the prefetch buffer can store. The BG/P chip implements a more flexible system in which L1 misses to increasing prefetched addresses can trigger the prefetch of two consecutive 128-byte data lines, thus supporting streams of depth three in the case of continued, uninterrupted access to that stream.

The increased memory latency, relative to clock period, of the BG/Q memory system and the demands of four threads per core, each potentially accessing multiple streams, results in increased pressure on prefetch buffer storage in fixed-depth prefetch schemes. To minimize the prefetch storage requirements, we designed a powerful adaptive stream-prefetching engine. The engine assigns a variable depth to each established stream, up to a maximum of 16 streams. The largest supported depth is eight. Each stream begins with a default preloaded

depth that is typically small (for example, two). When a stream is first established, data is prefetched up to this initial depth. Subsequent prefetches are triggered when an L1 miss address lies within the stream. Such hits on data that have not yet been retrieved from memory trigger an adaptation event that increases that stream's depth. That depth is stolen from the least recently hit stream, keeping the total depth for all active streams at or below the prefetch buffer's 32-line capacity. With this demand-driven adaptation, buffer thrashing is reduced when many streams are active. The device will adapt to operate efficiently over a range of workloads, from a single stream from one thread to multiple streams and multiple threads.

List prefetching. The second prefetch scheme targets repetitive, deterministic code, such as that in iterative solvers, in which the same sequence of addresses is accessed again and again. This new method requires the beginning and end of a repeating code segment to be identified in the application program. Once activated, each list prefetching engine (one per thread) will capture the subsequent series of L1 miss addresses. The recording of addresses is controllable with memory page granularity. The captured miss addresses are packed in a list written to memory.

When this repetitive portion of code is later executed again, the recorded sequence is read from memory and the list of addresses is used to prefetch the needed data. The loading of this recorded address sequence and the subsequent data prefetching must be synchronized with code execution so the data is prefetched in time to be used, but not so far in advance to waste space in the prefetch buffer. We accomplish this by stalling the processor at the beginning of repetitive code until the first of the recorded addresses have been loaded into the prefetch engine. The list-prefetch engine then matches the L1 misses with these recorded addresses and will start prefetching the data at list addresses that follow the matched address, up to a preloaded depth. This address matching and data prefetching continues until a terminating end-of-list marker is reached in the list of recorded addresses or until the programmed maximum length of a list is reached.

Further refinements increase the tolerance to variations that are expected in a multiprocessor environment. Perhaps most important is the address-matching algorithm's ability to compare each L1 miss against not only the next unmatched recorded address, but also up to seven subsequent addresses. Thus, if one or more recorded L1 misses do not reoccur, they can be skipped over without losing synchronization. Furthermore, support is provided to continue the recording of L1 miss addresses on subsequent iterations through the code. This new list can then replace the list recorded earlier, allowing a self-healing evolution as the pattern of L1 misses changes.

Finally, added hardware control lets us implement important software features. When the application encounters a segment of code with nonrepetitive data access (such as a table look-up), it can pause and later resume the list prefetching. Similarly, a function call can stop list prefetching and save the state of the prefetching engine when a subroutine is entered and restore that state and continue prefetching when the function returns, allowing standard and modular code to exploit this list prefetching.

WakeUp unit

Each processor core has its own external WakeUp unit, whose main purpose is to increase overall application performance by reducing thread-to-thread interactions arising from software blocked in a spin loop or similar blocking polling loop. Because the processor core has four threads but performs at most one integer instruction and one floating-point instruction per processor cycle, a thread in a polling loop takes cycles from the other threads. This cost is highest if the polled variable is L1-cached, because the loop's frequency is highest. In general, degradation will occur whenever the polling thread competes for hardware resources with other threads.

Implementing a wait procedure using the WakeUp unit will incur less degradation than a polling loop implemented in software. Upon entering a wait, software writes a base address and enable mask for the polled address range to a wait address compare (WAC)

register in the WakeUp unit. Subsequently, an instruction is executed to pause the corresponding hardware thread. When any of the other 67 hardware threads, the message unit, or the PCI Express unit writes a matching address, the WakeUp unit will drive the thread's pause/enable signal to resume execution. The resumed program then reads the data from the matching address range. If the desired condition has indeed been reached, the polling loop is exited. Otherwise, the program again configures the WakeUp unit and pauses itself.

The WakeUp unit has 12 WACs; each WAC snoops all writes on the chip by snooping the local store operations and incoming L1 invalidates. The WakeUp unit can also resume a thread using interrupts (such as network packet arrivals). Each of the core's four hardware threads has a status and enable register in the WakeUp unit, so that software can select the WACs and interrupts used to resume.

The WakeUp unit also outputs a logical-OR of a programmable subset of the 12 WACs. System software can use this function to generate an external interrupt to the A2. For example, it can use this interrupt to recognize a stack overflow.

Processor unit redundancy

The A2 core, QPU, L1P, and WakeUp units comprise a processor unit (PU), which is replicated 18 times on the chip (see Figure 1). Because user and system software use only 17 PUs, one of these PUs is redundant. The redundant PU increases manufacturing yield. If we find a defect in a PU during manufacturing test (at wafer, module, card, or system level), we can quiesce the defective PU and activate the redundant PU. PU sparing information is carried with the physical chip in eFuses or in an on-card EEPROM, and is activated on the chip at boot time. PU sparing is transparent to the end user—that is, system and user software will see 17 contiguous operational processors, logically numbered 0 through 16, irrespective of which physical processor (if any) has been spared out.

Crossbar

The crossbar switch is the central connection structure between all the PUs (specifically, the L1P units), the L2 cache, the networking logic, and various low-bandwidth units on the chip. The crossbar and L2 cache are clocked at half the processor frequency, with a separate 800 MHz clock grid. The crossbar has 22 master ports (18 PUs, three network logic ports, and a DevBus port used by PCI Express) and 18 slave ports (16 L2 slices, the network logic, and the DevBus slave). The crossbar actually consists of three switches, one each for request traffic, response traffic, and invalidation traffic. The response switch has a 32-byte-wide read data path from all slave devices, such as L2 caches, to all master devices, such as L1P units. The aggregate maximum read bandwidth from all L2 slices is 409.6 Gbytes/s. The write data path is 16 bytes wide, delivering an aggregate write bandwidth of 153.6 Gbytes/s (under simultaneous reads).

L2 cache

All processor cores share the L2 cache. To provide sufficient bandwidth, the cache is split into 16 slices. Each slice is 16-way set associative, operates in write-back mode, and has 2-Mbyte capacity. Physical addresses are scattered across the slices via a programmable hash function to achieve uniform slice utilization.

The L2 cache serves as point of coherence as well as control for atomic accesses using the PowerISA's Load and Reserve/Store Conditional (larx/stcx) instructions.

Given the large number of concurrent threads and the target of high aggregate performance, we designed the L2 cache to provide hardware assist capabilities to accelerate sequential code as well as thread interactions. Specifically, the L2 cache provides memory speculation support and atomic memory update operations.

Memory speculation: transactional memory (TM) and speculative execution (SE). The BQC chip implements hardware support for memory speculation in the form of a multiversioning L2 cache. During memory speculation, the L2 cache tracks state changes caused by speculative threads and keeps them separate from the main memory state. At the end of a speculative code section, the modifications can either be reverted (invalidate) or

made permanent (commit). This allows programmers to write code sections without having to reason in advance about the correctness of concurrent execution. The L2 cache will track access conflicts between threads, letting a speculation runtime system reason about the correctness of execution. This hardware support enables transactional memory (TM) and speculative execution (SE).

For TM, sections of a parallel program are annotated to be executed atomically and in isolation. Data that is speculatively written is visible only to the thread that has written it. The L2 cache tracks Read-after-Write (RAW), Write-after-Write (WAW), and Write-after-Read (WAR) conflicts. The program can configure the L2 cache to react with an invalidation or with a notification to software, or both. In the second case, software can decide which thread to invalidate to resolve the conflict. In the common case, the invalidated thread will retry to execute the section.

For SE, a sequential program is partitioned into tasks, which execute speculatively in parallel. For SE, there is a definite order of precedence, namely the expected sequential execution behavior, which hardware uses to resolve access conflicts. Data written by threads earlier in sequential semantics (older) is forwarded to threads later in sequential semantics (younger). Writes of older threads to locations already read by younger threads (WAR conflict) are signaled to the runtime system. The partitioning into speculative tasks can be controlled by `#pragma` statements similar to those of OpenMP. For example, the following code section causes the loop's iterations to execute speculatively in parallel:

```
#pragma speculative for
for (i=0; i<SIZE; i++) {...}
```

The L2 cache controls memory speculation. Speculative versions are placed in separate ways of the directory set that stores the nonspeculative version. Additional tags in the directory distinguish the versions. The L2 cache can store up to 30 Mbytes of speculative state. To avoid lengthy directory traversals that alter the tags on commit and invalidation, versions are identified by dynamically allocated short speculation IDs instead of thread IDs. The system provides more speculation IDs than there are hardware threads. Commit and invalidate operations are mere state transitions of the speculation ID—consequently, they're very fast. A background scrub executes the altering of the directory tags on commit and invalidate while the hardware thread allocates simply another ID and proceeds with executing the next speculative section in parallel.

There are 128 speculation IDs. Programs can divide the set of IDs into 1, 2, 4, 8, or 16 domains. Each domain can operate either in TM mode or SE mode, allowing different hardware threads to use all modes concurrently. Each SE domain maintains its order locally for its IDs, letting ID allocations proceed independently per domain.

L2 atomic operations. The BQC chip L2 cache logic also provides an extensive set of atomic operations, intended to accelerate software operations, such as locking and barriers, and to enable efficient work-queue management with multiple producers and consumers.

The atomic operations are 8-byte load or store operations that can modify the value at the given memory address. Each L2 slice can process an atomic operation every four processor clock cycles. This high throughput enables some simple synchronization constructs and concurrent data structures that scale well to the many BG/Q threads. As an example, to perform a ticket lock wherein each thread contains a unique number, 64 threads can use the L2 atomic `Load-Increment` operation to obtain their unique value in 78 (initial latency) + 64 (threads) × 4 (cycles) = 334 cycles in an ideal case. An equivalent atomic increment implemented using general-purpose load-linked/store-conditional (LL/SC; lwarx/stwcx) instructions requires approximately 82 cycles for each core-to-L2-to-core round trip. The same round-trip latency holds for systems that use compare-and-swap (CAS) primitives. Thus, 64 threads using LL/SC or CAS would require approximately 2 × 64 × 82 = 10,496 cycles to perform the equivalent operation.

The L2 cache supports atomic operations to any 8-byte address, while still supporting

the usual load, store, and coherence operations to that memory location. An atomic operation is distinguished from a regular load or store by software setting a high-order address bit. Three other bits distinguish up to eight different atomic operations. Software maps the atomic addresses as uncached. For the processor hardware, an atomic operation is a normal load or store. Similarly, the network message unit can remote-put or remote-get an atomic operation.

The L2 atomic load operations follow. `LoadClear` returns the current memory value and then stores 0 there. `LoadIncrement` returns the current value and then increments the memory value; `LoadDecrement` is similar. `Load` returns the current memory value. `LoadIncrementBounded` assumes an 8-byte counter in the address and an 8-byte boundary in the subsequent address. If the counter and boundary values differ, `LoadIncrementBounded` behaves like `LoadIncrement`. Otherwise, 0x 8000 0000 0000 0000 is returned and indicates to software that the boundary has been reached. `LoadDecrementBounded` is similar, with the boundary located in the previous address.

The latter two operations support some high-throughput lock-free concurrent data structures. For example, for an array used as a circular buffer for a queue, `LoadIncrementBounded` prevents the consumer pointer from passing the producer pointer. In another example, for a concurrent stack, `LoadDecrementBounded` prevents the stack pointer from passing through the bottom.

The L2 atomic store operations follow. `StoreAdd` adds the given and memory values. `StoreAddCoherenceOnZero` minimizes coherence traffic, while software waits for the value 0 to be reached. `StoreTwin` stores the given value to the address and the subsequent address, if their values were previously equal. For example, this can move the top and bottom pointers of a double-ended queue (deque) into the middle, when the deque is empty. `Store` simply stores the given value. `StoreOr` does a logical-OR of the given and memory values; `StoreXor` is similar. `StoreMaxUnsigned` and `StoreMaxSigned` store the maximum of the given and memory values for unsigned integers and floating-point numbers, respectively.

L2 replacement policy. Similar to the last-level cache in previous Blue Gene architecture generations, the BG/Q L2 cache provides prefetch capabilities (from the DDR3 memory controllers) driven by hint bits generated by the L1 prefetch units. The L2 cache uses a true LRU replacement policy with a configurable LRU stack insertion point to reduce cache pollution. For example, a prefetch from main memory into the L2 cache could be inserted at position 14 of the 16-entry LRU stack, making it a probable candidate for eviction if not accessed soon. Once accessed by a real demand fetch, it could be promoted to position 8, causing unused prefetches to be evicted before it would become a victim. When accessed for a second time, it could be promoted to position 1, now least prone to eviction, as such a policy would assume likely further reuse.

DDR3 memory controllers

L2 cache misses are serviced by two memory controllers (MC0 and MC1 in Figure 1). Each controller drives a 16 + 2-byte wide DDR-3 channel at 1.333 Gbits/s per pin. It uses an 8-byte error-correcting code (ECC) for a 64-byte data block (that is, ECC is calculated across four consecutive beats), with double symbol error correct and triple detect, where a symbol is 2 bits wide × 4 beats. The ECC supports retry and partial or full chip-kill for 8 bit-wide DDR3 memory modules. The ECC protects data in the face of a full DRAM device fail, synchronous with a second bit fail.

The peak DDR bandwidth is 42.67 Gbytes/s, excluding ECC.

The memory controllers support multiple DDR3 density/rank/speed configurations. Blue Gene/Q will initially be configured with 16 Gbytes of DDR3 per BQC chip. The 1.35V DDR3 memory chips are directly soldered onto the same compute card as the BQC chip for high reliability. Because of the resulting short interconnects, we can leave the compute card's data nets unterminated, which helps with power efficiency.

Network and messaging unit

The BG/Q networking papers give a detailed description of the network and messaging unit,[5,6] so we only briefly describe them here.

The BG/Q network consists of a set of compute nodes (BQC chip + memory) arranged in a 5D torus configuration. On compute nodes, 10 chip-to-chip communication links are used to build the 5D torus. A subset of the compute nodes, called *bridge nodes*, has an eleventh communication link active to connect with an I/O node. We build the compute node and the I/O node from the same BQC chip, with different configuration settings and packaging.

Each of the BQC chip's 11 communication links is implemented using high-speed serial I/Os, which drive and receive data at (effectively) 4 Gbits/s per differential wire pair. A set of 4 + 4 (send + receive) pairs comprises a chip-to-chip link. Thus, each of the 11 chip-to-chip links can transmit raw data at 2 Gbytes/s and simultaneously receive data at 2 Gbytes/s, for a peak total bandwidth of 44 Gbytes/s.

The logic for this BG/Q interconnection network is integrated onto the BQC chip. The network logic supports point-to-point, collective, and barrier messages between nodes over a single physical network.

The message unit provides a high-throughput, low-latency interface between the network routing logic and the memory subsystem, with enough bandwidth to keep all the links busy. The message unit is connected to the crossbar switch via a single slave port and three master ports. Through this memory master capability, the message unit provides remote DMA facilities, supporting direct puts, remote gets, and memory first-in, first-out (FIFO) messages. The remote DMA extends to the atomic operations described above, such as `LoadIncrement`.

In addition, the on-chip networking logic provides hardware-assist facilities for broadcast and reduction operations, such as integrated floating-point, fixed-point, and bitwise arithmetic.

These hardware assists will handle most aspects of messaging autonomously, with minimal disturbance of the PUs.

PCI Express device

On BQC chips configured as I/O nodes, two of the 11 chip communication ports are repurposed and combined to form a PCI Express Generation 2 x8 (4 Gbytes/s) interface, supporting communication cards to the outside world. Two other 2-Gbyte/s links connect back to two compute bridge nodes, thus balancing the 4-Gbytes/s PCI Express port. On chip, the PCI Express interface logic is connected to the memory subsystem via a single crossbar switch master port.

Other slave devices

A set of low-bandwidth on-chip devices shares a crossbar slave port via an arbiter called the DevBus interface. Such devices include the boot eDRAM, which is a local static 256 Kbyte memory that can be preloaded with boot code via a serial IEEE 1149.1 (JTAG) port. During functional operation, the JTAG port serves as a side-band communication channel with the external control system, for node monitoring and control.

The DevBus also connects to the Universal Performance Counter unit, the heart of a distributed performance monitoring system that can simultaneously monitor 512 events, each with a 64-bit wide counter. Software can configure these counters to monitor and count events or to capture a 1,024-cycle-long sequence of events.

Data integrity

Because data integrity is crucial when scaling out to tens of thousands of chips and millions of threads, we have protected all on-chip data paths by a single error correct, double error detect (SEC-DED) ECC. ECC also protects most memory arrays. We implemented the register files in the processor and QPU with parity and redundancy, allowing restoration of corrupted entries from redundant copies. The L1 caches are parity protected and are automatically invalidated and reloaded from L2 on parity error. Configuration-carrying registers and state machines are typically implemented with soft error resistant latches (stacked or DICE [Dual Interlocked Storage Cell] latches) and are parity protected to reduce the probability of a silent error.

Software and performance

In BG/Q, we will maintain the BG/L[1] and BG/P[2] system software philosophy of a simple streamlined stack designed for scalability. The BG/Q system is optimized for applications that are based on standard programming models such as MPI and OpenMP. Beyond these standard programming models, the BG/Q system also lets users explore new ways to utilize the multicore, multithreaded BG/Q compute node.

The BG/Q I/O nodes will run Linux. On the BG/Q compute nodes, however, the compute node kernel (CNK) will continue to be the operating system kernel.[7] CNK offloads file I/O and maps the communication hardware directly in user space for efficient messaging. CNK, the IBM XL compiler, and associated runtime support have all been enhanced and tuned to fully exploit the advanced hardware features provided on the BQC chip.

As of November 2011, a 4,096-node BG/Q installation had achieved a performance of 677.10 Tflops on Linpack (http://www.top500.org). This system also achieved the top rating on the Graph 500 (http://www.graph500.org) at 254 Gteps (giga traversed edges per second). Finally, BG/Q systems achieved top ratings on the Green 500 list of the most energy-efficient supercomputers at 2.0 Gflops/W (http://www.green500.org).

Future work will present a more in-depth analysis of the machine, its software, and its performance. For more information on the Blue Gene team, see the ''Blue Gene Project Members'' sidebar. MICRO

Acknowledgments

The BG/Q project has been supported and partially funded by the Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the US Department of Energy, under Lawrence Livermore National Laboratory subcontract number B554331.

References

1. A. Gara et al., ''Overview of the Blue Gene/L System Architecture,'' *IBM J. Research and Development,* vol. 49, nos. 2/3, 2005, pp. 195-212.
2. The Blue Gene/P Team, ''Overview of the IBM Blue Gene/P Project,'' *IBM J. Research and Development,* vol. 52, nos. 1/2, 2008, pp. 199-220.
3. C. Johnson et al., ''A Wire-Speed Power Processor: 2.3 GHz 45-nm SOI with 16 Cores and 64 Threads,'' *2010 IEEE ISSCC Digest,* 2010, pp. 104-105.
4. S.P. Vanderwiel and D.J. Lilja, ''Data Prefetch Mechanisms,'' *ACM Computing Surveys,* vol. 32, no. 2, 2000, pp. 174-199.
5. D. Chen et al., ''The IBM Blue Gene/Q Interconnection Network and Message Unit,'' *Proc. Int'l Conf. High-Performance Computing, Networking, Storage, and Analysis* (SC 11), IEEE CS Press, 2011, doi:10.1145/2063384.2063419.
6. D. Chen et al., ''The IBM Blue Gene/Q Interconnection Fabric,'' *IEEE Micro,* vol. 32, no. 1, 2012, pp. 32-43.
7. M. Giampapa et al., ''Experiences with a Lightweight Supercomputer Kernel: Lessons Learned From Blue Gene's CNK,'' *Proc. ACM/IEEE Int'l Conf. High Performance Computing, Networking, Storage, and Analysis* (SC 10), IEEE CS Press, 2010, doi:10.1109/SC.2010.22.

Ruud A. Haring is a research staff member and manager of Blue Gene Chip Design at the IBM T.J. Watson Research Center, where he is responsible for the synthesis, physical design, and design for test of the Blue Gene chip designs, as well as for functional diagnostics of Blue Gene cards and systems. His research interests include electronic circuit design on microprocessors and application-specific integrated circuits (ASICs). Haring has a PhD in physics from Leyden University (the Netherlands). He is a senior member of IEEE.

Martin Ohmacht is a research staff member and manager at the IBM T.J. Watson Research Center, where he has worked on memory subsystem architecture and implementation for all Blue Gene supercomputer systems. His research interests include computer architecture, memory subsystems, speculative execution, design and verification of multiprocessor systems, and compiler

Blue Gene Project Members

The Blue Gene project is a team effort, benefiting from the cooperation of many individuals at IBM Research, IBM Systems and Technology Group, and IBM Software Group. The Blue Gene Team contributors are as follows.

- *From IBM:* Mike Aho, David W. Alderman, Eric Alter, Eberhard Amann, Anatoli Andreev, Sameh Asaad, John Attinella, Vernon Austel, Marcy L. Averill, Michael J. Azevedo, Joern Babinsky, Ralph Bellofatto, James R. Bentlage, Jeremy Berg, Darcy Berger, Randy Bickford, SuEllen Birkholz, Michael Blocksome, Matthias A. Blumrich, Lynn Boger, Alan Boulter, Thomas C. Brennan, Jeremy J. Brewer, Bernard Brezzo, Arthur A. Bright, Jose Brunheroto, Jay S. Bryant, Nathan C. Buck, Tom Budnik, Daniel Buerkle, Raymond J. Bulaga, Mark Campana, Kenneth M. Caron, Bob Cernohous, Douglas Chartrand, Jeffery D. Chauvin, Dong Chen, Wang Chen, Chen-Yong Cher, George L.T. Chiu, I-Hsin Chung, Paul K. Coffman, Miguel Comparan, Paul W. Coteus, Ron Daede, Bruce D'Amora, Kris Davis, Michael Deindl, Brian Deskin, Jun Doi, Marc B. Dombrowa, Roger Dong, Michael L. Eaton, Alexandre Eichenberger, Noel A. Eisley, Matthew R. Ellavsky, Kahn C. Evans, Sean T. Evans, Steve Faas, Daniel Faraj, Mitchell D. Felton, Blake G. Fitch, Ryan A. Fitch, Bruce M. Fleischer, Bill Flynn, Jeffrey Fosmo, Thomas W. Fox, John F. Fraley, Ross L. Franke, Scott Frei, Robert Germain, Philip Germann, Mark E. Giampapa, Frank P. Giordano, Mike P. Good, Thomas Gooding, Nicholas Goracke, Jason Greenwood, Michael K. Gschwind, John A. Gunnels, Shawn Hall, Michael Hamilton, Ruud A. Haring, James S. Harveland, Troy L. Haugen, Philip Heidelberger, Olaf Hendrickson, Brent Hilgart, Misky H. Hillestad, Judith W. Hjortness, Dennis Y. Huang, Todd Inglett, David J. Iverson, Randy T. Jacobson, Geert Janssen, Mark Jeanson, Mark C. Johnson, Steven P. Jones, Kirk Jordan, Jeffrey N. Judd, Kerry T. Kaliszewski, Robert Kammerer, Michael Kaufmann, Amanda R. Kaufer, Kyu-hyoun Kim, David Klepacki, Gerard V. Kopcsay, Anatoly Koyfman, Brant L. Knudson, Jon Kriegel, Sameer Kumar, Lih-Chung Kuo, Mark Kupferschmidt, David E. Lackey, Alphonso P. Lanzetta, Cory Lappi, Jay A. Lawrence, David Lawson, Tak O. Leung, Tom Liebsch, Meryl Lo, Ray Lucas, Bob Lytle, Scott H. Mack, David Malone, Amith R. Mamidala, Jim Marcella, Christopher M. Marroquin, John K. Masi, Patrick J. McCarthy, Moyra K. McManus, Mark Megerian, Douglas Miller, Sam Miller, Jaime H. Moreno, Adam Muff, Patrick Mulligan, Roy Musselman, Tom Musta, Indira Nair, Ben Nathanson, Mike Nelson, Hoang N. Nguyen, Carl Nilsen, Kinya Noguchi, Carl Obert, Kathryn O'Brien, Kevin K O'Brien, Alda S. Ohmacht, Martin Ohmacht, Bitwoded Okbay, Jim Van Oosten, Michael R. Ouellette, Bruce Owens, Mike J. Palmer, Jeff Parker, David P. Paulsen, Huyen Phan, Kerry Pfarr, Swetha Pullela, Don Reed, Michael T. Repede, Dennis Rickert, Thomas Roewer, Jeff Ruedinger, Bryan S. Rosenburg, Yogish Sabharwal, Valentina Salapura, David L. Satterfield, Jun Sawada, Vaibhav Saxena, Paul Schardt, Matthew Scheckel, Brandon Schenck, Heiko Schick, Dietmar Schmunkamp, Bob Schoen, Andrew A. Schram, Brian Schuelke, Alan D. Secor, Faith W. Sell, Woody Sellers, Robert M. Senger, James Sexton, Vinay V. Shah, Robert Shearer, Brian Smith, Karl Solie, Carlos P. Sosa, David L. Sparks, Burkhard Steinmacher-Burow, Will Stockdell, Scott Strissel, Craig Stunkel, Krishnan Sugavanam, Yutaka Sugawara, Nobu Suginaka, Takehito Sakuragi, Corey Swenson, Keith A. Tally, Yoshihisa Takatsu, Todd Takken, Andrew Tauferner, Kiswanto Thayib, John Thomas, Shurong Tian, Jose A. Tierno, Ailoan T. Tran, Matthew R. Tubbs, Priya Unnikrishnan, Jason L. VanVreede, Pascal Vezolle, Ivan Vo, Martha Voytovich, Charles Wait, Bob Walkup, Amy Wang, Alfred T. Watson, Bryan J. Weatherford, Robert W. Wisniewski, Bruce Winter, Bryon Wirtz, Kelvin Wong, Peng Wu, Ching Zhou, Jianwei Zhuang, Fred Ziegler, Matthew Ziegler, and Christian G. Zoellin.
- *From Columbia University:* Norman H. Christ.
- *From Columbia University and RIKEN BNL Research Center:* Changhoan Kim.
- *From University of Edinburgh:* Peter A. Boyle.
- *Formerly at IBM:* Jeremy Balster, James Clemens, Paul Curtis, Gabor Dozsa, Don Eisenmenger, Matthias Friitsch, Alan Gara, Russell Hoover, Fariba Kasemkhani, Stefan Koch, Brian Koehler, Matt Logelin, Curt Mathiowetz, Eric O. Mejdrich, Virginia Metayer, Timothy Moe, Mike Mundy, Eldon Nelson, Dennis Olson, Rick A. Rand, Joseph Ratterman, Daniele P. Scarpazza, Marcel Schaal, Ryan J. Schlichting, Catherine Trammell, Simeon Wahl, Tim Wensky, and Xiaotong Zhuang.

optimizations. Ohmacht has a Dr-Ing in electrical engineering from the University of Hanover, Germany. He is a member of IEEE and the ACM.

Thomas W. Fox is a senior engineer at the IBM T.J. Watson Research Center, where he has worked on digital signal processors, domain-specific graphics processors, PowerPC general-purpose processors, and the Blue Gene/Q Quad floating-point unit. His professional interests include processor architectures, floating-point arithmetic, and 3D graphics. Fox has an MEE in electrical engineering from Rensselaer Polytechnic Institute.

Michael K. Gschwind is a senior technical staff member and senior manager of system architecture in the IBM Systems and Technology Group in Poughkeepsie, where

his team defines the architecture of IBM's mainframe System z and Power systems. He previously served as Blue Gene/Q floating-point chief architect and lead at the IBM T.J. Watson Research Center. Gschwind has a PhD in computer engineering from Technische Universitaet Wien. He is an IEEE fellow, an IBM Master Inventor, and a member of the IBM Academy of Technology.

David L. Satterfield is a senior engineer at the IBM T.J. Watson Research Center, where he works on the design, verification, and bring-up of the Blue Gene/Q supercomputer. His professional interests have included the design, verification, and bring-up of the Blue Gene/P network interconnect ASIC, and the design of the microprocessors used in the XBOX 360 and of a low-power floating-point unit for IBM's embedded PowerPC processor line. Satterfield has a BS in electrical engineering from Tufts University.

Krishnan Sugavanam is a senior engineer at the IBM T.J. Watson Research Center. His research interests are in computer architecture, with emphasis on cache design. Sugavanam has an MS in computer engineering from the University of Louisiana at Lafayette and an MS in applied electronics from Anna University, Chennai, India.

Paul W. Coteus is a research staff member and the chief engineer of Blue Gene systems at the IBM T.J. Watson Research Center, where he leads the system design and packaging of these supercomputers. He manages the Systems Packaging Group, where he directs and designs advanced packaging for high-speed electronics, including I/O circuits, memory system design and standardization of high-speed DRAM, and high-performance system packaging. Coteus has a PhD in physics from Columbia University. He is a senior member of IEEE, a member of IBM's Academy of Technology, and an IBM Master Inventor.

Philip Heidelberger is a research staff member at the IBM T.J. Watson Research Center. He is a member of IBM Research's Blue Gene supercomputer hardware team, where he has been involved in network architecture and design, logic verification, hardware bring-up, network software interfaces, and efficient communications algorithms. Heidelberger has a PhD in operations research from Stanford University. He is a member of the IBM Academy of Technology and a fellow of IEEE and the ACM.

Matthias A. Blumrich is a research staff member and member of the Blue Gene hardware team at the IBM T.J. Watson Research Center, where he has worked on the architecture and design of the Blue Gene/L, Blue Gene/P, and Blue Gene/Q systems. Blumrich has a PhD in computer science from Princeton University.

Robert W. Wisniewski is the chief software architect for Blue Gene and manager of the Blue Gene and Exascale Research Software Team at the IBM T.J. Watson Research Center. His research interests include experimental scalable systems with the goal of achieving high performance by cooperation between system and application, as well as studying how to structure and design systems to perform well on parallel machines, and how to design those systems to allow user customization. Wisniewski has a PhD in computer science from the University of Rochester. He is an IBM Master Inventor and ACM Distinguished Member.

Alan Gara is an Intel fellow. He was previously the chief architect of the IBM Blue Gene systems at the IBM T.J. Watson Research Center. His research interests include the design and realization of power-efficient high-performance supercomputers. Gara has a PhD in physics from the University of Wisconsin, Madison.

George Liang-Tai Chiu is the senior manager of Advanced High Performance Systems in the Systems Department at the IBM T.J. Watson Research Center, where he is responsible for the overall hardware and software of the Blue Gene Platform. Chiu is one of the three cofounders of the Blue Gene project, and he has been in charge of the Blue Gene supercomputer designs. Chiu has a PhD in astrophysics from the University of California, Berkeley. He is a member of the International

Astronomical Union and the IBM Academy of Technology and a fellow of IEEE.

Peter A. Boyle is a reader in theoretical particle physics at the University of Edinburgh. His research interests are in quantum field theory, lattice quantum chromodynamics (QCD), and supercomputer codesign for QCD simulation. Boyle has a PhD in theoretical particle physics from the University of Edinburgh.

Norman H. Christ is the Gildor Professor of Physics at Columbia University. His research interests include particle physics, lattice gauge theory, and quantum field theory. Christ has a PhD in physics from Columbia University. He is a fellow of the American Physical Society.

Changhoan Kim is a research staff member at the IBM T.J. Watson Research Center. He designed the prefetch unit for Blue Gene/Q while at Columbia University and RIKEN BNL Research Center. His research interests include supercomputer chip design, algorithm development, application performance optimization on massively parallel supercomputers, and extending the utility of high-performance computing beyond academia. Kim has a PhD in physics from Columbia University.

Direct questions and comments about this article to Ruud A. Haring, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598; ruud@us.ibm.com.