The IBM Blue Gene/Q[1] System

IBM Blue Gene Team [10]
IBM
Somers NY, USA

***Abstract*— We describe the architecture of the IBM Blue Gene/Q system, the third generation in the IBM Blue Gene line of massively parallel supercomputers, expected to be generally available in the 2011/2012 time frame. The Blue Gene/Q architecture scales to tens of PF/s. Architectural features and a system package targeting high performance and exceptional energy efficiency in the node and across the network are discussed, with focus on key innovations addressing scaling challenges in typical HPC applications. Performance results and estimates are also presented.**

Keywords-Blue Gene, PetaFLOP, HPC

I. Introduction

The IBM Blue Gene/Q[1] system, the third generation in the Blue Gene system family, follows the principle of simplicity with key targeted innovation in areas with potentially substantial benefit. Like the previous generations Blue Gene/L [1] and Blue Gene/P [2], the Blue Gene/Q system is optimized to give exceptional price-performance and energy efficiency for existing applications based on standard programming models such as MPI, while also opening up new models enabling substantially more performance. A direction pursued in the Blue Gene/Q system is the exploration of new ways to allow users to efficiently utilize the multi core Blue Gene/Q compute node. This paper describes some of these innovations in detail. Future papers will present a more in depth analysis of this machine, its software, and its performance.

Each 1024 node Blue Gene/Q rack has a peak performance of about 200 TF/s. Broad application performance results show approximately a ten-fold improvement in single node performance. Rack to rack performance over the previous generation Blue Gene/P is expected to follow this 10 fold increase for most applications. At the system design maximum 100KW power per rack, Blue Gene/Q is expected to lead energy efficiency metrics on a broad range of applications. The Blue Gene/Q water cooled package enables its high space density and most importantly, its high energy efficiency.

Blue Gene systems continue playing an important research and product role within IBM. The application reach of Blue Gene/Q will be significantly enhanced over previous generations due to an exceptional cost-performance and energy efficiency, along with innovations targeted to bring users incrementally to high utilization of threaded systems. The Blue Gene/Q node is based on a 16-cores processor chip, with four threads per core. Hardware and software are utilized to leverage multiple hardware threads to execute a single user software thread through thread level speculation. In addition, transactional memory, memory based atomic support, list based prefetch, and other innovations provide a rich environment for users to garner high scaling efficiency and the appearance of very high single software thread performance.

[1] Blue Gene/Q is an internal name used prior to announcement of Limited Availability

II. System Overview

The Blue Gene/Q system organization is similar to Blue Gene/L and Blue Gene/P. Each node consists of an ASIC and SDRAM memory chips. Applications execute on "compute nodes" which are interconnected as a torus. Each rack has 1024 compute nodes. Additional "I/O nodes" provide file I/O and other services.

The fig below describes a 20 PF/s Blue Gene/Q system, expected to consume less than 10MW. Each BG/Q node is a card with the 16 processor core BQC ASIC and up to 16 GB of SDRAM memory. Thirty-two such cards plug into a node board. Each of the two midplanes per rack contains 16 node boards for a 5 dimensional torus of size 4 x 4 x 4 x 4 x 2. The 20 PF/s, 96 rack system is a 16x16x16x12x2 torus. The I/O nodes are in a separate 3D torus. An 11th port on a compute node connects to an I/O node.

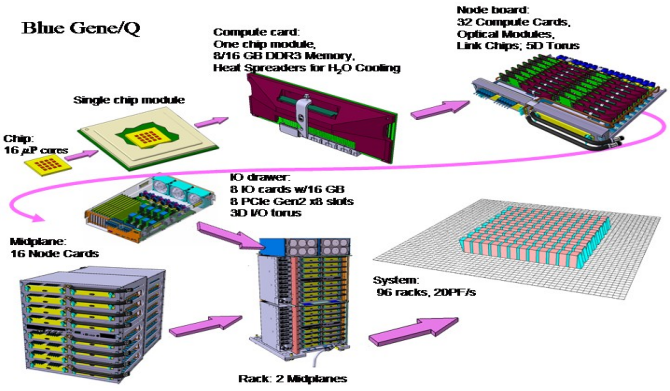


Figure 1: 96 rack, 20 PF/s Blue Gene/Q System

III. Interconnection Network and Message Unit

The BG/Q interconnection network and direct memory access engine (Message Unit, MU) are both integrated onto the BG/Q ASIC chip. Including I/O cells, the network and MU

comprise approximately 8% of the chip's area. The chip has 11 ports; each port can transmit raw data at 2 GB/s and simultaneously receive at 2 GB/s.

The BG/Q network consists of a compute node torus connected to an I/O node torus. The compute node and the I/O node are built from the same BG/Q ASIC chip, with the MU and network logic configured differently. On compute nodes, ten links are used to build a five dimensional (5D) torus for compute node to compute node communications. Some compute nodes have the 11th (I/O) link active and connected to an I/O node. On I/O nodes, two ports are used for PCIe communication to the outside world at 4GB/s. The I/O nodes are configured as a 3D torus with two 2 GB/s ports per I/O node to compute nodes to balance the 4 GB/s PCIe port.

In contrast to BG/L and BG/P, BG/Q integrates point-to-point, barrier and collective communications into a single network. BG/Q packets include a 32B header plus 0 to 512B data in multiples of 32B chunks. For long messages, the net user data rate is approximately 90% of the raw link bandwidth. Whereas BG/L and BG/P require two passes on the collective network to compute an MPI_Allreduce floating point add, BG/Q requires only a single pass. Separate up-tree and down-tree collective logic allows for arbitrary length vector all-reduces at near link bandwidth. The network can support reductions over MPI subcommunicators if it is a connected sub-rectangle of the 5D torus.

For high throughput, there are 16 network injection FIFOs and 16 network reception FIFOs: one FIFO for each of the 10 links on the 5D torus, two for intra-node local transfers, one for system I/O traffic, one for user mode high priority packets, and two for collectives (one system, one user).

A 5D torus was chosen for three primary reasons. First, from a performance perspective, a 5D torus achieves high nearest-neighbor bandwidth while increasing bisection bandwidth and reducing the maximum number of hops compared to a lower dimensional torus. For example, a 20 PF 16x16x16x12x2 BG/Q has about 46 (19) times the bisection bandwidth than a 64x48x32 BG/L (BG/P) with the same number of nodes. Compared to BG/L, 11.4x of the 46x comes from increasing the link bandwidth from 175 MB/s to 2 GB/s and 4x comes from reducing the hops of the maximum dimension. Second, the torus permits partitioning a large machine into independent sub-machines; applications running in different partitions do not affect one another at all, except possibly for file I/O. Third, from a packaging perspective, the torus permits most links, 7to be electrical rather than optical, reducing cost. The links internal to a midplane (4x4x4x4x2) are through circuit cards. The links that are on the surface of this 5-D cube connect through a 2nd ASIC, the link chip, to an optical transceiver. The link chips provide optical encode/decode, fiber sparing, and additional flexibility in creating partitions. All optical links use an error correcting IBM 8B/10B-P code [3] and have spare fibers that can be used to replace failed links without interrupting an application.

The message unit provides an interface between the network and the BG/Q memory system. It is designed to provide low latency and high throughput, enough to keep all the links busy. The MU provides similar functionality to the BG/P DMA, supporting direct puts, remote gets and memory FIFO messages. The MU maintains pointers to memory for up to 544 injection FIFOs and 272 reception FIFOs. Cores initiate messages by placing a 64B descriptor in a slot of an injection FIFO and updating that FIFO's tail pointer. A message may target one or more network injection FIFOs; each packet of the message is placed into one of the specified FIFOs. The MU packetizes messages and provides for simple address translation on the reception side. Messages can have arbitrary byte alignment and incoming packets can optionally cause interrupts. For each packet of a direct put message, the MU uses the BG/Q L2 StoreAdd functionality to atomically update that message's byte counter in memory.

Whereas the BG/P DMA has two engines, one for injecting packets and one for receiving packets, the MU has 32 such engines; 16 injection Messaging Engines (iMEs) and 16 reception Messaging Engines (rMEs). Each iME (rME) is tied to one of the 16 network injection (reception) FIFOs. The iMEs and rMEs share 3 master ports on the BG/Q memory system crossbar switch. The master port bandwidth is sufficient to support all 10 (user mode) ports on the network, when the messages fit in L2. VHDL logic simulation measurements show that up to 98% of peak performance is obtained for a full 5D nearest neighbor exchange. The MU has one slave port to the crossbar switch and thus provides memory-mapped addresses to update FIFO pointers, set up address translation, and handle certain interrupt conditions.

Both the network and MU have extensive logic and checks to separate user from system traffic, and to prevent user-space errors from interfering with system messaging. For example, the network has separate user and system virtual channels and MU FIFOs can be configured to be either user or system. All internal buffers and data paths in both the MU and the network are ECC protected, providing high resistance to soft errors.

IV. Node Overview

A. *Cores and memory nest*

Figure 2: BG/Q Processor Node Architecture

The BG/Q chip uses the 4-way multi-threaded 64 bit PowerPC architecture microprocessor core. The chip contains 17 active cores and one redundant core that can be swapped in

for any of the 17 cores in case of a hard failure. The 17th core is dedicated to system functions, thus it is not available to user programs. The core executes instructions in-order, in two pipelines at 1.6GHz. One pipeline executes all integer, control and memory access instructions, while the other pipeline controls executes all floating point arithmetic instructions through a 4-wide SIMD double precision floating point unit known as QPU, delivering up to four fused multiply-add results per processor clock.

The first level (L1) data cache is 16 KB, 8-way set associative, with 64B lines. A 32B wide load/store interface that allows loading or storing 32B per cycle. The instruction cache is 16 KB, 4-way set associative.

Each core is directly connected to a private prefetch unit (Level-1 prefetch, L1P), which accepts, decodes and dispatches all requests sent out by the core. The store interface from the core to the L1P is 32B wide, and the load interface is 16B wide, both operating at the processor frequency. The L1P implements a fully associative, 32 entry prefetch buffer. Each entry can hold 128B.

The second level (L2) cache is shared across all processor cores. To provide sufficient bandwidth, it is split into 16 slices and connected via a crossbar switch to the cores. Each slice is 16-way set associative, operates in write-back mode, and has 2 MB capacity. Physical addresses are scattered across the slices via a programmable hash function, to achieve an even slice utilization.

The L2-cache serves as point of coherence as well as control for atomic accesses using LARX/STCX instructions in the Power architecture. This cache also provides memory speculation support and atomic memory update operations, which are discussed in more detail in later sections.

Similar to the last-level cache in previous Blue Gene architecture generations, the BG/Q L2-cache provides prefetch capabilities driven by hint bits provided by the L1-prefetch unit. A true LRU replacement policy with configurable LRU stack insertion point is used 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 itself 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.

L2-cache misses are serviced by two memory controllers. Each controller is connected to two ring busses onto which four L2 slices are each chained. Each controller drives a 16+2B wide DDR-3 channel at 1.333 Gb/s. It uses an 8B ECC for a 64B data block, enabling chipkill correction capability for 8b wide DDR3 memory modules. The peak DDR bandwidth is 42.67 GB/s, excluding ECC.

The crossbar switch connecting all the cores with the L2 slices is clocked at half the processor frequency. The crossbar has a 32B wide read data path from all slave devices, e.g., L2 caches, to all master devices, e.g., L1P units. The aggregate maximum read bandwidth from all L2 slices is 409.6 GB/s. The write data path is 16B wide, delivering a peak aggregate write bandwidth of 204.8 GB/s

B. *Network and PCIe devices*

The network device as well as attached PCIe devices have memory master capabilities and are also connected directly to the crossbar switch. The network device has the equivalent of three core master ports to the switch, providing a maximum read bandwidth of 76.8 GB/s and 38.4 GB/s store bandwidth. It also provides a single slave port with the same bandwidth as an L2-cache slice. The PCIe interface controls an x8 PCIe generation 2 port and drives a single crossbar switch master port, providing 25.6 GB/s read and 1yes2.8 GB/s write bandwidth.

C. *Other slave devices, data protection*

All other devices share a single slave port via an arbiter called DEVBUS interface. Such devices include the boot eDRAM, which is a local static 256 KB memory that can be preloaded with boot code via a serial JTAG port. The JTAG port also serves during functional operation as side-band communication channel for node monitoring and control.

The DEVBUS connects also to the Universal Performance Counter unit, the heart of a distributed performance monitoring system that is capable of simultaneously monitoring 512 events, each with a 64b wide counter. These counters can be configured to monitor and count events, or to capture a 1024 cycle long sequence of events.

As data integrity is crucial when scaling out to millions of threads, all data paths and request attribute busses traversing the memory nest from the core to the main memory controllers have been protected by a SECDED ECC. All memory arrays, including register files in the processor core and QPU, are also protected by ECC. The L1 caches are parity protected and are automatically invalidated and reloaded from L2 on parity error. Configuration carrying registers and state machines are also parity protected to reduce the probability of a silent error.

V. Innovations to help single thread performance

Hardware capabilities to accelerate sequential code as well as thread interactions are a crucial part of the BG/Q chip architecture, given the large number of concurrent threads and the target of high aggregate performance. We describe the salient innovations in this area.

A. *Memory Speculation*

BG/Q implements hardware support for memory speculation. It is controlled by the shared multi-versioning L2-cache. During memory speculation, the L2 cache tracks state changes caused by the speculative thread, 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 code sections to be executed without reasoning in advance about the correctness of concurrent execution. The L2 cache also tracks access overlaps (conflicts) between threads, allowing a speculation runtime system to reason about correctness of concurrent execution. This hardware support enables Transactional Memory (TM), Speculative Execution (SE), and a mode that allows recovery via in-memory core snapshots and speculative version invalidation (Rollback).

For TM, sections of a parallel program are annotated to be executed atomically and in isolation. Data 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.

For SE, a sequential program is partitioned into tasks, and the tasks are executed speculatively in parallel. The L2 cache provides ordering of the speculative threads to model sequential execution behavior. Data written by threads earlier in sequential semantics (older) is forwarded to threads later in sequential semantics (younger), and 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 iterations of the loop to be executed speculatively in parallel:

#pragma speculative for

for (i=0; i<SIZE; i++) { … }

For Rollback, execution of an application is partitioned into speculative sections or generations by timer interrupts. During execution of the timer interrupt handler, the state of the core is stored in main memory. A new speculative section is started before returning from the handler, and the subsequent memory updates by the application code are kept separate from the main memory state. If a soft error in a core is detected that can not be corrected by other redundancy mechanisms, the operating system can revert all memory updates of the last section, restore the core state from memory, and resume execution at the point of the last core snapshot.

Rollback behaves different from SE and TLS on eviction. If speculative data needs to be evicted from the L2 cache, the speculation is not rendered invalid but the state is committed, thus rollback is impossible for the remainder of the current section.

Memory speculation is controlled by the L2-cache. Speculative versions are stored in separate ways of the directory set that stores the non-speculative version. The versions are distinguished by additional tags in the directory. The L2-cache can store up to 30MB 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 (Information Descriptors) instead of thread IDs. The system provides more IDs than there are hardware threads. Commit and Invalidate operations are mere state transitions of the speculation ID, consequently they are very fast. The altering of the directory tags on Commit and Invalidate is executed by a background scrub while the hardware thread allocates simply another ID and proceeds executing the next speculative section in parallel.

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

Conflict detection operates at a granularity down to 8B. To represent this information with a reasonable directory size, the storage area for coherence associated with non-speculative data is reused to serve as per-8B-dirty-bits for speculative versions, guaranteeing the speculative writer recording to always maintain 8B resolution. The speculative reader recording uses a dynamic encoding that balances the number of speculative reader IDs with the access footprint resolution. If a line has only been read by a single speculative ID, the reader recording provides 8B resolution; in contrast, if the encoding needs to represent three arbitrary IDs, the granularity is increased to 64B.

B. L2 Atomic Operations

The L2 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 4 processor clocks (PClks). This high throughput enables some simple synchronization constructs and concurrent data structures which scale well to the many BG/Q threads. For example, 64 threads can use L2 LoadIncrement to indicate arrival at a barrier in 64*4=256 cycles in an ideal case. An equivalent atomic increment implemented using general-purpose load-linked/store-conditional (LL/SC) requires approximately 50 cycles for a core-L2-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 64*50 or 3200 cycles to indicate arrival at a barrier, for this simple algorithm.

The L2 cache supports atomic operations to any 8B 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 eight different atomic operations. Software typically maps the atomic addresses as uncached. For the processor hardware, an atomic operation is a normal load or store. Similarly, the network and message unit can remote-put or remote-get an 8B 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, 0x8000 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 queue, LoadIncremetBounded 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, when 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 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.

C. L1-Prefetch Architecture

A level-1 prefetch unit (L1p) accompanies each processor core. Two types of automatic data prefetching attempt to reduce the latency of accesses to the DDR memory and the L2 cache. Data is prefetched in 128 byte lines (twice the L1 cache line length) and stored in a buffer capable of holding thirty-two 128-byte 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 is designed to anticipate a sequence of loads made from an arbitrary series of addresses, when that pattern of load addresses is accessed more than once, a behavior common to many HPC codes. Stream prefetching operates automatically without intervention from the application code once a few control parameters have been set. In contrast, pattern prefetching requires the application program to identify the beginning and end of the repetitive pattern, allowing the pattern prefetching hardware to first record the pattern of L1 misses and then, on subsequent executions of this same code sequence, to anticipate those L1 misses by prefetching according to that recorded pattern.

A single unit manages the stream prefetching, making no distinction between the four hardware threads in each BG/Q core. Pattern prefetching is performed by four separate units, each associated with a specific processor hardware thread. Preset control bits determine when stream prefetching, pattern prefetching or both is active.

1) Stream Prefetching

For L1 misses, the two previous BlueGene machines incorporate hardware which automatically detected and prefetched 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 length two and can maintain 7 active streams within the 15, 128 byte lines that can be stored in the prefetch buffer. 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 the prefetch buffer storage in fixed depth prefetch schemes. To minimize prefetch storage requirements, we have designed a powerful adaptive stream prefetching engine. This assigns a variable depth to each established stream, up to a maximum of 16 streams. The largest supported depth is 8. Each stream begins with a default, preloaded depth that is typically small. When a stream is first established, data is prefetched up to this initial depth. Subsequent prefetches, again limited by this allowed depth, 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 the depth of that stream. That depth is stolen from that of the least recently hit stream, keeping the total depth for all active streams at or below the 32 line capacity of the prefetch buffer. By performing this demand-driven adaptation buffer thrashing is reduced when many streams are active. The device will adapt to operate efficiently over a range of loads; from a single stream from one thread, to multiple streams and multiple threads. The highly adaptive system also supports a mixture of high bandwidth and low bandwidth streams by allocating depth in response to demand.

The stream prefetcher has some degree of programmability for expert programmers. Programmable options can select modes of operation in which all misses optimistically establish a stream, software cache touches immediately establish a stream, or only confirmed linear sequences establish a stream.

For memory access patterns that are known to consist of numerous short length streams of length between one and eight 128 byte lines this length can be programmed into the prefetch engine to prevent fetching of unused data.

2) Pattern Prefetching

The second prefetch scheme targets repetitive, deterministic code, such as that in iterative solvers, which accesses the same pattern of addresses 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 stream prefetching engine will capture the subsequent series of L1 miss addresses produced by the corresponding hardware thread. A user bit accompanying the load command and set by the TLB page assigned to the data identifies those miss addresses to be captured. These addresses are packed into 16-byte words and written to memory.

On later execution of this repetitive portion of code, this recorded pattern is read from memory and the list of addresses is used to prefetch the needed data. The loading of this recorded address pattern 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. This is accomplished 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 L1 misses are then matched with these recorded addresses. The data at those addresses which follow the matched address up to a preloaded depth are then prefetched. This address matching and data prefetching continues until a terminating end-of-pattern marker is reached in list of recorded addresses or the programmed maximum length of list is reached.

Further refinements increase the tolerance to variations which are expected in this multi-processor environment. Perhaps most important is the ability of the address matching algorithm 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. Similarly new L1 miss addresses which do not match those in the recorded pattern can be dropped. Further, support is provided to continue the recording of L1 miss addresses on subsequent iterations through the code. This new pattern can then replace that recorded earlier, allowing a self-healing evolution as the pattern of L1 misses changes.

Finally, added hardware control allows important software features to be implemented. When a segment of code with non-repetitive data access is encountered (*e.g.* a table look-up), the application can pause and later resume the pattern prefetching. Similarly, a function call can stop pattern 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 pattern prefetching.

D. The WakeUp Unit

Each processor core has its own external WakeUp unit, whose main purpose is increasing overall application performance by reducing the degradation arising from software blocked in a spin loop or similar blocking polling loop. Since the processor core has four threads but performs at most 1 integer instruction and 1 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 frequency of the loop is highest. Another degradation arises when many L1-cached addresses are polled and thus take L1 space from other threads. In general, such degradation arises whenever the polling thread competes for hardware resources with other threads.

Each of the four hardware threads in the processor core has a pause-enable signal driven by the WakeUp unit. The following loop using the WakeUp unit has less degradation than a polling loop implemented in software. The software writes a base and enable mask for the polled address range to an address compare (WAC) register in the WakeUp unit. An instruction is executed to pause the corresponding hardware thread. When a matching address is written by any other unit, such as the other 63 hardware threads or the message unit or PCIe, the WakeUp unit drives the thread's pause-enable signal so it resumes execution. The resumed program reads the data from the matching address range. If the desired condition has 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. An L2 spill thus can spuriously resume a thread, so software must read the address to ensure that the desired condition has been reached. The WakeUp unit also can resume a thread using message unit interrupts, core-to-core interrupts, or interrupts via the global event aggregator (GEA). For example, a GEA counter can periodically resume a thread. Each of the four hardware threads in a core 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 software-enabled subset of the 12 WACs. For example, this interrupt can be used to recognize a stack overflow.

E. Quad Floating-Point Processing Unit (QPU)

The BG/Q floating-point Quad Processing Unit (QPU) implements the QPX Architecture, a new instruction set extension developed specially for BG/Q. QPX is an architectural definition and instructions for auxiliary processing in conjunction with a Power architecture microprocessor core.

The computational model of the QPX architecture is a vector Single Instruction Multiple Data (SIMD), with four elements per vector, four execution lanes, and a register file containing 32 registers of 256 bits, wherein each register contains four 64 bits vector elements.

The QPX architecture contains a full set of arithmetic instructions, including fused multiply-add instructions, a new set of permute instructions for data manipulation across execution lanes, compare, convert, 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 lanes to operate on data from adjacent lanes. A collection of load and store instructions for complex data complements the complex arithmetic instructions.

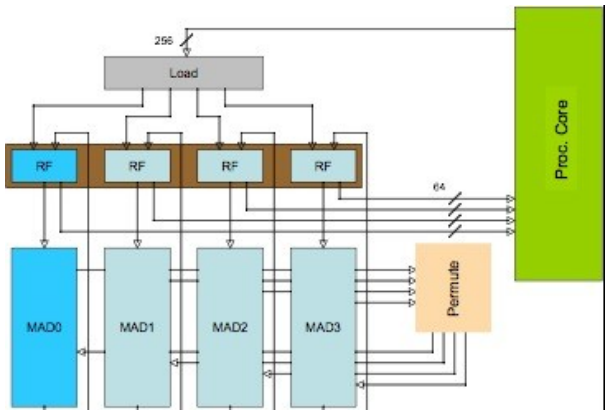
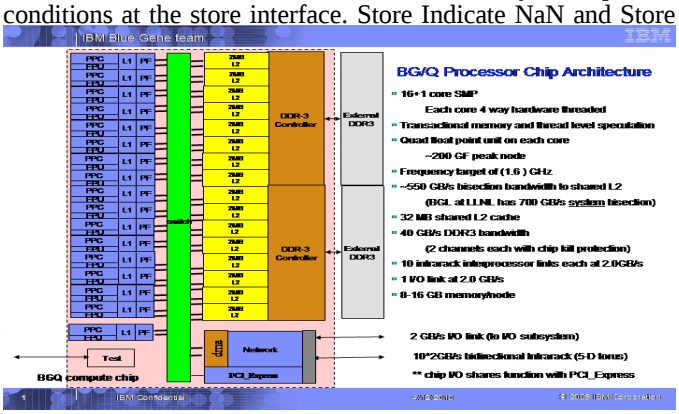


Figure 3: BG/Q Quad Float Point Processing Unit

QPX store instructions also provide a novel mechanism for the detection and indication of numerically exceptional conditions at the store interface. Store Indicate NaN and Store



arithmetic of QPX vector instructions is similar to that of Power scalar instructions, simplifying code development and SIMD vectorization by the compiler.

F. Hardware Performance Counters

The Universal Performance Counting unit (UPC) shown below collects hardware performance events from: (A) 17 processor cores, L1P and Wakeup units (B) 16 L2 units, (C) The Message, PCIe, and DEVBUS units (collectively referred to as I/O Units) and (D) the Network Unit.

Each processor core is accompanied by a local UPC unit (UPC_P). The UPC_P provides twenty-four 14-bit counters to count processor core, L1P, Wakeup event signal edges and levels, completed instructions and floating point operations. UPC_P also directs threaded performance monitor interrupt counter overflows.

Similarly, there is a UPC unit (UPC_L2) with 16 counters for each L2 slice; I/O units collectively have 43 counters.

The counters from each of the UPC units are periodically accumulated into a corresponding 64-bit counter within a central UPC module (UPC_C). The UPC_C, in coordination with the UPC_P, provides overflow detection and counter aggregation operations.

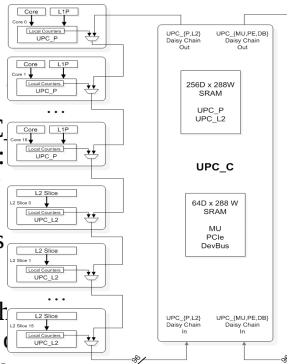


Figure unter

The UPC ree modes of collection: unters from each UPC_P 116 event signals from ich triggers and captures debug bus activity.

Outside th separate set of 66 64-bit capable of counting filterable packets and events to derive mean packet queue lengths and waiting times as well as link utilizations.

VI. System Packaging

Figure 1 shows the major packaging concepts for BG/Q. Each compute rack consists of 1 or 2 sets of 512 compute nodes. To accommodate typical machine room floor tiles, racks are 4ft wide x 4ft deep and are 84U high. Cabling and plumbing may be overhead and a raised floor is thus not required. Each node is a card with the BQC ASIC and up to 16 GB of external memory. The 1.35V SDRAM-DDR3 memory is direct soldered for high reliability and owing to the short interconnects, can be run with un-terminated data nets for excellent power efficiency. A novel marking ECC code protects data in the face of full DRAM device fail synchronous with a 2nd bit fail. 32 cards plug into a node board. Each of the up to 2 midplanes per rack contains 16 node boards for a 5 dimensional torus of size 4x4x4x4x2. The signaling rate for the 5D compute node interconnect is 4Gb/s over maximum 80cm differential traces. This torus can be extended in 4 dimensions through link chips on the node boards, which redrive the signals optically with an architecture limit of 64 to any torus dimension. The signaling rate between midplanes is 10 Gb/s, (8/10+ parity encoded), over 48 fiber multi-mode optical cables at 850nm. For example, a 20 PF/s, 96 rack system is connected as a 16x16x16x12x2 torus, with the last x2 dimension contained wholly on the node boards. Although the retry network buffers support 50m cables, the maximum torus cable length is 16m for this configuration. The link chips allow the system to be repartitioned electronically into smaller systems. To ensure high reliability, for each 4 optical fibers which carry data there are 2 fibers which are available as spares. Should a high error rate on a fiber occur, the fiber can be dynamically replaced with a spare fiber.

Each rack is powered through a set of 4 N+1 redundant bulk power supplies, which accept 400-480V 3phase AC input and deliver 48V DC output at 92.5% efficiency, and is easily air cooled. The approximately 60W ASIC maximum power ASIC and its up to 72 SDRAMs, each 2Gb, are indirect water cooled by a serpentine water pipe which contacts a heat spreader on each compute card in serial fashion. Coolant flow at 18°C is 110 l/m/rack at 30psi which results in a 12°C temperature rise. At this temperature condensation will not occur. Indeed, inlet water may be as hot at 25°C, which dramatically reduces or eliminates the need for chilled water for many data centers. The coolant is supplied to the rack through 25mm internal dimension inlet and outlet hoses with quick connects. We have developed with Proteus and integrated into our rack a novel coolant monitor which measures water temperature and flow, local humidity, and dewpoint – and which will shut down the power system in the extremely unlikely event of a leak. The node board cooling pipe, encased in an extruded Aluminum frame, also cools a 3.0KW 48V input DC-DC converter providing 6 voltages in N+2 or N+1 redundant fashion depending on delivered current, as well as 8 link chips and 16 pairs of 12 channel optical transceivers. The highly efficient DC-DC converter is built of two stages with the final stage immediately adjacent the loads for minimal distribution loss, and the initial stage is on pluggable units in 1+1 redundant fashion. The power converter can be serviced without powering off the node board.

I/O nodes, formed of identical BQC ASICs but with one torus dimension remade as a byte wide PCIe Gen2 port supporting a full sized adapter card, are packaged 8 at a time in rack-mounted, air-cooled 3U drawers. Six link chips with optical transceivers allow I/O drawers to be connected in a 3D or 4D torus, and also connect optically to compute racks. Alternatively up to 4 I/O drawers may be placed above the BG/Q compute rack, creating a compact single rack structure. A 9th link chip and its associated transceivers are available on select node boards to connect an 11th port from up to 8 compute nodes to these I/O nodes. To match I/O bandwidth on the I/O port (2 GB/s send + 2GB/s receive) to the I/O bandwidth of the PCIe adapter card associated with the I/O node (4 GB/s send + 4GB/s receive), the I/O links of 2 compute nodes are combined to feed an I/O node. Through these self contained 48V DC powered drawers, the mix of compute node to I/O node can be varied on a per rack basis from a minimum to 2 I/O nodes, to a maximum of 128 I/O nodes, with over 512 GB/s of I/O connectivity, per rack. Supported PCIe Gen2 cards are QDR_IB, 10Gb Enet, SATA, and others as requested.

Each midplane contains a service card which distributes 1Gb Ethernet from a control host to every node, and also distributes a common system clock cabled to all racks through

5m long coax cables and up to 8 layers of 1 to 10 PECL clock fanout. The system clock distributes a programmable "glitch" used to align processors and allow cycle accurate stop clock function. The service card performs atomic functions such as power throttling, environmental monitors, safety shutdown, bulk power supply control, and other control/diagnostic function under control of a central control host.

The BG/Q node board is shown below. A research prototype consisting of one midplane and two I/O drawers has been built and tested at the IBM T.J. Watson Research facility.

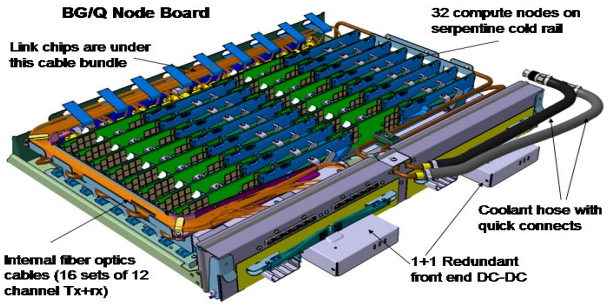


Figure 5: Blue Gene/Q Node Board

VII. Software

Previous publications have described the Blue Gene/L [1] and Blue Gene/P [2] software models. In this section, although we present the overall Blue Gene/Q software model, we focus on those areas that have seen the most significant changes relative to the earlier models. We have maintained the philosophy of a simple streamlined system software stack with a design goal of extreme scalability. This keeps the Compute Node Kernel (CNK) introduced with BG/L, which offloads file I/O and which maps the communication hardware directly in user space for efficient messaging. The increasing numbers of cores on chip and the increasing ratio of floating point to I/O operations drive the most significant changes for the BG/Q system software.

The major planned software enhancements for BG/Q include support for a variable number of software threads per hardware thread, with up to 64 processes per node, efficient support for mixed-mode and shared memory programming paradigms, support for scalable atomic operations, support for hardware transactional memory, support for thread-level speculation, the ability to run multiple jobs from a given I/O node, micro-rollback capability, further reduction of operating system jitter, and Red Hat Linux running on the I/O nodes. While the research and implementation of many of these are well along, it is possible that some will not in the end succeed and thus would not be available in the product.

We describe the process and threading limits and the challenges in maintaining the scaling philosophy while providing additional functionality. We use these tradeoffs as a motivating example of the issues faced in the design of a light weight stack targeted at extreme scalability. On Blue Gene/L and originally on Blue Gene/P we provided only one thread of computation per hardware computing entity (on BG/L and BG/P that was a core, and on BG/Q it is a hardware thread), and no more that one process per core. This was driven by two concerns. The first was a desire to implement a simple non-preemptive scheduler, and the second was a desire to ensure zero TLB misses. Studies on Linux have shown that historically it has had performance challenges with respect to high performance computing. The studies have pointed to a broad array of challenges, but noise induced by scheduling and memory management have been the primary issues. Even recent work that has modified Linux to address these does not achieve the performance reproducibility and scalability of CNK [5]. Thus, although there was a demand for more processes and more threads, we had to be careful how this was introduced. We allow more threads on a given hardware thread but the threads run to completion or until an external interrupt arrives. The user model is a cooperatively scheduled one in that the application and runtime(s) must work together to ensure that the runnable thread is the one desired to be running. A combination of utilizing the increased number of TLBs per core on BG/Q and not enforcing/providing as extensive memory protection as Linux allows up to four processes to be run per core. Tradeoffs analogous to these were applied across the system software. In the rest of this section we describe the resultant mechanisms provided.

As described in section IV.A, the BG/Q L2 cache implements a set of atomic primitives that allow common operations to be performed with a single round trip to the L2, avoiding the need to retry operations in software (which is a performance problem associated with the general purpose LL/SC or CAS atomic primitives). Operations specifically designed to support barriers and common data structures are included. They are encoded as load or store instructions to addresses derivable from the actual target storage addresses, so there is no change to the instruction set architecture and no toolchain support is needed.

In cooperation with the compiler and runtime, CNK exports an interface to the transactional memory and thread-level speculation hardware described in section IV.A. More details on CNK may be found in [6]. CNK leverages the multi-value L2 hardware used by TM and SE to provide local micro-rollback capability. We define *generations*, which are currently envisioned to be approximately 100 microseconds, which is based on the time to save the critical state and a probability analysis of a non-local event occurring. At the beginning of each generation a snapshot of the critical registers are saved, and if a soft error occurs during that generation, CNK will bring the local node back to the state it was at the beginning of the generation. While there are many caveats that will be described in future work as well as the performance tradeoffs and MTBF improvement expectations, in the ideal situation outside nodes view the node failure as a minor delay in responding rather that a failure that cascades to bringing down the whole machine.

We enhanced the BG/Q control system allowing more than a single compute block and a single job to run from a given I/O node. It is possible to run up to 64 processes per compute node. Therefore, with an I/O node controlling a block of 512 compute nodes, the ratio of I/O node to compute process could be as great as 1 to 32,768. To allow these to be subdivided and used for smaller jobs, we introduced sub-block jobs, which allows multiple jobs to run from a given I/O node within a block of compute nodes. It should be noted, however that there

is no longer complete isolation as there is between jobs in different physical partitions.

Although CNK is already optimized for low-noise and jitter, we utilize the 17th core on each compute node to offload OS and RAS processing to further minimize the noise on the compute cores. Many of these topics will be explored and analyzed in greater detail in future work.

Blue Gene/Q system's software provides a low level System's Programming Interface (SPI) and an optimized portable message layer similar to BG/P. A HWI (Hardware Interface) layer defines the structures and bits used by the hardware. The SPI has C programming inlines to the HWIs to configure for example, the Messaging Unit (MU) hardware. The kernel SPI layer allocate, initialize, configure, and free resources such as MU injection FIFOs, MU reception FIFOs, MU base address table entires and collective class routes. The implementation of these functions is kernel-dependent, but the interfaces are standard across kernels. User space SPIs build message descriptors for the message being sent, inject descriptors to initiate message send operations, receive message packets, and check for message completion. These interfaces do not have system calls. The message layer software also uses SPIs to program the MU. The message layer provides a point-to-point API and non-blocking collectives similar to the Deep Computing Messaging Framework API on BG/P [7]. High level programming paradigms such as MPI, PGAS, ARMCI and Charm++ will be built upon this portable message layer. As the BG/Q node can have up to 64 threads per process, the message layer will have several parallel communication paths to enable the threads to initiate communication concurrently. The concurrent paths are similar to the endpoints proposal presented at the MPI3 Forum [8]. The message layer will also take advantage of scalable atomic increment to design lock-less queues. For applications that are communication bound, the message layer can accelerate messaging via one or two dedicated communication threads per core of BG/Q. The wait-on-pin technique will be used to provide low overhead interrupts to sleeping communication threads.

VIII. Performance

At this early stage, we examine application performance on Blue Gene/Q by comparing to the performance on the earlier Blue Gene/L and Blue Gene/P generations. The BG/L node had two cores, a clock speed of 700 MHz and a peak performance of 5.6 GigaFlops. The BG/P node doubled the number of cores and increased clock speed to 850 MHz delivering a peak performance of 13.6 GigaFlops. The BG/Q node has 16 user cores, a clock speed of 1.6 GHz and has introduced a 4 wide floating point vector unit to deliver a peak performance of 204.8 GigaFlops. The ratio of peak performances per node between BG/P and BG/Q is 15.05.

Each BG/Q core supports execution of four independent hardware threads. On any cycle, one of the four hardware threads per core may issue a floating point operation, while another may issue a load / store operation. In contrast for BG/L and BG/P, each core supported only a single thread but that thread could simultaneously issue both a floating point operation and a load store operation each cycle. Since all codes share a mix of floating point instructions and load/store instruction, high sustained performance on BG/Q requires running at least two threads per core. Much of the application performance data to date uses all four available hardware threads on each core for application code execution.

With 16 cores and 4 hardware threads per core, the BG/Q node provides an application programming environment with up to 64 application threads. These threads allow a variety of programming modes varying from 1 MPI task with 64 threads, to 64 MPI tasks each with 1 thread. A cycle accurate simulation of the BG/Q chip has been used to estimate application performance on a single BG/Q node. The simulation is software-based or uses FPGA-based hardware. The applications used for these estimates was compiled using prototype compilers and system software which are not yet configured to take advantage of the BG/Q four way vector floating point unit, nor the special prefetch capabilities in BG/Q. The data presented Table 1 shows performance speedups per core and per node for a set of application micro-kernels provided as part of the Sequioa benchmark suite [9]. The data presented in Table 2 provides performance speedups for a selection of other applications. To generate these performance estimates, in all cases, the kernels and applications were configured to run on the BG/Q node with sufficient independent copies to use the 64 hardware threads of the BG/Q node.

Table 1: Performance estimates for a BG/Q core and node relative to observed performance of similar codes on a BG/P node for the Sequoia micro-kernels.

Case	*Perf per core Q/P*	*Perf per node Q/P*
AMGmk	2.72	10.86
IRSmk	2.58	10.30
LAMMPSmk	3.01	12.03
UMTmk (-O2)	3.03	12.11
SPHOTmk	2.55	10.21

Table 2: Performance estimates for a BG/Q core and node relative to observed performance of similar codes on a BG/P node for a selection of kernels and applications. Those applications which are marked "asm" include hand coded assembly elements to take advantage of the four way vector floating point unit available on Blue Gene/Q.

Case	*Perf per core Q/P*	*Perf per node Q/P*
DGEMM asm	3.94	15.78
Crystal div	3.45	13.80
Crystal pow	2.93	11.73
Crystal cholesky	2.87	11.49
NEK asm	3.51	14.05
MILC asm	3.81	15.23
LS3DF	3.23	12.90
NAMD	2.18	8.72
Saturne	2.96	11.83
RTM	3.30	13.20

In Tables 1 and 2, the data show at least ten fold speedups in performance from BG/P to BG/Q for nearly all applications. At this early stage, the ten fold speedup is an encouragingly

large fraction of the factor 15.05 increase in peak performance. In Table 2, for those cases where hand coded assembly language has been used to take advantage of the four way vector unit which BG/Q provides, performance speedups are fourteen fold or more. Further improvements are expected as software and compilers mature. At the time of presentation latest values will be shown.

Acknowledgment

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/Q project has been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy, under Lawrence Livermore National Laboratory subcontract no. B554331.

References

[1] A. Gara et. al, "Overview of the Blue Gene/L system architecture," IBM Jorunal of Research and Development, vol 49, no. 2/3, pp. 195 – 212, March/May 2009.

[2] The Blue Gene/P Team. "An overview of the BlueGene/P project," IBM Journal of Research and Development, vol. 52, no. 1/2, pp. 199-220, January/March 2008.

[3] A.X. Widmer, "Transmission code having local parity", IBM US Patent 5,699,062, December 1997.

[4] S. P. Vanderwiel and D. J. Lilju, "Data Prefetch Mechanisms", ACM Computing Surv. 32, No.2, 174-199 (2000).

[5] S. Alam, R. Barrett, M. Bast, M. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. Vetter, P. Worley, and W. Yu. Early evaluation of BlueGene/P. In Supercomputing, Austin Texas, November 15-21 2008.

[6] M. Giampapa, T. Gooding, T. Inglett, and R. W. Wisniewski. Experiences with a lightweight supercomputer kernel: Lessons learned from Blue Gene's CNK. In submitted to Supercomputing, New Orleans Louisiana, November 15-19 2010.

[7] S. Kumar, G. Dozsa, G. Almasi, Dong Chen, P. Heidelberger, Mark E. Giampapa, Michael Blocksome, Ahmad Faraj, Jef f Parker, Joseph Ratterman, Brian Smith and Charles Archer, The Deep Computing Messaging Framework, In Proceedings of International Conference on Supercomputing, Kos, Greece, 2008.

[8] https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/MPI3Hybrid

[9] https://asc.llnl.gov/sequoia/benchmarks/

[10] IBM Blue Gene Team spans a wide range of technical disciplines and organizations within IBM, as well as several Universities. The Blue Gene Team is comprised of:
From IBM: Alan Gara, Alda Sanomiya, Alexandre Eichenberger, Alphonso P Lanzetta, Amanda R Kaufer, Amith R Mamidala, Anatoly Koyfman, Andrew Tauferner, Arthur A Bright/Watson, Barbara McGinnis, Ben J Nathanson, Bernard Brezzo, Bill Donegan, Bill Flynn, Blake Fitch, Bob Cernohous, Bob Lembach, Bob Lytle, Bob Schoen, Bob Walkup, Brandon Schenck, Brant L Knudson, Brian Deskin, Brian Hruby, Brian Schuelke, Brian Smith, Bruce D'Amora, Bruce M Fleischer, Bruce Winter, Bryan S Rosenburg, Bryon Wirtz, Burkhard Steinmacher-Burow, Carl Nilsen, Carl Obert, Carlos P Sosa, Catherine Trammell ,Changhoan Kim, Charles J Archer, Charles Wait, Chen-Yong Cher, Ching Zhou, Christian G Zoellin, Christopher M Marroquin, Cory Lappi, Craig Stunkel, Curt Mathiowetz, Damir Jamsek, Daniel Faraj, Daniele P Scarpazza, Darcy Berger, Dave Naatz, Dave Sparks, David Alderman, David Klepacki, David L Satterfield, David Lackey, David Lawson, Dennis Olson, Dennis Rickert, Dietmar Schmunkamp, Don Eisenmenger, Don Reed, Dong Chen, Doug Dreibelbis, Douglas Miller, Eberhard Amann, Edi Shmueli, Eldon Nelson, Emanuel Gofman, Faith W Sell, Franco Motika, Frank P Giordano, Gabor Dozsa, Geert Janssen, George Chiu, Gerard V Kopcsay, Gheorghe Almasi, Giovanni Fiorenza, Glenn Leckband, Gordon W Braudaway, Guansong Zhang, Haruki Imai, I-hsin Chung, Indira Nair, Ivan Vo, Jaime H Moreno, James Clemens, James Harveland, James Sexton,Jay A Lawrence, Jay S Bryant, Jeff Burns, Jeff Parker, Jeff Ruedinger, Jeffery D Chauvin, Jeremy Balster, Jeremy Berg, Jim Bentlage, Jim Marcella, Jim Van Oosten, Joel T Ficke, Joern Babinsky, John A Gunnels, John Attinella, John Fraley, John H Magerlein, John P Orbeck, John Sheets, John Thomas, Jose A Tierno, Jose Brunheroto, Jose G Castanos, Joseph Ratterman, Judith W Hjortness, Jun Doi, Jun Sawada, Kahn C Evans, Karen Magerlein, Karl Solie, Kathryn O'Brien, Kelly C Lyndgaard, Ken Caron, Kerry Kaliszewski, Kerry Pfarr, Kevin K O'Brien, Kinya Noguchi, Kiswanto Thayib, Kris Davis, Krishnan Sugavanam, Kyu-hyoun Kim, Lynn Boger, Marc B Dombrowa, Maria Eleftheriou, Mark Campana, Mark E Giampapa, Mark Jeanson, Mark Megerian, Mark Mendell, Martin Ohmacht, Matthew R Ellavsky, Matthew Scheckel, Matthew Ziegler, Matthias Blumrich, Matthias Fritsch, Meera Rangarajan, Meryl Lo/Dallas, Michael Blocksome, Michael Deindl, Michael Gschwind, Michael Hamilton, Michael Kaufmann, Michael Malms, Michael Maurice, Michael R Ouellette, Michael T Repede, Mike Aho, Mike Mundy, Mike Nelson, Mike P Good, Mitchell D Felton, Moyra McManus, Nicholas Goracke, Nikhil Jain, Nobu Suginaka, Noel A Eisley, Pascal Vezolle, Pat McCarthy, Patrick Mulligan, Paul Allen, Paul Coteus, Paul Curtis, Peng Wu, Philip Germann, Philip Heidelberger, Priya Unnikrishnan, Ralph Bellofatto, Ramon R Colon, Randy Bickford, Randy Jacobson, Raul Silvera, Ray Lucas, Rick A Rand, Rick Behun, , Robert Germain, Robert M Senger, Robert Sharrar, Robert Wisniewski, Roch Archambault, Ross L Franke, Roy Musselman, Ruud A Haring, Ryan A Fitch, Ryan J Schlichting, Sam Ellis, Sam Miller, Sameer Kumar, Sameh Asaad, Scott Frei, Scott Strissel, Seetharami Seelam, Shawn Hall, Shawn P Fetterolf, Sheila Conway, Shurong Tian, Simeon Wahl, Stefan Koch, Steve Douskey, Steven Jones, Steven Schwartz, SuEllen Birkholz, Susan Lee, Takao Moriyama, Takehito Sakuragi, Thomas Brennan, Thomas Fox, Thomas Gooding, Thomas Roewer, Tim Wensky, Timothy Moe, Todd Inglett, Todd Takken, Tom Budnik, Tom Liebsch, Tom Musta, Troy L Haugen, Vaibhav Saxena, Valentina Salapura, Vernon Austel, Virginia Metayer, Wang Chen, Wilfried Haensch, Will Stockdell, Woody Sellers, Xiaotong Zhuang, Yogish Sabharwal, Xiaotong Zhuang and Yutaka Sugawara

From Columbia University: Norman H Christ, Robert D Mawhinney and Chulwoo Jung

From University of Edinburg: Peter A Boyle