UCRL-JRNL-224370

BlueGene/L Applications: Parallelism on a Massive Scale

B. R. de Supinski, M. Schulz, V. V. Bulatov, W. Cabot, B. Chan, A. W. Cook, E. W. Draeger, J. N. Glosli, J. A. Greenough, K. Henderson, A. Kubota, S. Louis, B. J. Miller, M. V. Patel, T. E. Spelce, F. H. Streitz, P. L. Williams, R. K. Yates, A. Yoo, G. Almasi, G. Bhanot, A. Gara, J. A. Gunnels, M. Gupta, J. Moreira, J. Sexton, B. Walkup, C. Archer, F. Gygi, T. C. Germann, K. Kadau, P. S. Lomdahl, C. Rendleman, M. L. Welcome, W. McLendon, B. Hendrickson, F. Franchetti, J. Lorenz, C. W. Uberhuber, E. Chow, U. Catalyurek

September 12, 2006

The International Journal of High Performance Computing Applications

BlueGene/L Applications: Parallelism on a Massive Scale

Bronis R. de Supinski, Martin Schulz, Vasily V. Bulatov, William Cabot, Bor Chan, Andrew W. Cook, Erik W. Draeger, James N. Glosli, Jeffrey A. Greenough, Keith Henderson, Alison Kubota, Steve Louis, Brian J. Miller, Mehul V. Patel, Thomas E. Spelce, Frederick H. Streitz, Peter L. Williams, Robert K. Yates, Andy Yoo
Lawrence Livermore National Laboratory
{*bronis, schulzm, bulatov, cabot1, chan1, awcook, draeger1, glosli1, greenough1, keith, kubota, stlouis, bjmiller, mehul, spelce1, streitz, plw, kimyates, ayoo*}*@llnl.gov*

George Almasi, Gyan Bhanot, Alan Gara, John A. Gunnels, Manish Gupta, Jose Moreira, James Sexton, Bob Walkup
IBM Thomas J. Watson Research Center
{*gheorghe, gyan, alangara, gunnels, mgupta, jmoreira, sextonjc, walkup*}*@us.ibm.com*

Charles Archer,
IBM Systems and Technology Group
archerc@us.ibm.com

Francois Gygi
University of California, Davis
fgygi@ucdavis.edu

Timothy C. Germann, Kai Kadau, Peter S. Lomdahl
Los Alamos National Laboratory
{*tcg, kkadau, pxl*}*@lanl.gov*

Charles Rendleman, Michael L. Welcome
Lawrence Berkeley National Laboratory
{*CARendleman,mlwelcome*}*@lbl.gov*

William McLendon, Bruce Hendrickson
Sandia National Laboratories
{*wcmclen, bahendr*}*@sandia.gov*

Franz Franchetti
Carnegie Mellon University
franzf@ece.cmu.edu

Jürgen Lorenz, Christoph W. Überhuber
Vienna University of Technology
juergen.lorenz@aurora.anum.tuwien.ac.at, c.ueberhuber@tuwien.ac.at

Edmond Chow
D. E. Shaw Research and Development
Edmond.Chow@deshaw.com

Ümit Çatalyürek
Ohio State Univeristy
catalyurek.1@osu.edu

.

[0]Part of this work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48 (UCRL-JRNL-224370)

Address of Contact Authors

Bronis R. de Supinski

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
PO Box 808, L-557
Livermore, CA 94551
USA

bronis@llnl.gov

Martin Schulz

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
PO Box 808, L-560
Livermore, CA 94551
USA

schulzm@llnl.gov

Abstract

BlueGene/L (BG/L), developed through a partnership between IBM and Lawrence Livermore National Laboratory (LLNL), is currently the world's largest system both in terms of scale with 131,072 processors and absolute performance with a peak rate of 367 TFlop/s. BG/L has led the Top500 list the last four times with a Linpack rate of 280.6 TFlop/s for the full machine installed at LLNL and is expected to remain the fastest computer in the next few editions.

However, the real value of a machine like BG/L derives from the scientific breakthroughs that real applications can produce by successfully using its unprecedented scale and computational power. In this paper, we describe our experiences with eight large scale applications on BG/L from several application domains, ranging from molecular dynamics to dislocation dynamics and turbulence simulations to searches in semantic graphs. We also discuss the challenges we faced when scaling these codes and present several successful optimization techniques. All applications show excellent scaling behavior, even at very large processor counts, with one code even achieving a sustained performance of more than 100 TFlop/s, clearly demonstrating the real success of the BG/L design.

Keywords

- Massively Parallel Architectures
- BlueGene/L
- Application Scalability
- Performance Study and Optimization

1 Introduction

BlueGene/L (BG/L) is the product of an innovative partnership between Lawrence Livermore National Laboratory (LLNL) and IBM. From the project's inception, the machine was designed to target not only effective application performance but also to reduce several important factors in the total cost of ownership significantly. As a result, we built a highly integrated machine with relatively low power requirements, resulting in reduced costs in terms of both floor space and electricity. Nonetheless, the machine offers outstanding performance: it has occupied the top spot on four straight Top500 lists (one quarter of the final machine the first time, half the second, and the full machine thereafter) and "is expected to remain the No. 1 Supercomputer in the world for the next few editions" (University of Mannheim, University of Tennessee, and NERSC/LBNL). More importantly, new scientific breakthroughs have resulted from the excellent performance achieved by real applications.

BG/L systems use a system-on-a-chip design and five integrated networks to provide excellent overall system balance. The tight integration and use of low power processors (each BG/L rack has 2048) not only reduce electrical requirements but still allow air cooling of the systems. Building such low power systems does involve trade-offs: each dual core node has a peak performance of only 5.6 GFlop/s and 512 MB main memory. Nonetheless, very large systems can be assembled from these modest nodes: the 64 rack machine installed at LLNL has a total peak performance of 367 TFlop/s and 32TB of main memmory.

This performance is achieved by using a number of nodes much greater than in any previously deployed system. BG/L has 131,072 processors organized in 65,536 dual core nodes and, thus, requires applications to scale beyond levels previously achieved. While the system was being built, many questioned whether performance at this scale was too challenging. As evidenced by the performance of the 2005 Gordon Bell Prize winner, ddcMD (over 100 TFlop/s sustained) (Streitz, Glosli, Patel, Chan, Yates, de Supinski, Sexton, and Gunnels 2005), as well as several other applications, we met these challenges.

This paper details the performance of several applications on BG/L. We first discuss the system hardware and software architecture. We then present details of eight applications and their performance on the LLNL system. Next, we examine key issues in achieving good single node and communication performance on BG/L systems. We conclude by reviewing what we have learned from building BG/L and running real applications on it.

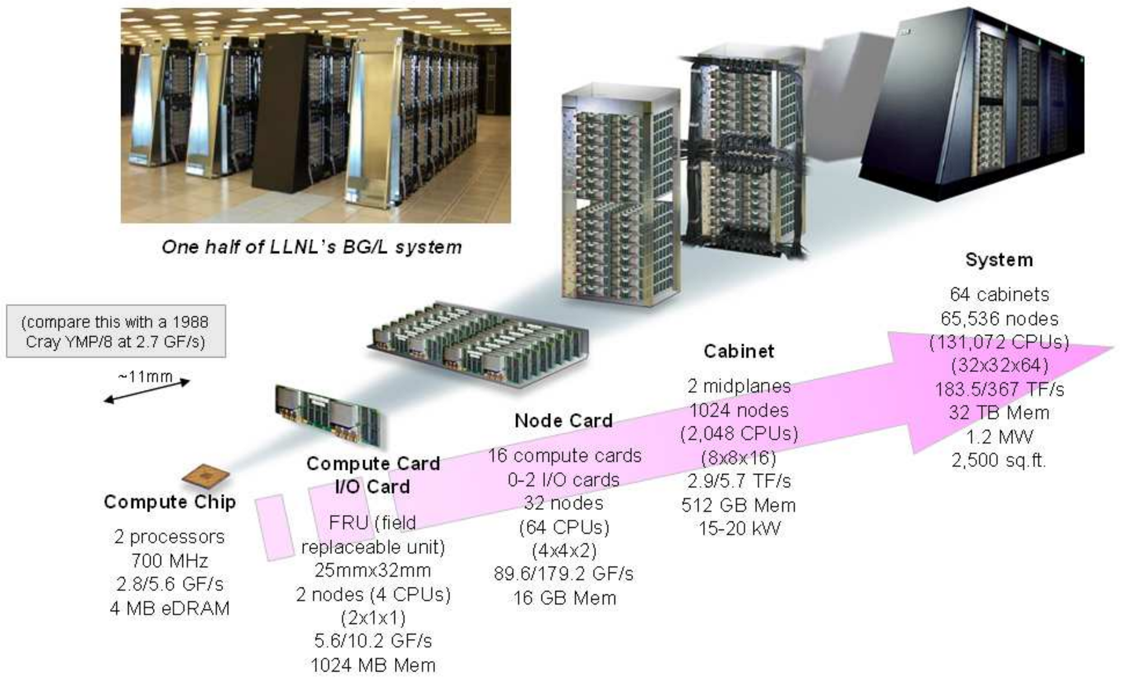


Figure 1: BG/L's scalable system architecture.

2 The Blue Gene/L System

BG/L is a tightly-integrated large-scale computing platform jointly developed by IBM and LLNL. The system installed at LLNL consists of 65,536 compute nodes, each with a dual-core (PowerPC 440) ASIC, resulting in an overall peak performance of 367 TFlop/s. It currently heads the Top500 (University of Mannheim, University of Tennessee, and NERSC/LBNL) list with a Linpack performance of 280.6 TFlop/s.

2.1 Architectural Overview

BG/L has a modular and hierarchical design, as illustrated in Figure 1. A full description of the architecture is available in an SC2002 paper (Adiga and et al. 2002). The smallest entitity is the compute node ASIC or compute chip. It includes all networking and processor functionality; in fact, a compute node uses only that ASIC and nine DRAM chips. This system-on-a-chip design results in extremely high power and space efficiency.

Two of those compute chips together with their DRAM are then placed on one compute card; 16 such cards (32 nodes or 64 processors) form a node card. 16 node cards form a midplane and two midplanes make up one cabinet with 1024 nodes or 2048 CPUs. The total size of the BG/L installation at LLNL comprises

64 cabinets totaling to 65,536 (64K) nodes or 131,072 (128K) processors. This full installation of BG/L occupies only 2,500 square feet and has a total peak power consumption of only 1.2 MW. In comparison, the previous number one machine on the Top500 list, the Earth Simulator, occupies 34,000 square feet and has a peak power consumption of 10 MW (Vetter, de Supinski, May, Kissel, and Vaidya 2005), but achieves only 12.8% of BG/L's Linpack rate (University of Mannheim, University of Tennessee, and NERSC/LBNL).

In addition to the compute nodes, BG/L also has a set of I/O nodes built from the same compute cards as the compute nodes. Each I/O node is connected to a set of compute nodes that it controls and for which it provides I/O services. The exact ratio of I/O to compute nodes can vary between installations and is 1:64 for the system at LLNL, resulting in an additional 1024 I/O nodes. Each of those nodes has a Gigabit Ethernet connection, which connects it to a large federated switch.

Using the Ethernet fabric the machine is connected to a set of front end nodes. For the installation at LLNL, 14 of such nodes are available: eleven for user access; three for system testing and maintenance; and one service node for RAS and job submission database management. The former are the only machines directly accessible by the user and are intended for all software development, compilation, and job launch, while the others are reserved for administrative purposes only. Each of these nodes is a dual 64-bit PowerPC (PPC) 970 processor *js20* blade. In addition, the Ethernet switch also connects LLNL's BG/L to a 900TB disk array as well as to other systems at the laboratory, including a 256 node Opteron cluster entirely dedicated for postprocessing and visualization.

2.2 BG/L Nodes

Each BG/L compute node has two 32-bit embedded PPC 440 processor cores. The PPC 440 processor is a low-power superscalar processor with L1 data and instruction caches (32 KB each). The BG/L nodes include hardware prefetching based on detection of sequential data access. The prefetch buffer for each processor (its L2 cache) holds 64 L1 cache lines (16 128byte L2/L3 cache lines). Each chip also has a 4 MB embedded DRAM L3 cache, and an integrated DDR memory controller. A single BG/L node has 512 MB memory. The PPC 440 design does not support hardware cache coherence at the L1 level. However, there are instructions to invalidate a cache line or flush the cache, allowing software coherence management.

BG/L uses a SIMD-like extension of the PPC floating-point unit, the double floating point unit

(DFPU) (Bachega, Chatterjee, Dockser, Gunnels, Gupta, Gustavson, Lapkowski, Liu, Mendell, Wait, and Ward 2004). To the primary FPU, the DFPU adds a secondary FPU with a twinned register file. The second FPU is not an independent unit: it is used with a comprehensive set of special parallel instructions. This instruction set includes parallel add, multiply, fused multiply-add, and additional operations to support complex arithmetic. All SIMD instructions operate on double-precision floating-point data.

Each node can be operated in two modes. In *Co-Processor Mode* one core is used for computation and the second is dedicated to communication, while in *Virtual Node Mode* both cores are used for computation and communication. The use of the two modes depends on the application's computation and communication characteristics as well as memory requirements.

2.3 BG/L Networks

The BG/L ASIC supports five different networks: torus, collective, global interrupt, Ethernet, and JTAG. The main communication network for point-to-point messages is a three-dimensional torus and connects all compute nodes. Each node has six bi-directional links for direct connection with nearest neighbors. The raw hardware bandwidth for each torus link is 2 bits/cycle (175 MB/s at 700 MHz) in each direction. The torus network provides adaptive and deterministic minimal path deadlock-free routing.

The collective network is implemented as a tree network and connects all compute nodes as well as the I/O nodes. It is used for both collective MPI operations and any communication between the compute and I/O nodes. Each type of communication (collectives vs. I/O node) is performed on a separate hardware context to avoid conflicts and potential deadlocks. The collective network implements broadcasts and reductions with a hardware latency of 1.5 microseconds for a 64K node system. Reduction operations are limited to integer operations. Floating point reductions can be supported either through a two pass operation on the exponent and mantissa or through a one pass bit shifted operation (using 1024 bits for doubles). The availablity of a single context for collectives restricts usage of the collective network to a single, fixed set of nodes within a node allocation and hence is typically used only for collective operations on all nodes within an allocation or machine partition.

The global interrupt network supports fast barriers, also with 1.5 microsecond latency over 64K nodes. Each BG/L chip also supports a serial JTAG network for booting, controlling and monitoring the system.

Finally, each node contains a 1Gbit/s Ethernet interface on the ASIC for external connectivity. However, only I/O nodes are attached to the Ethernet network, which connects the BG/L system to external file servers and host systems.

2.4 BG/L System Software

Each compute node runs a custom light weight kernel (CNK or Compute Node Kernel). This kernel does not include any support for scheduling or multi-tasking and only directly implements light-weight system calls. A larger subset of standard Unix system calls is “function shipped” to the I/O node, which executes them as a proxy for the compute node and returns the results. Since the CNK supports only one thread per core, tools as well as runtime systems cannot rely on daemons in their software architecture. While this restricts the programming and usage model of applications on BG/L, it has the benefit that applications can not be interrupted by background jobs or daemons. As a direct consequence, the machine is virtually noise free (Davis, Hoisie, Johnson, Kerbyson, Lang, Pakin, and Petrini 2004) leading to low overheads and high reproducibility of individual experiment runs.

On the I/O node, BG/L deploys a restricted operating system based on Linux, which provides limited multi-tasking and threading support. The central component running on every I/O node is the *CIOD*. This daemon manages the communication between the front end and compute nodes, executes system call requests from the compute nodes, and provides I/O access to applications. In addition, the *CIOD* allows tools to start and to control one additional I/O node daemon, which in turn can communicate with the *CIOD* and control the compute nodes using a proprietary debugging interface provided by the *CIOD*. Debuggers, such as TotalView, use this additional daemon to implement their functionality (DelSignore). We also use it to implement scalable data collection infrastructures for performance analysis based on tree overlay networks (Schulz, Ahn, Bernat, de Supinski, Ko, Lee, and Rountree 2005).

Service and front end nodes run a full Linux/PPC OS image based on SUSE Linux Enterprise Server (SLES). This choice provides the user with all standard tools for software development on the front end nodes, creating a development environment that is very similar to that of an IBM SP platform. The service node hosts a DB2 database system that is used to control jobs on BG/L and to log any system events including RAS data. This approach has proven very helpful in detecting hardware and software problems.

2.5 Writing and Running Applications on BG/L

Users only access the front end nodes directly. Application development is performed on these nodes, including cross-compilation and static linkage with any CNK libraries. The CNK does not support shared libraries or dynamic linking. The `mpicc` script encapsulates cross compilation, making it basically transparent.

The CNK provides a limited set of system calls and users must accommodate this restriction when implementing their applications. However, the set includes most commonly used OS functionality, including file I/O and client sockets. The CNK library also includes BG/L-specific functionality allowing a process to query its node's *personality*, which describes all properties specifically associated with the compute node such as its physical location within the torus network. This functionality supports optimizations such as efficiently embedding logical nearest neighbor communication.

Once the static binary is created, it can be launched using SLURM (Laboratory 2005), which accesses the service node database to reserve a partition. At this time, a job can select its run mode (e.g., Co-Processor Mode or Virtual Node Mode) as well as size constraints. The service node then boots the partition if necessary (reboots are not required to reuse an initialized partition), electrically isolates the partition (for all partition sizes larger than 512 nodes), and copies the application binary to all compute nodes. At the same time, the I/O nodes are initialized and connected to the session. Once completed, the application is started on all nodes and run isolated from the remaining jobs on the system. When a job terminates, the service node is notified and the database is updated on the service node to remove the partition reservation, thus making it available for other jobs.

3 Applications

In section we present several BG/L applications and discuss the challenges porting them to BG/L as well as their scaling properties. The application areas range from molecular dynamics to dislocation tracking and fluid simulation and also includes non-numerical problems like graph searches. This shows the breadth of application areas that benefit from BG/L's unprecedented scale and speed.

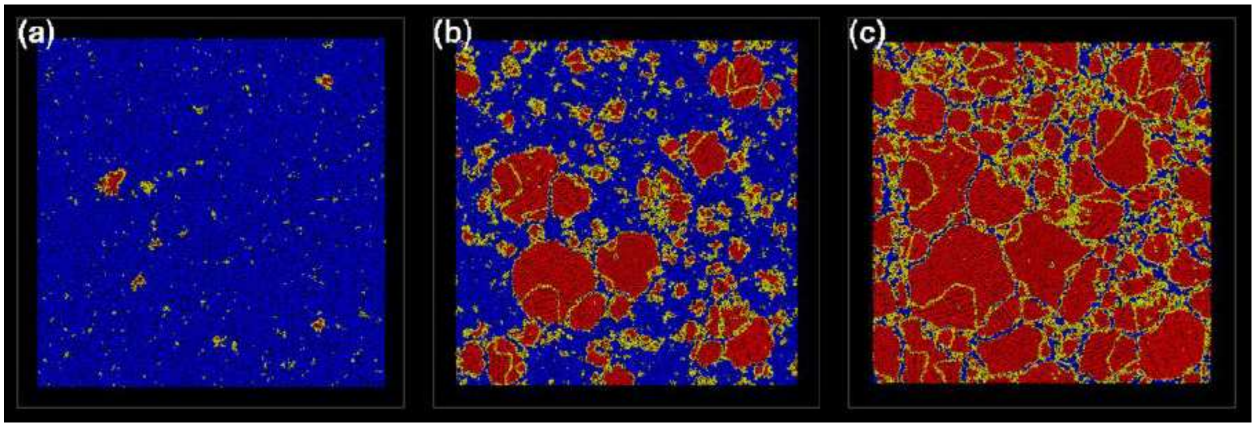


Figure 2: Cross section of the 16M atom solidification at three timesteps (a)-(c).

3.1 ddcMD

ddcMD is a scalable, general purpose code for performing classical molecular dynamics simulations. Using *ddcMD* on BG/L, we have modeled the pressure-induced solidification of molten tantalum (see Figure 2 (Streitz, Glosli, Patel, Chan, Yates, de Supinski, Sexton, and Gunnels 2005)) and investigated solidfication in quenched uranium. These simulations have substantially increased our understanding of the process of grain formations in solidification of molten metals (Streitz, Glosli, and Patel 2006).

Application Description

ddcMD achieves its accuracy by using MGPT potentials (Moriarty 1990; Moriarty 1994; Moriarty and et al. 2002; Söderlind and Moriarty 1998). Quantitative investigation of the dynamic behavior of transition metals and actinides under extreme conditions requires these semi-empirical potentials, which are based on a rigorous expansion of many body terms in the total energy. Accurate atomic scale simulation of materials behavior has previously been constrained by the maximum size that can be modeled, as un-physically small simulation cell sizes introduce artificial "size effects" into the dynamics. Scientists must either draw inferences from such small simulations, or make sufficient approximations to the underlying physics (i.e., use a "cheaper" potential) to enable larger simulations. By scaling the simulation to tens of thousands of processors, we can model dynamic behavior with results independent of system size, while using the most accurate semi-empirical potentials available, for the first time. Our code achieves nearly linear weak scaling, which is required to develop meso-scale and continuum level models of behavior.

Approach to Scaling

An innovative domain decomposition scheme is the key to the outstanding performance achieved by *ddcMD* on BG/L. Our particle-based decomposition strategy allows the processors to compute potentials for overlapping spatial regions, which is an essential property for an MD code that supports arbitrarily low numbers of atoms per processor. In addition, the domain decomposition scheme of *ddcMD* was designed to minimize redundant calculations in order to minimize time-to-solution.

Traditional decomposition algorithms determine the atoms for which a given processor computes potentials through a geometry-based (i.e., spatial) decomposition: simulated regions or *zones* are assigned to processors. Communication is determined by proximity: typically only zones that share a boundary (i.e., nearest neighbors within the simulated space) need to exchange locations of the atoms. The interaction cut-off distance supports this limited communication: provided that zones are at least as large as that distance in all three dimensions then atoms in non-neighboring zones do not interact. Although not an inherent limitation, MD codes that use a spatial decomposition frequently assume this communication pattern, which restricts the lower limit of particles per processor that they can support. Further, systems with large density inhomogeneties, such as cracks and voids, make good load balance difficult with a spatial decomposition.

The systems simulated with *ddcMD* would be especially impacted by these limits since MGPT potentials have relatively long cut-off distances. Further, simulating solidification under the regimes of interest requires fairly long time-scales. Finally, *ddcMD* is intended to run on truly massively parallel systems, such as BG/L. These factors combine to require a decomposition scheme that supports strong scaling to a point with very few atoms per processor.

While traditional algorithms apply a spatial partitioning in which one processor tracks all particles assigned to a given region of space, *ddcMD* uses a particle-based domain decomposition scheme that does not assume spatially-implicit communication partners. Instead, each processor maintains a communication list that explicitly tracks the other processors with which it must communicate. The processors on the list are exactly the ones that own atoms within the cut-off distance of an atom owned by that processor. This list allows *ddcMD* to limit communication to (almost) the minimum required during the normal simulation step. These savings come at the cost of communication required to maintain the communication lists. The *ddcMD* decomposition scheme is similar in some ways to particle decomposition schemes for particle-in-cell

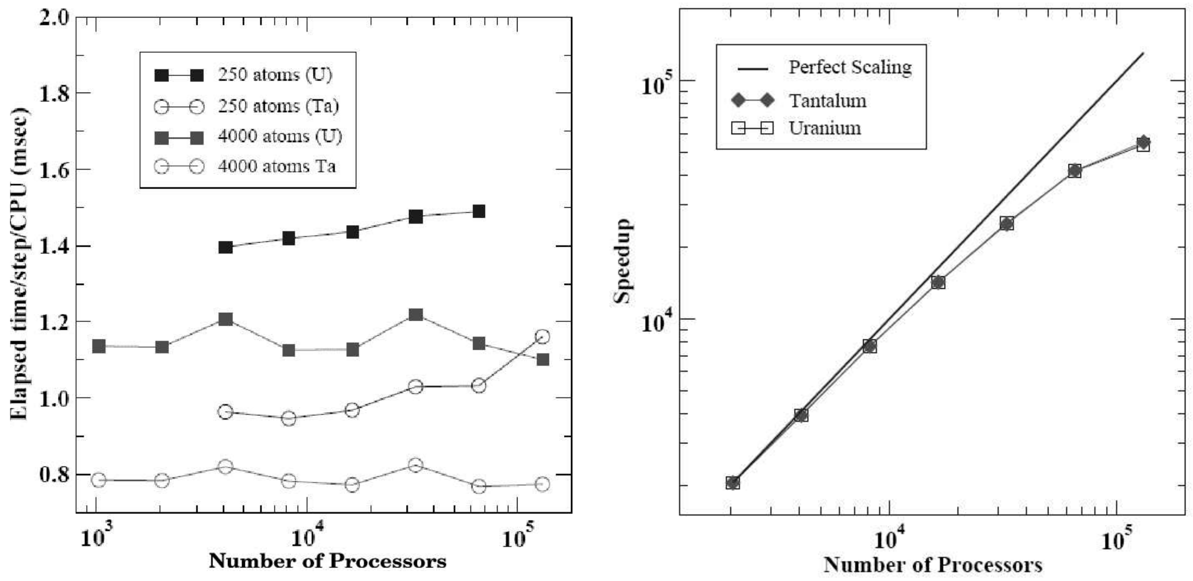


Figure 3: Weak (left) and strong (right) scaling performance of *ddcMD* on BlueGene/L.

(PIC) codes. However, PIC codes require frequent global communication while *ddcMD* relies primarily on point-to-point communication. The *ddcMD* decomposition scheme exploits the traditional MD concept of a cut off distance so that it only requires periodic global communication to ensure each communication list contains the set of processors that own domains that overlap with its own. Although not employed with the results presented here, load balancing is also simplified with this decomposition scheme (Streitz, Glosli, Patel, Chan, Yates, de Supinski, Sexton, and Gunnels 2005).

Results

Although the total TFlop/s rate varies with the number of tasks and the number of particles per task, *ddcMD* exhibits excellent weak and strong scaling behavior. Figure 3 (left side) demonstrates that time-to-solution is consistent over a variety of particles per node as the number of processors is increased. Even more impressive, we saw continuous strong scaling speedup: wall clock time for a 131,000 particle system always decreased with more processors, as shown in Figure 3 (right side). Even with only 2 particles per processor we saw speedup on a classical MD calculation—an unprecedented scaling behavior. This behavior demonstrates that the *ddcMD* decmposition scheme supports very low atom counts per processor, unlike spatial partitioning.

Table 1 shows BG/L weak scaling results for *ddcMD* on a series of short benchmark runs. The code exhibits almost linear scaling behavior across all processor counts and exceeds 107 TFlop/s on the full

Processors	No. of Atoms	Runtime [s]	TFlop/s
1,024	4,096,000	914.90	0.84
2,048	8,192,000	914.38	1.67
4,096	16,384,000	969.20	3.09
8,192	32,768,000	906.39	6.62
16,384	65,536,000	906.22	13.23
32,768	131,072,000	985.33	24.41
65,536	262,144,000	916.81	52.23
131,072	534,288,000	886.00	107.63

Table 1: Weak Scaling study for *ddcMD*

machine using Virtual Node Mode. More importantly, we achieve similar performance in full production mode that includes all I/O. For a seven hour run of the isochoric quench process in a molten uranium system, the sustained performance was over 101.7 TFlop/s on 131,072 (128K) processors/65,536 (64K) nodes (Streitz, Glosli, Patel, Chan, Yates, de Supinski, Sexton, and Gunnels 2005).

3.2 SPaSM

SPaSM (Scalable Parallel Short-range Molecular dynamics) (Bachega, Chatterjee, Dockser, Gunnels, Gupta, Gustavson, Lapkowski, Liu, Mendell, Wait, and Ward 2004; Davis, Hoisie, Johnson, Kerbyson, Lang, Pakin, and Petrini 2004; Germann, Kadau, and Lomdahl 2005) is also a classical molecular dynamics code. It was originally developed at Los Alamos National Laboratory for the Thinking Machines CM-5 in the early 1990s, but has been continuously improved and successfully ported to many large scale platforms including BG/L. *SPaSM* has been used extensively for large-scale materials science and physics simulations.

Code Structure

SPaSM uses spatial decomposition to partition space into cells with edge lengths no smaller than a parameter r_{cut}. However, rather than the standard neighbor lists used to keep track of which particles interact, *SPaSM* scans all particles in the same or immediately adjacent cells on-the-fly.

Since the CNK does not support dynamic loading, porting the current (Python-based) *SPaSM* implementation was deemed impractical. Instead, we used the pre-Python code that had been successfully ported from

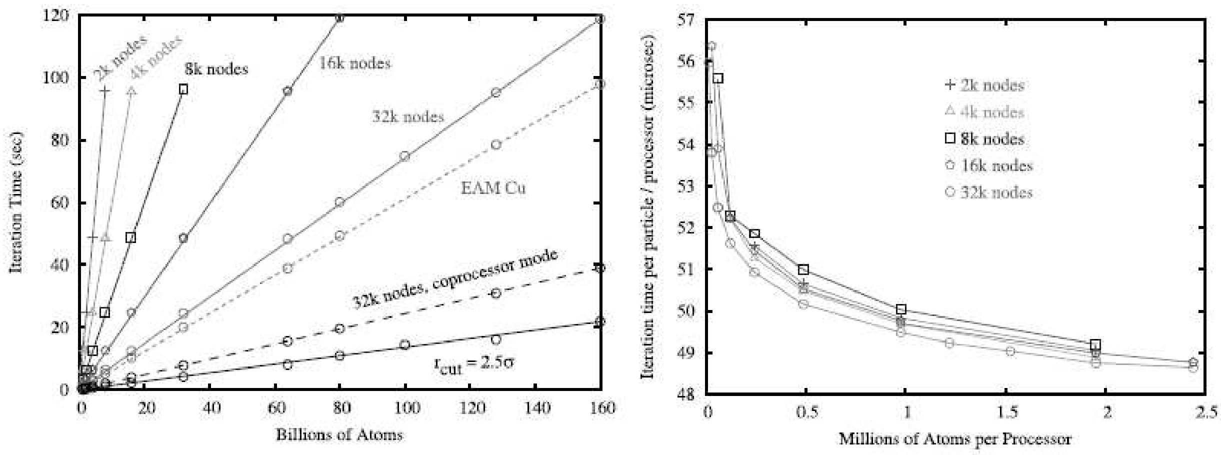


Figure 4: (Left) Average time in *SPaSM* per timestep in seconds, as a function of problem size on different BG/L partitions (2 tasks per node), for an analytic LJ potential with $r_{cut} = 5.0\sigma$. Also shown are 32k-node results for an EAM copper potential and for $r_{cut} = 2.5\sigma$, with 2 tasks per node and in coprocessor (1 task per node) mode. (Right) Scaled timings for an analytic LJ potential ($r_{cut} = 5.0\sigma$, virtual node mode).

the original CM-5 version to a variety of platforms, with either MPI, PVM, or shared memory parallelism. The critical code, the Lennard-Jones (LJ) force calculation loop, is written entirely in C and hand optimized for modern superscalar architectures by loop unrolling, macro expansion, and careful use of register variables.

Results

We carried out an extensive series of short (10 timestep) benchmark runs for timing purposes, with different numbers of particles and processors on the full system using Virtual Node Mode. The atoms are set up in a 3D fcc lattice at a temperature near the melting point, although melting of course cannot occur within such a short simulation time. Nevertheless, the timings are representative of an actual simulation at these particle densities (or, as is typically the case in many realistic simulations, the maximum local particle density at any given instant). Table 2 shows that BG/L's extremely large overall system memory size across the whole machine of 64K nodes (32TB) supports simulations of unprecedented size: 320 billion atom simulations achieve an iteration time of 26.83 seconds with $r_{cut} = 2.5\sigma$. Further, overall computational performance is also outstanding: SPaSM attains 48.1 Tflop/s with the larger $r_{cut} = 5.0\sigma$, as shown in Table 3. The difference in overall performance arises from the increased interactoin calculations required per time step with the larger r_{cut}.

Number of Atoms in Billions	Iteration Time [s]	Performance [TFlop/s]
1	0.22	11.1
2	0.31	15.8
4	0.66	15.0
8	1.35	14.6
16	1.94	20.3
32	3.84	20.1
64	7.14	21.3
128	11.86	25.1
320	26.83	27.2

Table 2: *SPaSM* Benchmarks ($r_{cut} = 2.5\sigma$) using 65,536 nodes in Virtual Node Mode (128K CPUs)

Number of Atoms in Billions	Iteration Time [s]	Performance [TFlop/s]
16	7.35	41.9
32	14.29	43.1
64	27.99	44.0
128	51.53	47.8
256	100.50	48.1

Table 3: *SPaSM* Benchmarks ($r_{cut} = 5.0\sigma$) using 65,536 nodes in Virtual Node Mode (128K CPUs)

In addition, we have run several scaling experiments with SPaSM. For each experiment we measured the average wall clock time per timestep (including force calculation, timestep integration, and particle redistribution). Figure 4 shows the times for different task numbers up to 32K nodes. Except for relatively small problem sizes (less than 100,000 atoms per processor), this resulted in 17.5–18.5 Tflop/s for the shorter-range LJ potential, and 24.5–25.5 Tflop/s for the longer-range LJ potential.

3.3 MDCASK

MDCASK simulates the motion of large collections of individual atoms using the classical laws of Newtonian mechanics and electrostatics.

Code Structure

The basic features of the code, as in any "classical" (as opposed to "quantum mechanical") molecular dynamics code, are an algorithm for the integration of the equations of motion, an inter-atomic potential, and boundary conditions and constraints. A given problem defines initial positions for the atoms (e.g., lattices for crystalline solids). *MDCASK* calculates the forces on each atom using the inter-atomic potential and atom positions, updates the velocities and then obtains new positions for the atoms based on the new velocities. Each atomic material and spatial configuration type uses an inter-atomic potential, derived from atomic theory and quantum mechanical, "ab initio" calculations. The code repeats this cycle to evolve the system over time. It can use a wide variety of potentials that allow for the simulation of metals, semiconductors, insulators, glasses and other materials.

It is the specifics of atomistic behavior that gives rise to phenomena at the meso- and macroscopic scale that are in turn responsible for the wide range of material properties important for science and industry. BG/L allow us to span the gap between the microscopic scale of individual atoms to the meso-scale, thereby providing critical validation of meso- and macroscopic models of material properties.

Results

We have conducted a weak scaling test of *MDCASK* in which a constant workload per processor (of about 250,000 atoms) was tested in powers of two from one node to 8K nodes in Co-Processor Mode. Communication in *MDCASK* is primarily broadcast and point-to-point. The runtime remains constant up to 4K nodes, as shown in Figure 5. Only at a scale of 8K performance drops off slightly.

3.4 Qbox

Qbox implements First-Principles Molecular Dynamics (FPMD), an accurate atomistic simulation approach. It is routinely applied to a variety of areas including solid-state physics, chemistry, biochemistry, and nanotechnology. FPMD combines a quantum mechanical description of electrons with a classical description of atomic nuclei. The Newton equations of motion for all nuclei are integrated in time in order to simulate dynamical properties of physical systems at finite temperature. At each discrete time step of the trajectory, the forces acting on the nuclei are derived from a calculation of the electronic properties of the system. *Qbox*

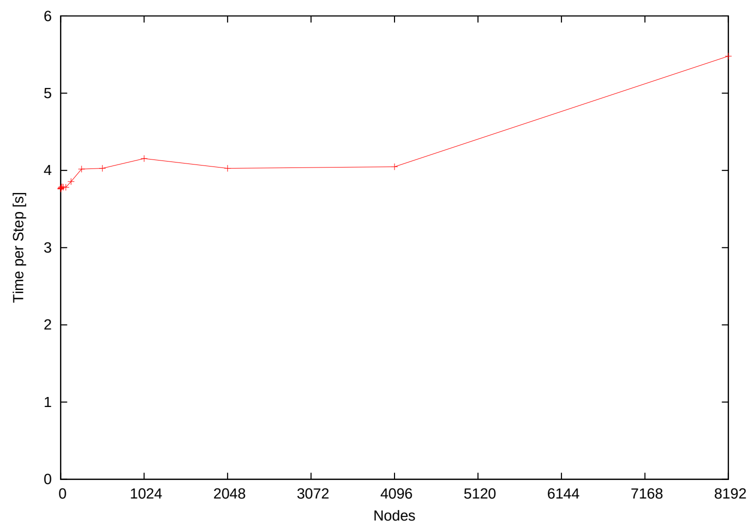


Figure 5: Weak scaling results for *MDCASK*

solves the Kohn-Sham (KS) equations (Gygi 2005) within the pseudopotential, plane wave formalism. The solution of the KS equations has been extensively discussed by various authors (Car and Parrinello 1985; Parrinello 1997).

Code Structure and Challenges

Qbox is written entirely in C++ and uses highly optimized FFTW as well as the widely known ScaLAPACK and BLACS libaries for its computation and communication (Gygi, Draeger, de Supinski, Yates, Franchetti, Kral, Lorenz, Überhuber, Gunnels, and Sexton 2005). It also relies on hand optimized DGEMM routines, which we will discuss in more detail in Section 4. The design of *Qbox* yields good load balance through an efficient data layout and a careful management of the data flow during time consuming operations.

Results

We performed simulations on partitions of increasing partition sizes up to the full 64K node BG/L system, using Co-Processor mode in all cases. The problem size was kept constant for all partition sizes. For each experiment we report the aggregate floating point rate measured by hardware performance counters. We observe superlinear scaling with 2K-8K nodes, which arises from the reduction in the amount of data per node thus leading to better use of the cache and to node mappings that provide more efficient communication

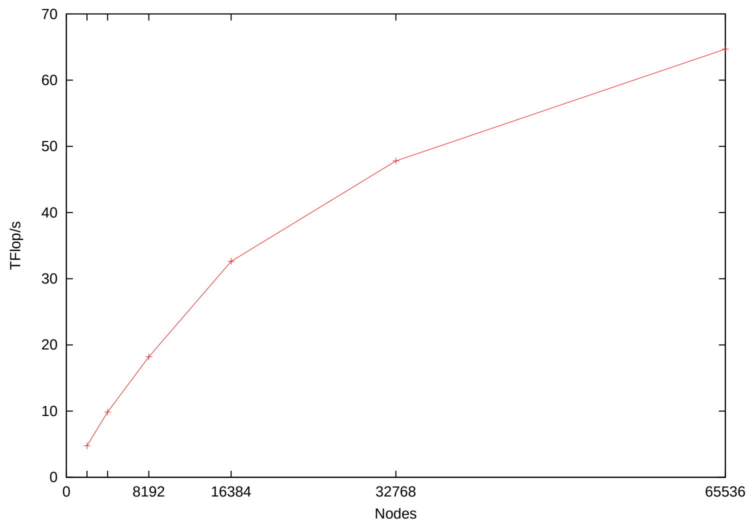


Figure 6: *Qbox* performance data for a molybdenum simulation including 1000 atoms and 12000 electrons. The peak FP rate for 64K nodes in Co-Processor mode is 64 TFlop/s.

on the 2K and 4K node partitions than on the 1K-node partition. For the largest partition, we achieve 64 TFlop/s.

3.5 ParaDiS

ParaDiS (for Parallel Dislocation Simulator) (Bulatov, Cai, Fier, Hiratani, Hommes, Pierce, Tang, Rhee, Yates, and Arsenlis 2004) directly computes the plastic strength of materials by tracking simultaneous motion of millions of dislocation lines. Simulations using *ParaDiS* provide an understanding of the fundamental nature of self-induced strengthening (or hardening) and the origin of intricate patterns that dislocations spontaneously form under mechanical straining. The code, developed at LLNL, is primarily written in C and uses MPI for interprocess communication.

Code Structure

ParaDiS relies on a line-tracking model that only considers the defects and not the rest of the material. Tracking the constantly evolving topology of the dislocation network still requires considerable effort despite the dramatically reduced degrees of freedom. *ParaDiS* uses a minimal set of (irreducible) topological operators to achieve this breakthrough in topology handling. Nonetheless, good load balance is difficult to achieve since dislocation lines tend to cluster in space and to develop highly heterogeneous distributions of degrees of freedom, which implies widely varied computation complexity across the lines. Thus, *ParaDiS* recursively partitions the problem domain and shifts the domain boundaries at regular time intervals.

Any two dislocation line segments interact with each other since dislocation interaction is long ranged. For computational efficiency, *ParaDiS* partitions all interactions into local and remote contributions, based on proximity of the interacting segments. The local interactions are computed explicitly for each local segment pair, while the effect of all remote segments in a single cell are lumped together into a super-segment contribution, using a Fast Multipole algorithm. Evaluation of forces among dislocation segments typically takes more than 80% of compute time. Optimizations for BG/L have only been through compiler options, although significant modifications were made to execute the code within BG/L's single node memory limit. The latter issue also required us to run the code in Co-Processor mode rather then Virtual Node Mode.

Results

We targeted *ParaDiS* towards a large simulation to cover the length and time scales sufficient to observe the hardening transitions that occur naturally as a result of motion and rearrangement of dislocations. A full simulation should include from 1M to 100M dislocation segments and should be traced over millions of time steps. Line dynamics capabilities available up to now at LLNL and elsewhere stop short of these target performance figures by about 2-3 orders of magnitude.

The experimental results, presented in Figure 7, show strong scaling for an identical problem running an identical number of steps at processor counts of 4K, 8K, and 16K BG/L nodes with one task per node. The runs show a speedup of 1.8x when doubling the processor count from 4K to 8K, and a speedup of 2.8x in quadrupling the processor count to 16K. However, our analysis indicates that the results were skewed by *ParaDiS*'s dynamic load-balancing. Given a specific initial problem, the load-balance of the problem decreases as the number of processors is increased. *ParaDiS* then dynamically adjusts the load balance to settle into the optimal work distribution. However, our tests show that the load-balancing mechanism requires longer to converge as the number of processors increases. Thus, we expect greater parallel efficiency for longer runs.

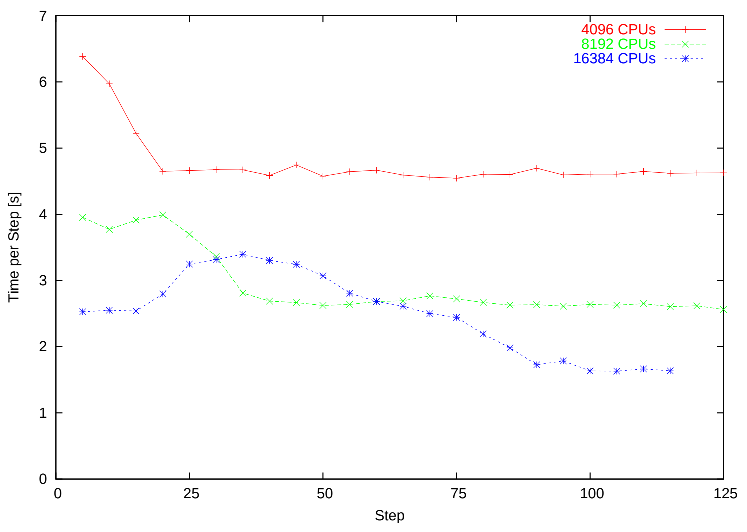


Figure 7: *ParaDiS* Time per Step with Constant Problem Size (strong scaling).

3.6 Miranda

Miranda is a high order 3D hydrodynamics code for computing fluid instabilities and turbulent mixing including the example shown in Figure 8. The application uses a massively parallel spectral/compact solver for variable-density incompressible flow, including viscosity and species diffusivity effects (Cook, Cabot, Welcome, Williams, Miller, de Supinski, and Yates 2005).

Scaling Needs and Challenges

Miranda employs FFTs and band-diagonal matrix solvers for computing spectrally-accurate derivatives, combined with high-order integration methods for time advancement; e.g., fourth-order Runge-Kutta. Fluid properties, i.e., viscosity, diffusivity and thermal conductivity, are computed from kinetic theory. The code contains solvers for both compressible and incompressible flows. It is primarily used to study Rayleigh-Taylor (R-T) and Richtmyer-Meshkov (R-M) instabilities, which occur in supernovae and inertial confinement fusion. Very large simulations, using over 1000 grid points in each direction, are needed to support the large range of length scales necessary to grow R-T and R-M instabilities to full turbulence. Smaller simulations simply cannot capture true rates of growth and mixing due to initial/boundary effects.

Due to the extensive use of FFTs and the connected all-to-all message patterns, *Miranda* is very communication sensitive. To avoid bottlenecks and achieve good load balancing, *Miranda* has been optimized for the

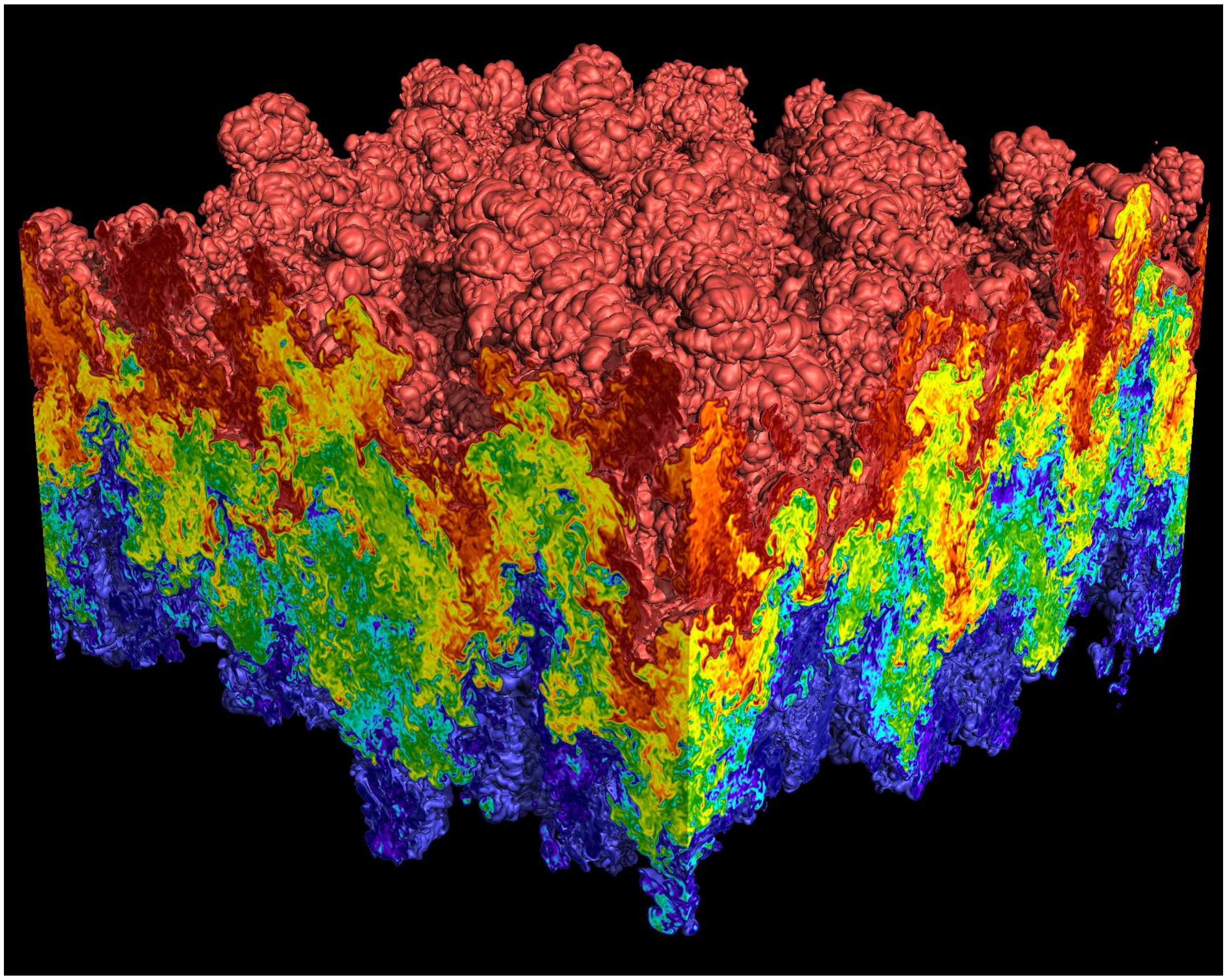


Figure 8: Turbulent state of Rayleigh-Taylor instability. Light fluid (density=1) is blue and heavy fluid (density=3) is red.

BG/L torus by distributing the data among Cartesian communicators. We then map these communicators directly onto the torus to reduce message time, primarily in *MPI_Alltoallv* on subcommunicators. All of the code operating in a master-slave CPU manner has been replaced with fully parallel routines, memory overhead has been reduced, and an FFTW library tuned for BG/L has been used.

Results

We ran *Miranda* simulating Rayleigh-Taylor instability, on up to 32,768 nodes in Co-Processor Mode and measured the overall performance (communication plus computation). Weak scaling results (fixed workload per node) are shown in on the left side of Figure 9 and strong scaling results (fixed problem size) on the right.

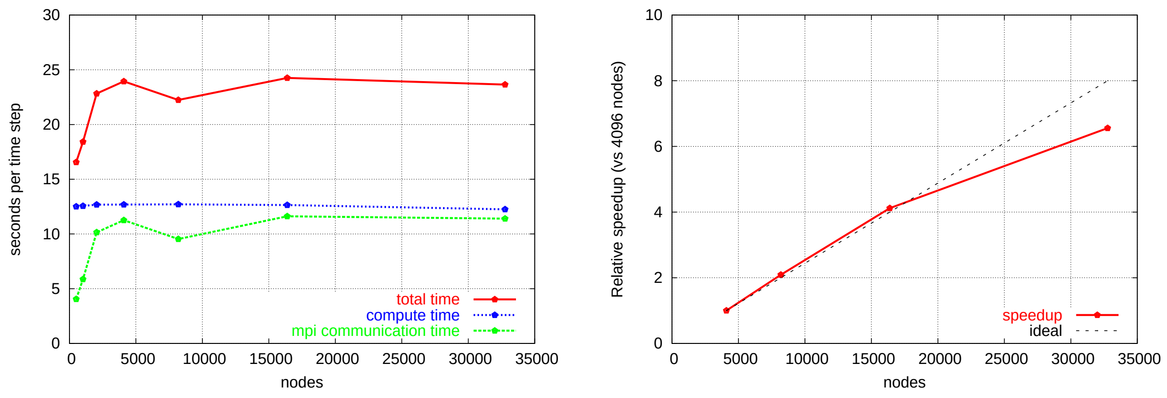


Figure 9: Weak (left) and strong (right) scaling performance of *Miranda* on BG/L.

For the weak scaling runs, each node contained a $16 \times 16 \times 2048$ point grid. For the strong scaling runs, $2048 \times 2048 \times 512$ total grid points were used. Custom torus mappings were used to improve performance on 8K and 32K nodes. The weak scaling results show that the transpose approach to computing implicit derivatives yields near perfect scaling on the BG/L architecture. Furthermore, the numbers for strong scaling show a superlinear speedup for up to 8K nodes and only a slight degradation on 16K nodes.

3.7 Raptor

Raptor, a multi-physics Adaptive Mesh Refinement (AMR) code being developed at LLNL (Greenough, de Supinski, Yates, Rendleman, Skinner, Beckner, Lijewski, Bell, and Sexton 2005), can simulate physical systems in such diverse fields as astrophysics and inertial confinement fusion. Its physics capabilities include gray diffusion radiation, electron conduction, and multifluid hydrodynamics. *Raptor* can also simulate a wide variety of materials by either using analytic or tabular equation-of-state.

Code Structure

Raptor is a hybrid C++/`Fortran` code that uses software infrastructure developed and maintained by the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory (Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory ; Rendleman, Beckner, Lijewski, Crutchfield, and Bell 2000). The base library, known as Boxlib, provides C++ classes and data containers for representing block-structured data and software for distributing and exchanging data on parallel computers using MPI. An Additional library, AmrLib, implemented in BoxLib, supports AMR methods on block-structured data. These libraries implement the main AMR operations including communicating among the grids at a particular level of refinement (intra-level communications) and communications between different levels of refinement (inter-level communications). They also control details of a calculation (e.g., number of levels of refinement, how many steps to take and parallel I/O).

Raptor implements an applications layer that contains the physics classes defined in terms of virtual functions within the AmrLib class hierarchy. Data blocks are managed in C++, in which ghost cells are filled and temporary storage is dynamically allocated, so that when the calls to physics algorithms (usually finite difference methods implemented in `Fortran`) are made, the same stencil can be used for all points and no additional special treatments are required. Thus, high-level objects encapsulate the functionality for parallelization and AMR, independent of the details of the physics algorithms. As a result, it is easy to use different physics modules as long as they implement the AmrLib interface.

Results

Figure 10 (left) shows a summary of several scans (calculations performed on a fixed size partition in Co-Processor Mode and varying the number of blocks per node) for different partition sizes. In this plot, we use the time (in seconds) to advance a grid or block one step in time as the metric for measuring performance; smaller times are better. In computing this number, we take 5 steps and then average the time. As discussed previously, the time per step at a given number of blocks on BG/L is exceptionally reproducible due to very low system noise.

Overall, *Raptor* performance on BG/L is outstanding due to the excellent balance of computation capabilities to main memory bandwidth. Performance is essentially identical across all partition sizes, with a time per step of about three seconds with one data block per node. For comparison, we show the performance of *Raptor* on the LLNL MCR system, a 1152 node Linux cluster based on 2.4 GHz Dual Intel Xeon nodes and connected with Quadric's QsNet (Elan-3) network, on the right of Figure 10. We observe several differences. First, the Linux performance shows significant variability. Second, and more importantly, the time per step on the Linux system is a comparable approximate three seconds with fewer than 500 nodes but rises steadily as the number of nodes is increased. With less than 2000 nodes, the Linux system averages about four

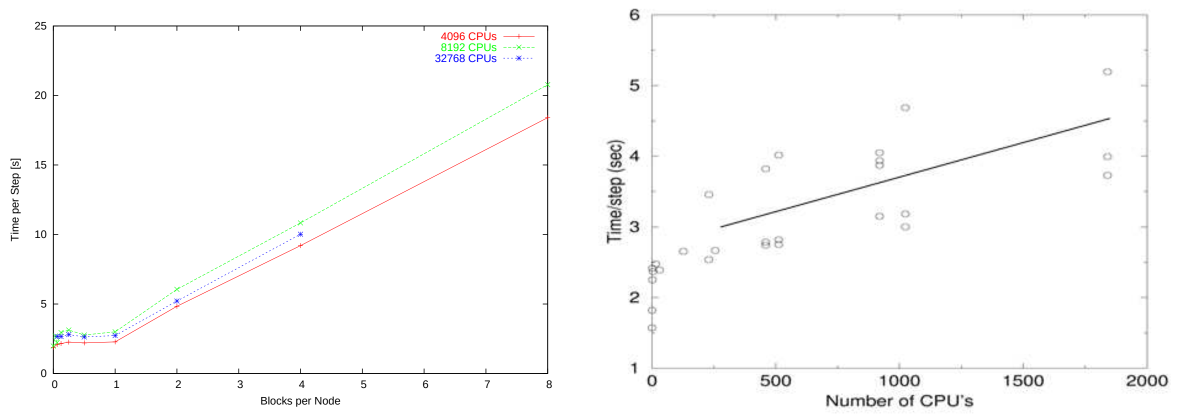


Figure 10: Left: *Raptor* runs performed with varying the number of blocks per node for different partition sizes; Right: performance on equivalent one block per node problems on LLNL MCR Linux cluster.

seconds per step despite having a clock speed more than three times that of BG/L nodes.

3.8 Breadth First Search (BFS)

Outstanding application performance on BG/L is not limited to numerical simulations. In particular, we have developed an efficient algorithm to perform parallel searches on large graphs based on Breadth-First-Search (*BFS*) (Yoo, Chow, Henderson, McLendon, Hendrickson, and Çatalyürek 2005). This operation is commonly used for analyzing semantic graphs, for example, to determine the nature of the relationship between two vertices in the graph. Such a query can be answered by finding the shortest path between those vertices using *BFS*. Further, *BFS* can be used to find a set of paths between two vertices whose lengths are in a certain range.

Scaling Challenges

Searching very large graphs with billions of vertices and edges poses challenges mainly due to the vast search space imposed by the large graphs. It is often impossible to store these large graphs in the main memory of a single computer. This makes the traditional PRAM-based parallel *BFS* algorithms (Crauser, Mehlhorn, Meyer, and Sanders 1998; Grama and Kumar 1993; Han, Pan, and Reif 1992; Klein and Subramanian 1997) unusable and calls for distributed *BFS* algorithms that move the computation to the processor that owns the data. However, it is difficult to scale even these algorithms to a machine the size of BG/L since local memory requirements and interprocessor communication increase with graph size. We used a novel *BFS* algorithm, based on sparse matrix linear solvers, to overcome these challenges (Yoo, Chow, Henderson, McLendon, Hendrickson, and Çatalyürek 2005). We also used collective operations specifically optimized for *BFS* on BG/L, as discussed in Section 4.

Results

Our experiments used synthetic random graphs that vary in the number of vertices and average degree. These random graphs do not have any structure that would lead to good partitionings. Thus, they stress the code. Our *BFS* scheme exhibits good scalability in tests with graphs of up to 3.2 billion vertices and 32 billion edges on 32,768 nodes (using Co-Processor Mode). To the best of our knowledge, this is the largest explicitly formed graph ever explored by a distributed algorithm.

Figure 11, which shows the results from weak-scaling experiments performed on BG/L. These experiments use a local problem of 100000 vertices with average degree of 10. We used balanced 2D edge partitioning that distributes the global graph to a processor space with equal row and column stripes. The code ran on up to 6400 (80×80) processors with a maximum graph size of 640 million vertices and 6.4 billion edges. To the best of our knowledge, no other distributed parallel *BFS* algorithm has ever been scaled to 6400 processors.

Figure 11 shows that execution time increases proportionally to $\log P$, where P is the total number of processors used. The increase in the execution time comes from the increased communication time, since the local problem size and, hence, computation time remains constant. This $\log P$ increase arises from limitations on collective operations over torus row and column communicators, as demonstrated through microbenchmarks we have run to test this MPI functionality.

3.9 Application Results Summary

Overall, the results for the eight applications presented here demonstrate that BG/L serves a wide array of application domains well. Clearly, BG/L with its torus point-to-point network and good overall system balance is an excellent platform for classical MD simulations, with *ddcMD* achieving over 100 TFlop/s and *SPaSM* reaching nearly 50 TFlop/s. In addition, Qbox, with strong scaling performance of over 64TFlop/s, demonstrates that application domains that require significant communication, such as those that use FFTs

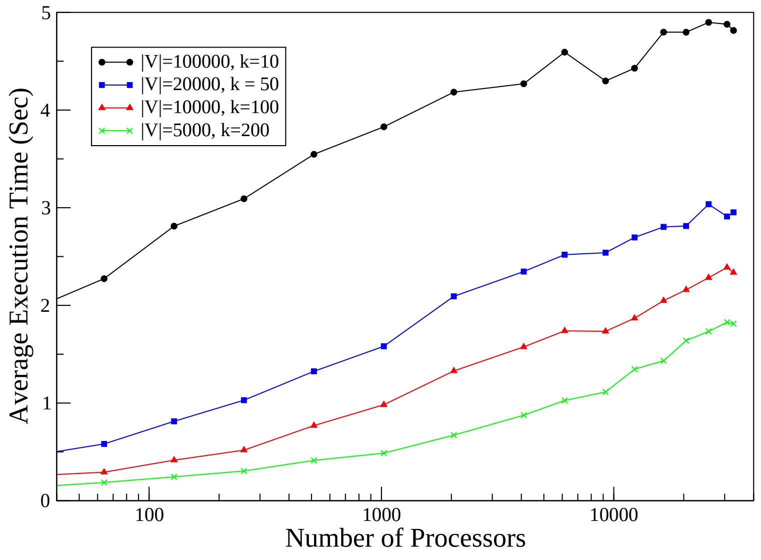


Figure 11: Weak scaling of *BFS* on BlueGene/L.

like FPMD, can also achieve outstanding floating point performance. Further, BG/L can outperform smaller Linux clusters in terms of time to solution in weak scaling mode for turbulence codes such as Raptor.

Classical MD simulations can be decomposed to use very little memory per task. As a result, they usually can effectively use Vitual Node Mode. However, many application domains have difficulty fitting within only 256 MB memory. Thus, the other applications provide their best performance under Co-Processor Mode. We note that the large overall memory available on LLNL's BG/L system supports applying these applications to previously unattainable problem sizes, as demonstrated by the BFS implementation.

4 Enhancing Scalability and Efficiency

Achieving the unprecedented scalability described in the previous sections requires careful attention to all aspects of the applications, including initialization routines and other aspects often ignored in performance optimization. Further, full scalability can entail mechanisms specific to the target architecture in order to take full advantage of its features. We now discuss some of these latter optimizations, which can be broadly classified as single node computation optimizations and inter-node communication optimizations.

4.1 Computation Optimizations

Matrix-Matrix Multiplies in ddcMD

We use hand-tuned linear algebra kernels, implemented primarily in C with some assembler, to exploit BGL's dual FPUs. In particular we focus on matrix-matrix multiplies as those are one of the fundamental operations in *ddcMD*.

We exploit the capability of the BG/L PPC440 architecture to issue one floating-point and one load-store instruction per cycle. Thus, we attempt to schedule the two in an interleaved fashion while using the two-way SIMD instructions that can potentially double FP performance. Further, we use a memory layout that ensures all operands for the kernels are found in the L1 cache, with its 16 byte/cycle bandwidth, approximately 90% of the time. We also take advantage of a more global picture than might be available to the compiler in order to use BG/L's SIMD instructions. We fuse a matrix multiplication ($A \times B \rightarrow C$) with a series of dot products immediately following the matrix multiplication.

Optimized FFTW in Qbox

In addition to matrix-matrix multiplies, which are optimized similarly to those in *ddcMD*, *Qbox* also relies on a large number of Fast Fourier Transformations. *Qbox* uses FFTW-GEL for BG/L (Lorenz, Kral, Franchetti, and Überhuber 2005), an FFTW 2.1.5 replacement (Frigo and Johnson 1998) that explicitly exploits BG/L's SIMD instructions.

At the heart of FFTW-GEL for BG/L is the Vienna MAP vectorizer (Franchetti, Kral, Lorenz, and Überhuber 2005). MAP two-way vectorizes large computational basic blocks by a depth-first search with chronological backtracking. It produces an explicitly vectorized FFTW codelet with solely two-way vector instructions and a minimum of data reorganization instructions. This process fuses scalar variables into vector variables, which requires fusing the corresponding scalar operations into vector operations.

The performance increase of FFTW-GEL due to SIMD instructions is large (near two-fold speedup) when measured on a hot L1 cache (e.g. by transforming the same data multiple times). The increase that we observe in *Qbox* is smaller, since the transformed data far exceeds the size of the L1 cache, and memory bandwidth limits performance. Nonetheless, a speedup of 20-25% was measured when comparing the FFTW-GEL library with the conventional FFTW 2.1.5 implementation running within *Qbox*.

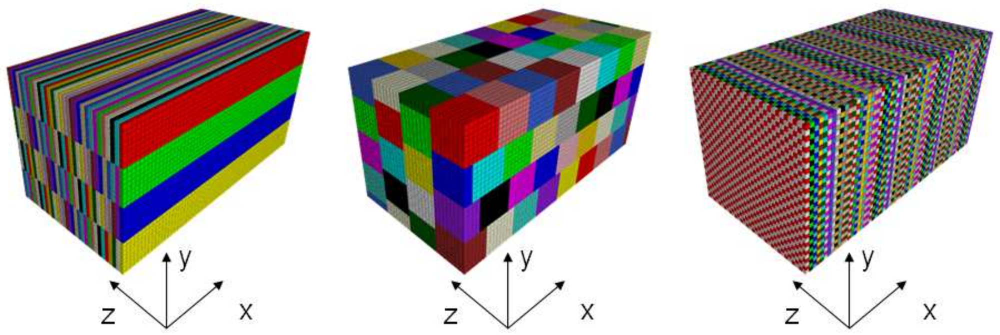


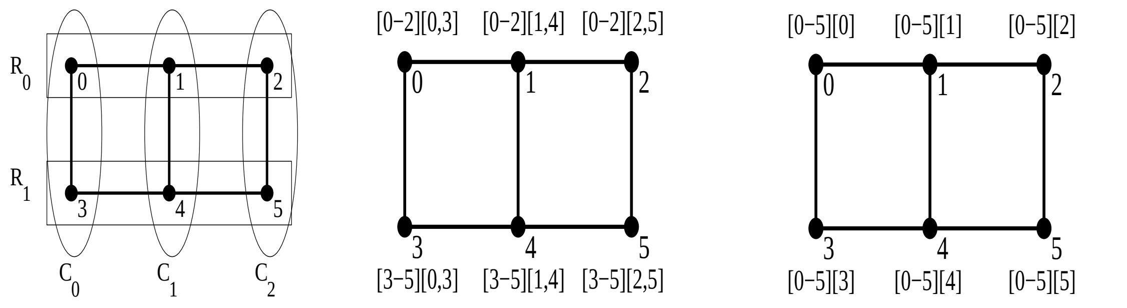
Figure 12: Three different node mappings for Qbox on 64K nodes (all elements in one column shown in the same color): individual planes (left), compact column mapping (middle), bipartite mapping (right).

4.2 Communication Optimizations

Task Mappings in Qbox

Unlike many applications for which a simple 3D domain decomposition naturally maps to a 3D torus architecture, the KS equations in the plane wave formalism do not exhibit any obvious way to map parts of the calculation to the torus. The data layout adopted in *Qbox* distributes the degrees of freedom describing electronic wave functions on a two dimensional process grid similar to the process grids used in the BLACS communication library (Blackford, Choi, Cleary, Azevedo, Demmel, Dhillon, Dongarra, Hammerling, Henry, Petite, Stanley, Walker, and Whaley 1997). MPI subcommunicators are defined appropriately in order to facilitate communication along rows and columns of the process grid. As discussed previously, these subcommunicators do not use the collective network due to its limited number of contexts. Thus *Qbox* performance is highly dependent on how tasks map to the torus network. Our results demonstrate that substantial performance benefits can result from mappings that are more complex than the default XYZ, YZX or ZXY orderings.

We explored several node mappings in order to optimize performance. Figure 12 illustrates three node mapping choices on the full 64K node system using the same logical processor topology and applications settings. By default, the individual columns are mapped in planes along the x-axis of the torus topology. Using this configuration, we achieved a sustained performance of 39.5 TFlop/s. In order to optimize the intra-column communication, we then chose a configuration that provides a compact mapping for each column. However, this scheme leads to a reduced communication performance across columns, mitigating



(a) A 2×3 processor array (b) Messages received after phase 1 (c) Messages received after phase 2

Figure 13: A reduce-scatter operation on a 2×3 processor grid (The notation [S][R] denotes a set of messages sent by processors in group S to processors in group R. The sending and receiving groups are represented as a range of comma-separated list of processors.).

any improvement within columns and leading to a slightly reduced performance of 38.2 TFlop/s. Only with a scheme that optimizes both row and column communication at the same time, as shown on the right side of the figure, do we achieve the best overall performance, in our case 64 TFlop/s.

Torus aware Collectives in BFS

We implemented optimized collectives for *BFS* on BG/L. They use a ring point-to-point communication that exploits the full torus connectivity. We improve the performance of these collectives by shortening the diameter of the ring in our optimization.

In this scheme, the collective communications are performed in two phases. We divide the processors in the ring into several groups and perform the ring communication within each group in parallel. To ensure that processors in a group can receive and process messages from the processors in all other groups, processors in each group initially send messages targeted to other processor groups. A processor sends messages to only one processor in each group in this stage (phase 1). These messages will eventually be received by all the processors in the targeted group during the ring communication (phase 2). The processes are mapped to processors in such a way that the processors in each group form a physical ring with wraparound edges.

Figure 13 shows a reduce-scatter operation implemented in this scheme on a 2×3 processor grid. The processors are grouped in two row groups and three column groups. After phase 1, each processor in a row group contains the messages from all the processes in the row group to the processes in the column group to which the processor belongs. After these messages are exchanged among the column processors in phase 2, each processor has received all its messages.

5 Observations and Lessons Learned

With the success experienced by application users on BG/L, we now have the opportunity to critically examine the system and the process by which it was built. This exercise will provide guidance to future supercomputer designers and can shape the supercomputing landscape of the next decade.

We derive many positive lessons from our partnership, including that successful systems emerge when the industrial leaders and the most advanced users in the supercomputing field collaborate closely. A key aspect of this collaboration was a constant focus on what could be built and what the applications could use while remaining on time and budget. Thus, the project involved many decisions that minimized risk, such as the choice of a readily available and well-tested processor design, in our case the Power 440 architecture, providing perhaps the most important overall lesson. Successful systems do not include every innovative idea but instead push a few well established ones to their limits.

Overall, the BG/L hardware design was very successful. Using a huge number of low power processors still supports excellent overall application performance assuming the application exhibits the necessary amount of parallelism. The latter, however, is typical for large scale scientific codes and hence does not impose a limitation in our environment.

Using a highly integrated design provides excellent main memory bandwidth, which significantly simplifies overall real application performance. Tight integration, including having main memory soldered to compute cards, also increases system reliability (although we have not discussed it, LLNL's system has an outstanding mean time between failure of approximately seven days). As seen here, 512MB per node (256MB per processor) supports a wide range of applications. However, many LLNL applications, particularly integrated multiphysics codes, will require more.

Our application experiences demonstrate that a torus interconnect can provide sufficient connectivity for point-to-point messaging. The extremely low latency global interrupt network can simplify programming and debugging since the cost of barriers becomes essentially limited to load balance. Further, the combining network for collective operations such as broadcasts and reductions can provide real performance benefits. However, task layout choice can significantly impact performance on BG/L even when the application performs little point-to-point communication. This issue partly arises from the complex collective communication patterns in scalable applications. However, the inability to use the combining network with MPI subcommunicators exacerbates it. Combining networks on future systems should support more than two deadlock-free concurrent contexts. Applications like Qbox could then optimize their torus task placement for the all to all communication across matrix columns and not have to consider the effect on reductions and broadcasts across the matrix rows.

Despite the overall success, we would change some aspects of the BG/L hardware now that we have the benefit of hindsight. We would increase the amount of memory per processor and the number of combining network contexts. We would also choose a processor design that includes hardware support of L1 cache coherence in order to simplify communication between processor cores, ease the implementation of communication libraries, and to reduce the effort required for applications to use boht cores for computation in Co-Processor Mode. Integrating a separate DMA engine, including scatter/gather support, for network transfers would also make it easier to use all cores for computation. Nevertheless, most system design choices performed well as demonstrated by the impressive performance across the applications discussed here.

Similar to the BG/L hardware design, our experiences affirm the choices of its system software design. In particular, a lightweight kernel for computation nodes provides performance reproducibility that is essential for large scale code optimization. Several BG/L developers have commented that this aspect of the machine allows them to evaluate the effect of coding changes intelligently, rather than having to guess if differences in two runs result from the changes or are just the random product of system noise.

For actually optimizing application performance, we have found that relying on highly optimized libraries routines is the key to success. In general, modern architectures are too complex to formulate system designs that require significant advances in compiler technology in order to achieve high performance. Such a strategy is at least as risky as relying on high levels of custom designed hardware components.

Again, we would change some aspects of the BG/L system software in future systems. In particular, we would include a mechanism for dynamic linking and loading. The restriction for static linking not only substantially limits the applications that can run on the machine; it also complicates tool development.

However, we envision a restricted dynamic mechanism that will not exceed physical memory limits and will exploit parallelism across nodes rather than full asynchronous shared library support.

Ultimately, design decisions for BG/L system software were by and large the correct ones. They support the programming model used by the vast majority of large scale applications with a familiar tool chain. LLNL users have long used MPI and are accustomed to the IBM compilers from experiences with the ASCI Blue Pacific and White machines. Overall, we see that minimizing risk, not only in the hardware design but also the choices for software, are the key to building a successful supercomputer.

6 Conclusions

The BlueGene/L system developed jointly by LLNL and IBM establishes that massively parallel systems with general purpose processors can achieve outstanding performance. With 65,536 dual core nodes, BG/L has required application programmers to stretch to unprecedented scaling levels. We discussed eight application from five different application domains: first principles molecular dynamics; classical molecular dynamics; dislocation dynamics; turbulence; and even discrete problems like breadth first search. We found that they met the scaling challenge presented by the BG/L design and provide guidance for improvements to that design. The results presented in this paper, including sustained performance of over 100 TFlop/s on a classical molecular dynamics simulation, demonstrate that the overall design of BG/L is a success: future supercomputers should use well tested processor designs integrated with multiple networks.

References

Adiga, N. and et al. (2002, November). An overview of the bluegene/l supercomputer. In *Proceedings of IEEE/ACM Supercomputing '02*.

Bachega, L., S. Chatterjee, K. Dockser, J. Gunnels, M. Gupta, F. Gustavson, C. Lapkowski, G. Liu, M. Mendell, C. Wait, and T. Ward (2004, September). A High-Performance SIMD Floating Point Unit Design for BlueGene/L: Architecture, Compilation, and Algorithm Design. In *Proceedings of the 2004 International Conference on Parallel Architectures and Compilation Techniques.*

Blackford, L., J. Choi, A. Cleary, E. Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammerling, G. Henry, A. Petite, K. Stanley, D. Walker, and R. Whaley (1997). ScaLAPACK Users. SIAM, Philadelphia.

Bulatov, V., W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, K. Yates, and T. Arsenlis (2004, November). A Performance and Scalability Analysis of the BlueGene/L Architecture. In *Proceedings of IEEE/ACM Supercomputing '04*.

Car, R. and M. Parrinello (1985). *Physics Review Letters* **55**, *2471.*

Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory. http://seesar.lbl.gov/CCSE.

Cook, A., W. Cabot, M. Welcome, P. Williams, B. Miller, B. de Supinski, and R. Yates (2005, November). Tera-scalable Algorithmms for Variable-Density Elliptic Hydrodynamics with Spectral Accuracy. In *Proceedings of IEEE/ACM Supercomputing '05*.

Crauser, A., K. Mehlhorn, U. Meyer, and P. Sanders (1998). A parallelization of Dijkstra's shortest path algorithm. *Lecture Notes in Computer Science 1450*, 722–731.

Davis, K., A. Hoisie, G. Johnson, D. Kerbyson, M. Lang, S. Pakin, and F. Petrini (2004, November). A Performance and Scalability Analysis of the BlueGene/L Architecture. In *Proceedings of IEEE/ACM Supercomputing '04*.

DelSignore, J. TotalView on Blue Gene/L. Presented at "Blue Gene/L: Applications, Architecture and Software Workshop", presentation available at http://www.llnl.gov/asci/platforms/bluegene/papers/26delsignore.pdf.

Franchetti, F., S. Kral, J. Lorenz, and C. Überhuber (2005). Efficient Utilization of SIMD Extensions. *Proceedings of the IEEE Special Issue on "Program Generation, Optimization, and Adaptation" 92*(2), 409–425.

Frigo, M. and S. Johnson (1998). FFTW: an adaptive software architecture for the FFT. In *Proceedings of ICASSP*, pp. 1381–1384.

Germann, T., K. Kadau, and P. Lomdahl (2005, November). 25 Tflop/s Multibillion-Atom Molecular Dynamics Simulations and Visualization/Analysis on BlueGene/L. In *Proceedings of IEEE/ACM Supercomputing '05*.

Grama, A. Y. and V. Kumar (1993). A survey of parallel search algorithms for discrete optimization problems.

Greenough, J., B. de Supinski, R. Yates, C. Rendleman, D. Skinner, V. Beckner, M. Lijewski, J. Bell, and J. Sexton (2005). Performance of a Block Structured, Hierarchical Adaptive Mesh Refinement Code on the 64K Node IBM BlueGene/L Computer. Technical Report LBNL-57500, Lawrence Livermore and Lawrence Berkeley National Laboratories.

Gygi, F. (2005). Qbox: a large-scale parallel implementation of First-Principles Molecular Dynamics. LLNL preprint.

Gygi, F., E. Draeger, B. de Supinski, R. Yates, F. Franchetti, S. Kral, J. Lorenz, C. Überhuber, J. Gunnels, and J. Sexton (2005, November). Large-Scale First-Principles Molecular Dynamics simulations on the BlueGene/L Platform using the Qbox code. In *Proceedings of IEEE/ACM Supercomputing '05*.

Han, Y., V. Y. Pan, and J. H. Reif (1992). Efficient parallel algorithms for computing all pair shortest paths in directed graphs. In *ACM Symposium on Parallel Algorithms and Architectures*, pp. 353–362.

Klein, P. N. and S. Subramanian (1997). A randomized parallel algorithm for single-source shortest paths. *J. Algorithms 25*(2), 205–220.

Laboratory, L. L. N. (2005, June). SLURM: Simple Linux Utility for Resource Management. http://www.llnl.gov/linux/slurm/.

Lorenz, J., S. Kral, F. Franchetti, and C. Überhuber (2005). Vectorization techniques for the BlueGene/L double FPU. *IBM Journal of Research and Development 49*(2/3), 437–446.

Moriarty, J. (1990). *Physics Review B* ***42****, 1609.*

Moriarty, J. (1994). *Physics Review B* ***49****, 12431.*

Moriarty, J. and et al. (2002). *Journal of Phyiscs: Condensed Matter* ***14****, 2825.*

Parrinello, M. (1997). From Silicon to RNA: the Coming of Age of First-Principle Molecular Dynamics. *Solid State Communications 103, 107.*

Rendleman, C. A., V. E. Beckner, M. Lijewski, W. Y. Crutchfield, and J. B. Bell (2000). Parallelization of structured, hierarchical adaptive mesh refinement algorithms. *Computing and Visualization in Science 3*(3), 147–157.

Schulz, M., D. Ahn, A. Bernat, B. de Supinski, S. Ko, G. Lee, and B. Rountree (2005, September). Scalable Dynamic Binary Instrumentation for Blue Gene/L. In *Proceedings of the Workshop on Binary Instrumentation and Applications.*

Söderlind, P. and J. Moriarty (1998). *Physics Review B* ***57****, 10340.*

Streitz, F., J. Glosli, and M. Patel (2006, June). Beyond Finite-Size Scaling in Solidification Simulations. *Physical Review Letters 96(22).*

Streitz, F., J. Glosli, M. Patel, B. Chan, R. Yates, B. de Supinski, J. Sexton, and J. Gunnels (2005, November). 100+ TFlop Solidification Simulations on BlueGene/L. In *Proceedings of IEEE/ACM Supercomputing '05.*

University of Mannheim, University of Tennessee, and NERSC/LBNL. TOP500 Supercomputing Sites. http://www.top500.org/.

Vetter, J., B. de Supinski, J. May, L. Kissel, and S. Vaidya (2005, August). Evaluating High Performance Computers. *Concurrency and Computation: Practice & Experience 17(10)*, 1239–1270.

Yoo, A., E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and U. Çatalyürek (2005, November). A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L. In *Proceedings of IEEE/ACM Supercomputing '05.*