Performance Analysis of Blue Gene/L Using Parallel Discrete Event Simulation

Ed Upchurch
Paul L. Springer
Maciej Brodowicz
Sharon Brunett
T. D. Gottschalk

Center for Advanced Computing Research
California Institute of Technology

Abstract

High performance computers currently under construction, such as IBM's Blue Gene/L, consisting of large numbers (64K) of low cost processing elements with relatively small local memories (256MB) connected via relatively low bandwidth (0.0625 Bytes/FLOP) low cost interconnection networks promise exceptional cost-performance for some scientific applications. Due to the large number of processing elements and adaptive routing networks in such systems, performance analysis of meaningful application kernels requires innovative methods. This paper describes a method that combines application analysis, tracing and parallel discrete event simulation to provide early performance prediction. Specifically, results of performance analysis of a Lennard-Jones Spatial (LJS) Decomposition molecular dynamics benchmark code for Blue Gene/L are given.

1. Introduction

Caltech's Center for Advanced Computing Research (CACR) is conducting application and simulation analyses of Blue Gene/L[1] in order to establish a range of effectiveness of the architecture in performing important classes of computations and to determine the design sensitivity of the global interconnect network in support of real world ASCI application execution.

Due to the large number (64K) of processing elements and adaptive routing networks in Blue Gene/L, performance analysis of meaningful application kernels requires innovative methods. This paper describes a method that combines application analysis, tracing and parallel discrete event simulation to provide early performance prediction. Specifically, results of performance analysis of a Lennard-Jones Spatial (LJS) Decomposition molecular dynamics benchmark code for Blue Gene/L are given.

Cycle-by-cycle level simulation a 64K node system running parallel applications is necessary for detailed machine design and IBM has done this, but runtimes are too long for spanning a number of applications at application iteration level. In lieu of this, our team is taking a statistical approach using parameterized models of the applications (workloads) and statistical (queuing) models of processing node message traffic derived from traces produced by the computational experiments. All 64K nodes of BG/L are explicitly represented, but the model is not cycle-by-cycle accurate although effects of adaptive routing and network contention are of necessity reliably modeled. For a further increase in simulation performance, an optimistic parallel discrete event simulation (PDES) framework is employed.

Our methodology was applied to select ASCI applications in order to provide statistical workloads for performance analysis using the parallel BG/L simulator. These applications stress load balancing, multiple languages, and dynamic behavior with respect to CPU/memory/communications usage throughout execution.

2. Approach

A benchmark code for an important class of applications, Lennard-Jones Spatial Decomposition (LJS) molecular dynamics [2], was selected as an example of an important class of numerical problems and a methodology of algorithm analysis, tracing and simulation was used to produce performance estimates for a 64K node Blue Gene/L.

LJS is an example of a fast parallel algorithm for short range molecular dynamics applications simulating Newtonian interactions in large groups of atoms. Such simulations are large in two dimensions: number of atoms and number of time steps. The spatial decomposition case was selected where each processing node keeps track of the positions and movement of the atoms in a 3-D box. Since the simulations model interactions on an atomic scale, the computations carried out in a single time step (iteration) correspond to femto-seconds of real time. Hence, a meaningful simulation of the evolution of the system's state typically requires a large number (thousands) of time steps. Point-to-point MPI messages are exchanged across each of the 6 sides of the box at each time step. The code is written in Fortran and MPI.

Application analysis played a major role in characterization of LJS to determine BG/L expected performance. Traces on a small number of nodes were used to identify phases of execution and resource consuming compute kernels and mix of MPI messages. Analysis however was necessary to derive the relationship between:

1. number of application "grid elements"/node vs. # physical BG/L nodes
2. communications vs. mapping onto BG/L physical nodes
3. number of application "grid elements"/node vs. physical node memory required
4. compute time vs. #"grid elements"/node vs. #BG/L physical nodes
5. comms distribution (type, length) vs. #"grid elements" vs. #BG/L physical nodes
6. i/o vs. #"grid elements"/node vs. # BG/L physical nodes

Algorithm analysis coupled with simulation results will also enable evaluation of potential application design enhancements to take advantage of BG/L's specific architectural strengths.

Application tracing of LJS is used to:

1. Gain insight into scaling up to 65K nodes and to verify application algorithm and code analysis – although it is very limited in scope (up to 1024 nodes, less than $1/60^{th}$ of BG/L's expected size)
2. Collect statistical data (via use of trace analysis tools) on message distribution, length, compute times etc. and give insight into statistical variation for application execution phases

And finally a parallel BG/L simulation model is used to estimate LJS performance on BG/L for a variety of mappings to various numbers of BG/L compute nodes. Performance metrics including: time to solution; network utilization and scalability are reported.

3 The Molecular Dynamics (MD) Problem

The basic computational problem is the evaluation of the total force on a particle, written as a sum over pair-wise forces arising from all other particles in an ensemble:

$$\mathbf{F_j = \Sigma_{i \neq j} F_{ij}}$$

The pairwise force Fij is provided by some dynamical model (e.g., described by a Lennard-Jones potential). It depends on the positions of the two particles involved and possibly on other state variables of the physics model.

The kinematic state of an individual particle at a time t is specified by the particle's position and velocity. The force equation gives the acceleration that is used to update the particle's state through some small time step $\mathbf{\Delta T}$. ("Real" MD codes generally use more sophisticated integrators. This is a per-particle computational cost and does not affect the scaling discussions of this paper.)

The essential simplifying assumption for MD models is limited range of the pairwise forces:

$$\mathbf{F_{ij} = 0, \quad |r_{ij}| > r_C}$$

The force cutoff r_C is a parameter of the model. Given this assumption, the total computational cost for a single update cycle is approximately

$$\mathbf{Cost = N_{TOT} * (\alpha + \beta * N_{NBD} + \delta)}$$

Where

1. N_{TOT} is the total number of particles
2. N_{NBD} is the (typical) number of particles in the force neighborhood of an individual particle
3. α is the cost of integrating the equation of motion for an individual particle over the (small) time step.

4. β is the cost of computing a single inter-particle force F_{ij}
5. δ is the cost of finding/enumerating particles in the neighborhood of the current particle of interest.

The coefficients 'α' and 'β' are fairly straightforward and could presumably be measures by profiles of single-processor executions of an actual code. The "finding" coefficient 'δ' is a bit more complicated and will be discussed in more detail below.

3.1 Spatial Decomposition Algorithm: Qualitative Overview

The Spatial Decomposition (SD) algorithm for parallel MD can be described as follows:

1. The physical volume is divided into a (regular) grid.
2. Each grid cell (see Figure 1) is assigned to a processor, and a processor is responsible for performing the force calculations and state updates for all particles (nominally) within the cell.
3. Force computation requires state information for some particles owned by other processors – the lightly shaded area in Figure 1. These are acquired by a communications phase at the start of each computational step.
4. Particles will occasionally drift across processor boundaries. These processors remain the responsibility of the original parent processor during the basic (Communicate,Update) cycle outlines in steps 2 and 3. Reassignment of particles to processors according to the cell boundaries is done periodically but (far) less frequently than the basic update cycle.

The communications for the data sharing of Step 2 are straightforward and involve synchronized messaging within the grid. The communications phase is a number of pairwise data exchanges between (logically) neighboring processors. The step are as follows (see Figure 2):

1. Processors send all particles within the interaction of a horizontal boundary to the other processor at that boundary, at the same time accepting particles from that processor.
2. The "vertical" sharing in step (1) is then repeated in the other physical dimensions.

During the second/horizontal sharing, a processor will generally send some particles it received during the preceding vertical sharing. This is the mechanism for acquiring relevant data from the "diagonal neighbors".

3.2 Complications and Simplifications

Ignoring the periodic, lower frequency reassignments of ownership of particles that drift across cell/processor boundaries, the basic update cycle for any one processor has two parts:

1. **Communications**: Retrieve current positions of "boundary" particles assigned to neighboring processors. Send current state of boundary particles known by this processor to neighbors
2. **Computation**: Perform the force evaluation and state update calculation for all particles owned by the processor.

The amount of communications depends on the relative magnitudes of the force range (r_S) and the width (d) of a physical grid cell assigned to a given processor. If $d < r_S$, then the current positions must be exchanged across multiple hops in the communications scheme of shown in Figure 2. In the other cases, we can approximate

$$\mathbf{N_{COMM} = \lambda N_{TOT}}$$

For some scale factor λ,

λ= Fraction of local particles interesting across a single boundary.

The analysis here makes this assumption, ignoring the more complex $d < r_S$ case.

The low frequency rearrangement of particles across cell boundaries will also typically involve some (smaller) fraction of the local particles. It is during this lower frequency exchange that Plimpton recommends reconstruction of the the data structures used for efficient near neighbor searches in the force computation loop. For now, the scaling behaviors and expectations for this low-frequency particle migration and search tree reconstruction are ignored.

3.3 Parameterized Model: Performance and Scaling

The activity of an individual processor for a single computational cycle can be modeled by a simple "time line", as shown in Figure 3. The activities and expected costs/times for these components are as follows:

1. **Communication**

   In each of three dimensions and two directions per dimension, the processor exchanges data with its neighbor. The amount of data exchanged is

   $$\mathbf{Data = \lambda * N_{LOC} * (Individual\ Datum\ Size)}$$

   A typical datum size would be three doubles for position and one int for particle ID. This gives the size of the message. Actual communications costs will depend on the location of the logically adjacent processor within the communications network.

2. **Computation**

   As described above in Section I, the cost/time for the computational phase can be written as

$$\mathbf{Cost = N_{LOC} (\alpha + \gamma N_{LOC})}$$

   Where, for simplicity, the data structure maintenance cost (δ) has been ignored.

3. **Synchronization/Waiting**

   The pairwise data exchanges of Figure 2 are synchronized. This will introduce various communications delays that have been collectively lumped into a single Wait Time before the start of the next simulation step.

In the above,

$$\mathbf{N_{LOC} = N_{TOT}/N_P}$$

Is the "local" particle count – the number of particles out of N_{TOT} total particles owned by one of N_P total processors. The N_{NBD} "force neighborhood" count from Section I has been estimated as some fraction of the Local count – essentially an assumption of approximately uniform particle densities across the system.

The overall scaling behavior will clearly depend on which of the parameters N_{TOT}, N_{LOC}, N_P are held fixed.

This provides a simple three parameter model for approximating the LJS algorithm in a Speedes-based simulation. The various points on the time axes of Figure 3 are the discrete events for the simulation. The communications message size estimates the total byte count for each message in terms of one parameter (λ) and the Computation cost is a simple two-parameter representation.

4 Tracing Overview

LJS (**L**ennard-**J**ones with **S**patial decomposition) target code is a molecular dynamics application developed by Steve Plimpton at Sandia National Laboratories [2]. It performs thermodynamic simulations of a system containing fixed large number (millions) of atoms or molecules confined within a regular, three-dimensional domain. Since the simulations model interactions on atomic scale, the computations carried out in a single timestep (iteration) correspond to femtoseconds of real time. Hence, a meaningful simulation of the evolution of the system's state typically requires a large number (thousands and more) of timesteps.

The particles in LJS are represented as material points subjected to forces resulting from interactions with other particles. While the general case involves N-body solvers, LJS implements only pair-wise material point interactions using the derivative of Lennard-

Jones potential energy for each particle pair to evaluate the acting forces. The velocities and positions of particles are updated by integrating Newton's equations (classical molecular dynamics). The interaction range depends on the modeled problem type; LJS focuses on short-range forces, implementing a cutoff distance r_c outside which the interactions are ignored. The computational complexity of $O(N^2)$, characteristic for systems with long-range interactions, is therefore substantially reduced.

LJS deploys spatial decomposition of the domain volume to distribute the computations across the available processors on a parallel computer. The decomposition process uniformly divides a parallelepiped containing all particles into volumes equal in size and as close in shape to a cube as possible, assigning each of such formed cells to a CPU. The correctness of computations requires the positions of some particles (depending on the value of r_c) residing in the neighboring cells to be known to the local process. This information is exchanged in every timestep via explicit communication with the neighbor nodes in all three dimensions (for details see [3]). LJS also takes the advantage of Newton's third law to calculate the force only once per particle pair; if the involved particles belong to cells located on different processors, the results are forwarded to the other node in a "reverse communication" phase.

Besides communications occurring for each iteration, additional messages are sent once every preset number of timesteps. Their purpose is to adjust cell assignments of particles due to their movement. To minimize the overhead of the construction of particle neighbor lists, LJS replaces r_c with an extended cutoff radius r_s ($r_s > r_c$), which accounts for possible particle movement before any list updates need to be carried out. Due to a relatively small impact of that phase on the overall behavior of the application, it is ignored it in this analysis.

4.1 LJS Benchmark Experiment Configuration

The runtime parameters of the simulation along with their values are listed in Table 1.

The total number of particles, N, is given as

$$N = 4\, n_x\, n_y\, n_z\,,$$

where n_i are integers (there is a fixed average of four particles per unit cube). The problem is executed on a grid of P processors, such that

$$P = p_x\, p_y\, p_z, \text{ with } p_i = n_i/k_i \text{ where } k_i \text{ are integers.}$$

In the benchmarks $k_i = 50$, hence the problem size was 50x50x50 (or 500,000 particles) on a single, 100x100x50 on four, 100x100x100 on eight and 200x200x200 on 64 processors. Such configurations require approximately 200MB of memory per CPU for all LJS data structures well within expected user memory for Blue Gene/L. Note that the cutoff distances are significantly smaller than the linear dimensions of the domain fragment assigned to a single processor ($50 \cdot alat \cong 84$), hence the spatial decomposition algorithm is performing efficiently (time spent in all communication phases is

significantly smaller than the total computation time and didn't exceed 15% of the application runtime in the experiments).

LJS initializes its data structures by assigning particle positions on a regular 3-D mesh (thus emulating a crystal lattice) and computing velocity vectors to satisfy the initial temperature requirement. The velocities are otherwise random in magnitude and direction. In the next few time steps of the simulation the particles move from their positions on the grid (the crystal melts). Therefore, to capture the application behavior as close to the average (no imbalances of particle counts between processors), tracing was limited to the first five iterations.

4.2 ETF Instrumentation

In order to extract more detailed runtime information, ETF (Extensible Tracing Facility) [4] was used to instrument the application code. The instrumentation's goal was to provide low-level instruction counts executed at the user level, obtain timing information, register parameters used by MPI routines for message passing and mark starting and ending points of important execution phases.

4.2.1 ETF Counters and Timers

ETF is capable of accessing high-resolution timers and hardware event counters on select platforms. Currently, such support exists for IBM SP2 (Power processors) via a PMAPI interface. Using Power architecture in our experiments is convenient, as BG/L processors are based on a modified version of the PowerPC core, which shares significant elements of ISA and hardware features with the Power processor line. PMAPI supports up to eight 64-bit event counters, however, they cannot be assigned to counters arbitrarily and that limits the effective number of events monitored concurrently. Another constraint is caused by OS overhead (1200..1500 cycles per readout of a counter set), which may produce skewed results if the counters are accessed too frequently. In our experiments we decided to restrict the monitoring to the execution in user mode only, as the kernel mode offers little insight into application behavior and cannot be properly verified due to lack of the source code. The events of interest included: the number of CPU cycles spent executing the code, the number of instructions completed and the cumulative count of all FPU operations (this required summing the counts of instructions retired by both floating point units of the processor).

4.3 MPI Communication

LJS uses a small subset of MPI-1 calls for message passing. The collective calls (*Barrier*, *Bcast*, *Allreduce*) are invoked only during the setup phase and when computing the thermodynamic state of the system (typically at the end of execution). Throughout the simulation, the bulk of data is transferred by point-to-point calls (blocking *Send* and non-blocking *Irecv*, which enable overlapping of bi-directional transmissions). For the parameter set listed above, the messages originating from each node are emitted to its six nearest neighbor (in a 3-D grid) nodes only. Due to the use of a periodic Cartesian

communicator, particles migrating outside the domain from boundary cells in any dimension, appear in the opposite boundary cell in that dimension.

To simplify the trace analysis, ETF was configured to register the message sizes and destination nodes of point-to-point communications. Memory reference tracking, which includes message buffer pointers and maps of MPI datatypes in the trace, was disabled.

4.4 LJS Execution Phases

The source code of LJS was augmented with calls injecting markers at the endpoints of the following phases:

- Setup and initialization (procedures: *input, setup_general, setup_memory, setup_comm*, *setup_neigh*, *setup_atom, scale_velocity, exchange, borders, neighbor*)
- Iteration of the main loop (*integrate*):
    - Calculation of the new positions of particles
    - Communication: update of the positions of remote particles (*communicate*)
    - Computation of forces (*force_newton*)
    - Reverse communication: propagation of forces (*reverse_comm.*)
    - Calculation of particle velocities
- Final thermodynamics evaluation and printout (*thermo, output*)

4.5 Computational Profile

The computational workload was very consistent from iteration to iteration and across the nodes. This is expected due to uniform initial distribution of particles and symmetric neighborhoods of each cell. Table 2 shows counter values collected for the setup, intermediate phases of the fourth time step of the simulation (which is representative for other iterations as well) and finalization phase for different number of processors.

The only significant inconsistencies are variances in event counts for the communication phases. This is understandable, since the message passing is inherently non-deterministic. For example, messages of identical sizes can be split into different number of packets depending on the transient condition of the interconnect network and hence the overhead of message fragmentation and reassembly may not be identical. Note that even though the network traversal time should be excluded from timings in user mode, the actual behavior is strongly implementation dependent; if the MPI library uses busy waiting to poll for incoming messages, this fact will be reflected in counts. The global trend of increasing the overhead with the problem size is, however, sustained.

LJS deploys a "leapfrog" integrator, whose operation is expressed as (only position computation shown; velocity calculation is identical in complexity with properly adjusted *dt*):

$$x_i(t+1) = x_i(t)+v_i(t+1/2)\ dt, \text{ for dimension } i = 1, 2, 3 \text{ in iteration } t.$$

This is in nearly perfect agreement with the FPU counts: 500,000 particles per CPU with 3 dimensional components yield 1.5 million operations. The compiler takes advantage of the fact that the Power ISA includes a multiply-add instruction; otherwise the FPU counts would be twice as high. The number of cycles spent in velocity calculation phase is higher than that of position integration, since the previous values of velocity vectors need to be preserved for thermodynamic state computations, while the old position vectors are simply overwritten. The copy operation doesn't use the FPU, hence the additional overhead manifests itself only in increased instruction/cycle counts. Still, by far the most dominant portion of each time step is devoted to the force computation, thus justifying the presence of reverse communication step.

4.6 Communication Profile

Due to symmetry of the problem decomposition and repeatability of parameters passed to MPI calls, only one-iteration behavior on a single processor was analyzed. The results are collected in Table 3 listing destinations and sizes of messages transmitted from rank 0. The send order in each communication phase is reflected by the row position (entries closer to the top of the table are sent earlier).

This scheme is repeated in every time step. Differences across the nodes are relevant only to destination rank numbers, but they stay fixed throughout the execution for a given sender node. Note that the number of messages is reduced when running on less than 8 processors. This is because 2^3 CPUs is the smallest configuration where the computational domain can be decomposed into at least two partitions along each dimension. A small inefficiency of LJS may be observed when running on less than 64 processors: the messages within the same communication phase are emitted to repeated destinations. When scaling beyond 64 CPUs, message sizes and number of destinations stay fixed as long as the size of local grid on every processor is preserved.

4.7 Algorithm Scaling

To verify the characteristics of program execution for other problem sizes, LJS was traced with reduced grid size of $n_x = n_y = n_z = 100$ on 64 processors. The computational workload parameters (fourth iteration only) and message sizes are given in Table 4.

As can be easily seen, the computational workload decreased proportionally to the problem volume (2^3 times, as the problem size in each dimension was halved). The memory allocation for LJS data arrays was 26.5MB per processor, again – roughly 1/8 of that required for 200x200x200 configuration. Message sizes were reduced approximately four times, what agrees well with the assumption of communication volume being proportional to the cell surface area. Note that the ideal ratio of four is observed only for the exchanges along the first dimension. This figure is distorted for transmissions along the second and third dimension due to the fact that the presence of volume characteristic increases in subsequent data sends.

The spatial decomposition algorithm implemented in LJS behaves consistently over a wide range of grid sizes. The anomalies resulting from the cutoff distance r_s being comparable with the physical dimensions of a sub-grid assigned to a single processor arise for relatively small problem sizes, for which the communication overhead nearly always exceeds the cumulative duration of computations. To investigate such a case, the program was configured to run a 20x20x20 problem (on 64 CPUs this yields a 5x5x5 grid per processor) with artificially increased values of r_c = 10 and r_s = 11.2. The communication characteristics are given in Table 5 with the "reverse" communication phase omitted for brevity.

Since r_s is longer than the linear dimension of the local sub-domain ($d = 5 \cdot alat \cong 8.4$), the communication must involve not only the immediate neighbors of a processor, but also cells located one more grid "hop" away (because $d < r_s < 2d$). As LJS processes don't communicate with the remote neighbors directly, the data are passed in multiple steps through the immediate neighbors' buffers. Unlike in the typical scenario described in section 4.2, the ratio of communication volume along the third dimension (the last four entries in the table) to that of the first dimension (the first four entries) is significantly larger due to much larger final volume of data accumulated from neighboring cells within the cutoff distance in all three dimensions compared to that for just one dimension. The memory consumption, 10.7MB per processor, also deviates from the simplistic estimates, most likely due to excessive buffer space required for communication. However, such situations arise rarely in practical short-range problems when executed on machines with sufficient amount of memory per node.

5 BG/L Parallel Simulation Overview

In order to simulate the full size Blue Gene/L system of 64K processing nodes for meaningful portions of ASCI applications, it has been necessary to resort to parallel discrete event simulation methods to produce models that can themselves run on parallel machines. Our goal is to extract as much parallelism as possible from the simulation. This goal has led to the use of optimistic simulation management methods.

In a discrete event simulation, care must be taken that all events are executed in the proper time order. In a sequential implementation a simple sorted global event list satisfies this requirement. In a parallel discrete event simulation while individual event lists on each simulation node are ordered in time, there is no global event list. Two events executed on separate nodes must be coordinated in order to avoid a causality error.

Mechanisms that address causality errors fall into two categories: conservative and optimistic [5]. Conservative algorithms avoid causality errors by constraining the operation of the simulation so that events that could possibly cause causality errors are properly serialized. Optimistic algorithms fully exploit available parallelism by allowing causality errors to occur, but detect this situation and force event rollbacks. Additional overhead is incurred but practice has shown that this cost is generally offset by the extra parallelism extracted.

It is expected for BG/L simulation that a PDES using optimistic time management will be more efficient than conservative methods at handling workloads involving uneven workload distributions, especially for the case where the load imbalances may exist at any instant in simulation time, but that over the course of the entire simulation the total loads on each node balance. This is because optimistic methods allow for *temporal load balancing*. Nodes are allowed to run at different rates, so that if a node temporarily has a heavier load, it is allowed to fall back in simulation time, and can catch up later when its load is eased.

A BG/L model was developed using SPEEDES, an optimistic parallel simulation framework developed in the early 1990's by Steinman [6]. By default, SPEEDES uses a synchronization algorithm called *breathing time warp* based on the concept of virtual time developed by Jefferson [7]. SPEEDES modifies Jefferson's original *time warp* concept by placing a limitation on the number of rollbacks that may occur in the course of the simulation. This algorithm uses a time window to prevent runaway objects from generating excessive numbers of rollbacks. However the choice of algorithm is governed by a runtime parameter that may be modified to remove any such limitations, allowing a pure Time Warp based algorithm to be use.

When it came time to install SPEEDES the decision of whether to install it on an SGI Origin 2000, or a Beowulf cluster had to be taken. Steinman recommended the Origin because it uses a shared memory model for communications between processors, which, for SPEEDES, is much more efficient than using a message passing model such as MPI found on the cluster. In addition, SPEEDES has been used for a battlefield simulation that handled 1,000,000 simulation objects running on 100 Origin nodes. This indicated feasibility that SPEEDES would scale efficiently on the Origin to handle the 64K nodes of BG/L. Because SPEEDES had previously been ported to the Origin, it was fairly straightforward to install on the 128 node Origin 2000 at JPL.

5.1 Model Development

Blue Gene/L uses a 3-D torus based network for point-to-point communications between nodes[1][8]. The torus router that exists on each BG/L node is modeled including the 6 injection FIFOs, as well as FIFOs associated with the 6 input and output network links at each node. Each link has associated with it two virtual channels that are used for adaptive routing, and 1 virtual channel used for deterministic routing. The latter is used for deadlock avoidance, and only when congestion prevents a packet from being adaptively routed. Tokens are used for flow control between routers. Virtual cut-through routing is used to minimize latencies.

The fundamental programming construct in SPEEDES is the simulation object. Each such object communicates to other objects by sending and receiving time-stamped messages. Receipt of a message eventually triggers the receiving object to process that message. The simulation time of the received message becomes the simulation time associated with the corresponding object when it executes. If an object executes during a certain time slice and is not rolled back, that event is said to be *committed*.

Each network node as a unique SPEEDES object and each network packet is modeled as a message in the simulation. This level of granularity is needed because of the complexity involved in the adaptive routing being used by the network. As congestion builds up in one part of the network, traffic patterns change in an attempt to route messages around the congested area.

Simulation messages are small, and only contain information relating to the transfer process, such as the origination node, destination node, and time of origination. When a message is sent from one object to another, it triggers an arbitration process in the receiving object that determines whether the message needs to be sent on, and if so, what route it should take. In parallel with this, the same object is examining its workload queue to determine whether new messages are being generated. Information about congestion on that node is sent back to the originating node to assist in flow control and the adaptive routing algorithm.

Each message injected in the simulation resulted in an event. An event was also generated each time the message was received, whether it was received on the originating node when it was first injected, or received by a destination or intermediate node. The network in our model is a three dimensional torus that allows packets to be sent in either direction for each dimension. If *x*, *y*, and *z* represent the size of the network in each dimension, each packet requires at most 1/2 * (x + y + z) hops to reach its destination. On the average, a randomly generated packet will require 1/4 * (x + y + z) hops. Using this information it became fairly easy to calculate the approximate number of total events. If *m* is the total number of messages, the expected number of events is close to m * [1 + 1/4 * (x + y + z)].

5.2 Random Workload Scaling Experiment

A simple base case experiment was performed to determine the performance, scaling, and use of SPEEDES. This case, while modeling the BG/L nodes and network, did not exercise BG/L message flow control or adaptive routing but instead used simple dimension ordered routing. The experimental workload consisted of a uniform burst of 256 byte packets injected at each BG/L node at simulation start time destined for randomly chosen nodes in the network. Results of this experiment are shown in Table 6.

With the large numbers of real messages being sent between physical nodes and a small amount of computation used to process each message, good simulation speedup was not expected. The focus has been on scaling the size of the simulation to 64K nodes. Nevertheless a small series of speedup tests, yielded a pleasant surprise. A speedup of 3.9 was measured when going from 4 physical processors to 16 physical processors for simulating 4096 BG/L nodes. Additionally, the simulation scaling performance was within about 10% of ideal, except for the 128 Origin processor case (see Figure 4). Because the 128 processor case used all available nodes of the Origin, one of the processors was busy running some of the standard O/S tasks as well as the simulation and possibly this caused the deviation shown in Figure 4.

5.3 LJS Molecular Dynamics Experiment

This experiment was driven a statistical workload generated from the message passing pattern from the LJS molecular dynamics application. Full support for adaptive routing and flow control was built into the software. The application is comprised of a number of cycles, with each cycle consisting of a compute stage and a communication stage. During each communication stage, a message is sent from each cell to its six immediate neighboring cells in three dimensional space. Messages average about 1750 packets in size, with each packet holding 256 bytes. The model maps a single cell to a single BG/L node, and only simulates a single communication stage, because of the repetitive nature of the communication and computation cycles.

The BG/L simulations were run on an SGI Origin 2000 with 128 R12000 processors available, each running at 300 MHz. These are configured as 2 processors per node. With each node containing 1 GB of RAM, it has a total of 64 GB of RAM.

Two cases were modeled. In the first case, ideal placement of cells is assumed, i.e. nearest neighbor cells are located on the physical nearest neighbor BG/L nodes. The second case assumes cells are mapped randomly to BG/L nodes. In the ideal first case, all messages move only one hop, from the originating cell to its nearest neighbor. Under this scenario, no flow control is needed: packets are collected on the receiving node as soon as they arrive, and need not be passed on. Maximum use of the bandwidth, and good scaling were observed. The model sizes used were 4K, 8K, 16K, 32K, and 64K BG/L nodes, with the ratio of one application cell per BG/L node remaining constant. These were run on 8, 16, 32, 64, and 128 Origin 2000 processors, respectively. Because not all messages were exactly the same size, the upper limit of bandwidth utilization one could expect to see was 82%. For each model size, 81% or better (see Figure 5) was observed.

Scaling was fairly flat, except for the 64K node case running on 128 Origin nodes, which was again attributed to system overhead (see Figure 6). For the largest size simulated transmission of almost 700 million packets was completed.

For the second case, random mapping of application to BG/L nodes, model sizes of 64 and 512 BG/L nodes, both run on 8 nodes of the Origin have been run. Because of the high volume of packets sent, and the multiple hops required for each delivery, one can see the effects of congestion in the network. Again using the figure of a maximum possible bandwidth utilization of 82%, we see 58% and 54% usage respectively (see Figure 7).

For BG/L the workload is specified for an average iteration by 1.48 seconds compute time, 3 ms of communications time for the “best” case independent of the number of nodes (not so for the random case which is 13 ms for 64 nodes and rises to 25 ms for 512 nodes) for sending 3.6Mbytes of data/node. These numbers are for the problem size defined and traced where there are 50^3 x 4 atoms/processing node (or 500,000 atoms/node). Results shown in Figures 5 and 6 are for the case where the problem size is

scaled with the number of nodes so that the number of atoms/node remains constant. For example the total problem size for 512 nodes is 512 x 500,000 atoms or approximately $2.56x10^8$ atoms.

Table 7 shows a comparison between cases (both best case mapping to BG/L nodes) where the problem size was kept fixed at 500,000 atoms/processor and a fixed problem size of 500,000 atoms. Figure 8 shows the preliminary results for another case where the problem size was held constant at a larger number of atoms than 500,000 (at $3.2x10^7$ atoms) and the number of processing elements were scaled. A crossover point at 32 nodes is seen where communications begins to dominate for the random distribution case. No such point is seen for the "best" nearest neighbor mapping. Table 8 shows a comparison of BG/L single node estimated performance with some other parallel machines. Data for the other machines comes from Plimpton [9].

A further experiment was conducted in which the injection of packets was throttled by a delay in the model at each node. Since all application nodes synchronize the exchange of data with their nearest neighbors, there is a burst of messages injected. In the case where cells are mapped randomly, this burst generates significant conjection in the torus as shown in Figure 7. Table 9 shows that slightly better performance can be obtained by "throttling" the injection of packets into the network.

6. Conclusions and Future Work

Several observations can be made from the experiments:

- SPEEDES has shown itself to be a valuable tool for the purposes of simulating a massively parallel machine.
- PDES enabled simulation of the full set of 64K BG/L nodes for full application workloads
- placement of application nodes on the torus to reduce hops can result in significant communications performance increase
- throttling injection of packets into the torus for highly bursty traffic appears attractive in order to increase overall communications performance

Planned future work includes:

- Investigate additional applications, including Quantum Monte Carlo (QMC) [10]; 3-D Adaptive Mesh Refinement (AMR3D) [11]; Magnetic Hydro Dynamics (MHD) [12]; multiscale polycyrstalline [13]; a new Lagrangian-Eulerian Shell-Fluid coupling algorithm, CONTACT [14]
- Add Checkpoint and restart capabilities to further extend simulation capabilities
- Explore various optimistic time management techniques to determine the effect on simulation performance, especially for unbalanced workloads

Acknowledgments
This work was performed under contract with Lawrence Livermore National Laboratory, Contract No. B520721 (Applications Requirements Machine Model Simulator). Time on the SGI Origin 2000 was provided by JPL Institutional Computing and Information Services and the NASA Offices of Earth Science, Aeronautics, and Space Science.

References

[1] Adiga, NR, et al, "An Overview of the BlueGene/L Supercomputer." In *Proceedings of SC2002*, November, 2002.

[2] Plimpton, S., "Fast Parallel Algorithms for Short-Range Molecular Dynamics", Journal of Computational Physics 117, 1-19 (1995)

[3] Gottschalk, T., "Scaling and Complexity: Spatial Decomposition Molecular Dynamics", CACR-2003-193, Center for Advanced Computing Research, California Institute of Technology, Pasadena, CA, May 2003

[4] Brodowicz, M., Brunett, S., "Blue Gene/L: Applications and Tracing",Applications, Algorithms, and Architectures Workshop, Lake Tahoe, 13-14 August 2002.

[5] Fujimoto, Richard M., "Parallel Discrete Event Simulation." *Communications of the ACM*, Vol. 33, No. 10 (October 1990), pp. 30–53.

[6] Steinman, Jeffrey, "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete-Event Simulation." In *Proceedings of the SCS Western Multiconference on Advances in Parallel and Distributed Simulation* (PADS91), vol 23, 1 (1991), 95-103.

[7] Jefferson, D., "Virtual Time." *ACM Trans. Prog. Lang. and Syst.* 7, 3 (July 1985), pp. 404-425.

[8] Heidelberger, P., and Steinmacher-Burow, B., "Overview of the BG/L Torus Network." http://www.llnl.gov/asci/platforms/bluegene/talks/heidelberger.pdf.

[9] Plimpton, S., home web page: www.cs.sandia.gov/~sjplimp/md.html, May 2003.

[10] Feldmann, M., Kent IV, D., Cummings, J., Muller, R., and Goddard III,W.,"Manager-Worker Based Model for the Parallelization of Quantum Monte Carlo on Heterogeneous and Homogeneous Networks", Internal Technical Report for Materials and Process Simulation Center, Beckman Institute (139-74), Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, California, 92215.

[11] Parashar, M., and Browne, J., "System Engineering for High Performance Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive Mesh Refinement," *IMA Volume 117: Structured Adaptive Mesh*

Refinement (SAMR) Grid Methods, Editors: S. B. Baden, N. P. Chrisochoides, D. B. Gannon, and M. L. Norman, Springer-Verlag, pp. 1 – 18, January 2000.

[12] Cummings, J., Aivazis, M., Samtaney, R., Mauch, S., and Meiron, D., "A Virtual Test Facility for the Simulation of Dynamic Response in Materials", LACSI Symposium 2001, Santa Fe, NM (October 2001).

[13] Cuitino A., Stainier L., Wang G, Strachan A., Cagin T., Goddard W.A., and Ortiz M., " A multiscale approach for modeling crystalline solids", Journal of Computer Aided Material Design, 2001.

[14] Cirak, F., Ortiz, M., and Schroder, P., "Subdivision Surfaces: A New Paradigm for Thin-Shell Finite-Element Analysis," Int'l. J. Numer. Methods Eng., Vol. 47, No. 12, 2000, pp. 2039-2072.

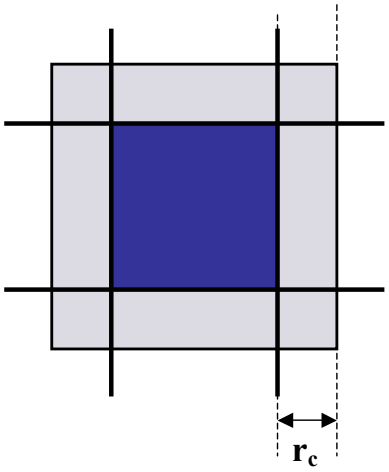

Figure 1. Typical Grid Cell and Cutoff Radius

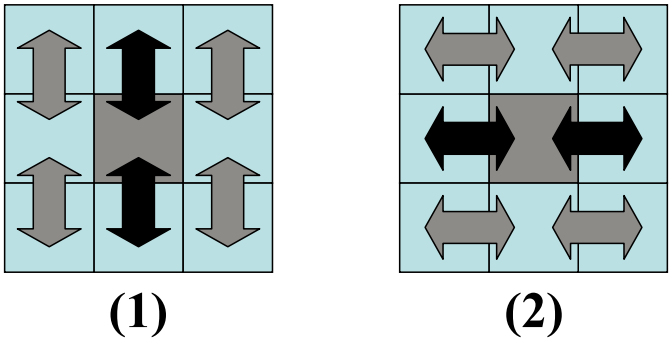

Figure 2. Communication Steps

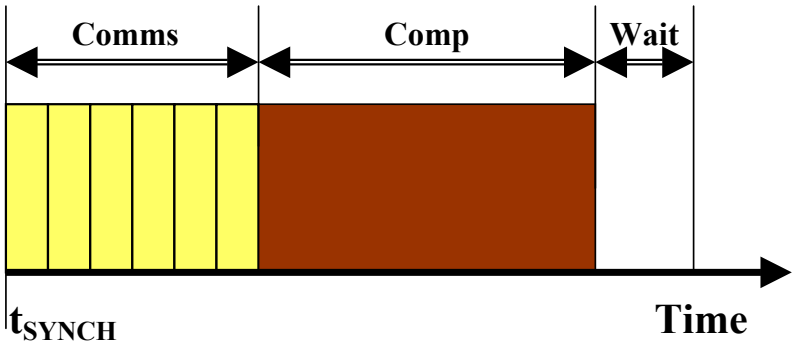

Figure 3. Computational Cycle Model

Name	***Value***	***Description***
Physics		
Dt	0.00442	Timestep size in reduced units
T_0	1.444	Initial temperature in reduced units
□	0.8442	Density in reduced units
r_c	2.5	Cutoff distance in reduced units
r_s	2.8	Extended cutoff distance in reduced units
Problem definition and execution control		
n_x, n_y, n_z	50, 50, 50 (per CPU)	Dimensions of domain bounding box (integer units)
alat	$(4/□)^{1/3} \cong 1.68$	Linear scaling factor
T	5	Number of simulation timesteps
n_{neigh}	20	Number of timesteps between re-binning
$nbin_x$, $nbin_y$, $nbin_z$	0.6 n_x, 0.6 n_y, 0.6 n_z	Number of cells per each dimension of the domain

Table 1. LJS Experiment Runtime Parameters

1 CPU, grid: 50x50x50	**Cycles**	**Instructions**	**FPU ops**
Setup	5052199616	4429953927	1770200815
Position computation	10177454	6250758	1500002
Communication	5773665	1296831	337623
Compute force	906525552	646414360	311087829
Reverse communication	3984631	1633667	337586
Velocity computation	24412276	8750806	1500003
Statistics and output	1582670466	1108981709	543052406

4 CPUs, grid: 100x100x50	**Cycles**	**Instructions**	**FPU ops**
Setup	5221111514	4511174346	1778322208
Position computation	14830811	6250758	1500004
Communication	10173647	3452399	247947
Compute force	919677779	642524200	309253334
Reverse communication	20714843	17943413	403433
Velocity computation	28379907	8750806	1500002
Statistics and output	1629528274	1127778373	539980682

8 CPUs, grid: 100x100x100	**Cycles**	**Instructions**	**FPU ops**
Setup	5219709948	4559748718	1778872028
Position computation	19109039	6250758	1500012
Communication	12344903	3682710	193669
Compute force	904651811	642808360	309405641
Reverse communication	69268341	71835364	383606
Velocity computation	33611634	8750806	1500004
Statistics and output	1611655969	1126834265	540171202

64 CPUs, grid: 200x200x200	**Cycles**	**Instructions**	**FPU ops**
Setup	8482763222	7700183441	1782447444
Position computation	18828594	6250744	1500010
Communication	21939538	14506940	197011
Compute force	905015895	642808346	309131305
Reverse communication	79774656	72790176	362519
Velocity computation	33838233	8750792	1500003
Statistics and output	1778038079	1283720429	540689376

Table 2. Operation Counts for Various Numbers of CPU's

Configuration	Comm. phase	Destination rank	Message size (bytes)
4 CPUs	Forward	2	496800
		2	372600
		1	1063152
		1	797352
	Reverse	1	797352
		1	1063152
		2	372600
		2	496800
8 CPUs	Forward	4	480000
		4	360000
		2	513600
		2	385200
		1	549552
		1	412152
	Reverse	1	412152
		1	549552
		2	385200
		2	513600
		4	360000
		4	480000
64 CPUs	Forward	48	480000
		16	360000
		12	513600
		4	385200
		3	549552
		1	412152
	Reverse	3	412152
		1	549552
		12	385200
		4	513600
		48	360000
		16	480000

Table 3. Communications Profile

64 CPUs, grid: 100x100x100	Cycles	Instructions	FPU ops
Setup	4518993233	4403610464	224582074
Compute positions	1130213	781994	187500
Communication	4574366	3009445	52475
Compute force	109366444	79949995	38478599
Reverse communication	7547645	6008098	92208
Compute velocities	3222944	1094542	187501
Statistics and output	206037876	148078372	67258754

Configuration	Comm. phase	Destination rank	Message size (bytes)	Ratio to msg. size for 200^3 grid
64 CPUs, 100x100x100 grid	Forward	48	120000	1:4
		16	90000	1:4
		12	136800	1:3.75
		4	102600	1:3.75
		3	155952	1:3.52
		1	116952	1:3.52
	Reverse	3	116952	1:3.52
		1	155952	1:3.52
		12	102600	1:3.75
		4	136800	1:3.75
		48	90000	1:4
		16	120000	1:4

Table 4. Tracing Results for $n_x = n_y = n_z = 100$ on 64 CPU's

Configuration	Destination rank	Message size (bytes)
64 CPUs, 20x20x20 grid	48	12000
	16	12000
	48	4800
	16	3600
	12	44400
	4	44400
	12	17760
	4	13320
	3	164280
	1	164280
	3	65712
	1	49272

Table 5. Tracing Results for $n_x = n_y = n_z = 20$ on 64 CPU's

Configuration	BG/L nodes	SGI 2000 Physical Processors	Total Injected Packets	Average hops/pkt
16x16x16	4K	8	400K	12
16x16x32	8K	16	800K	16
16x32x32	16K	32	1.6M	20
32x32x32	32K	64	3.2M	24
32x32x64	64K	128	6.4M	32

Table 6. Benchmark Problem Sizes

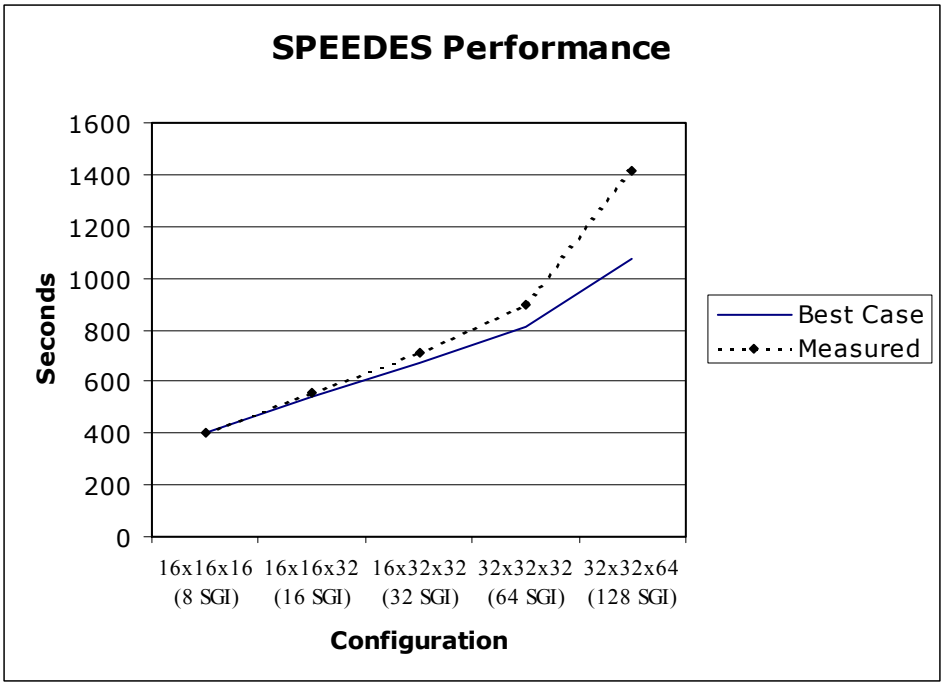

Figure 4. Scaling Results – Random Workload Case

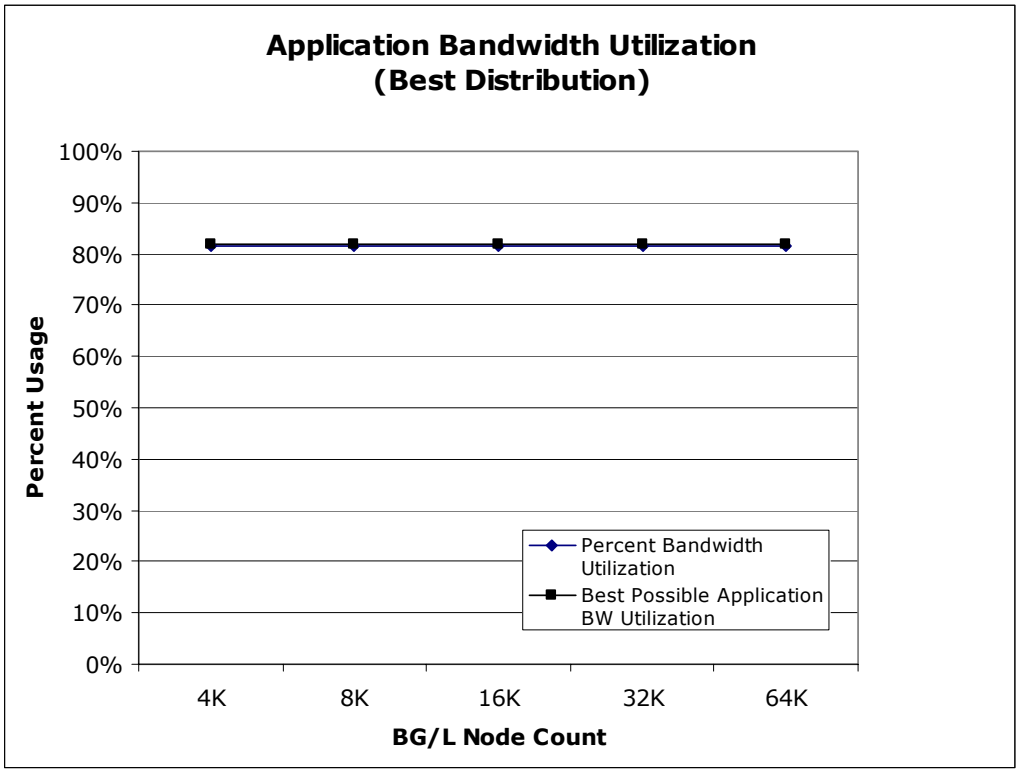

Figure 5. Simulation Results for Best (Physical Nearest Neighbor) Distribution

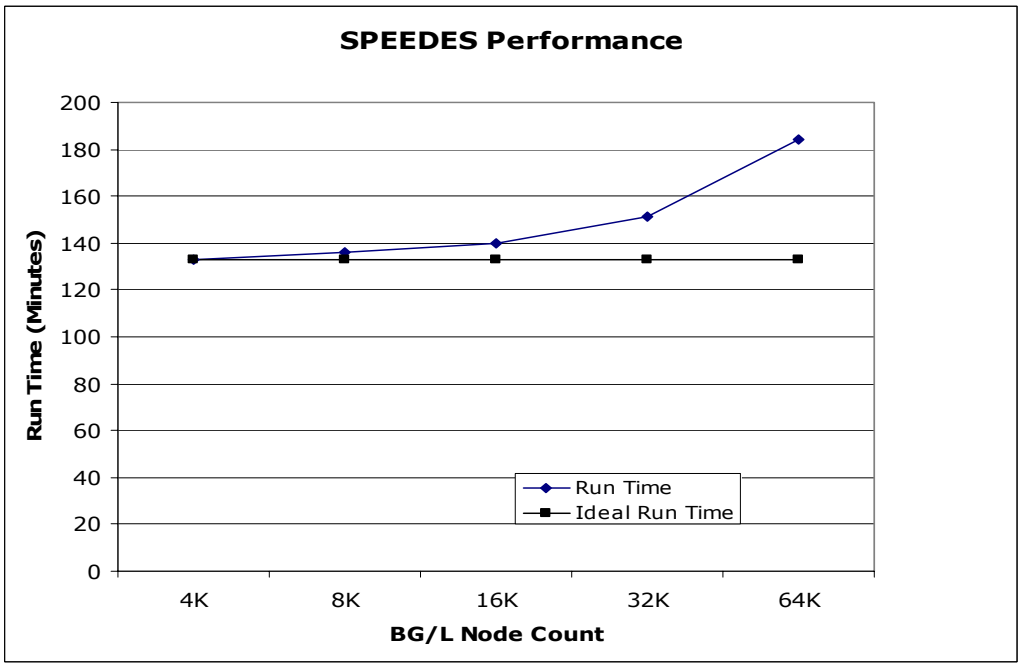

Figure 6. SPEEDES Scaling Performance

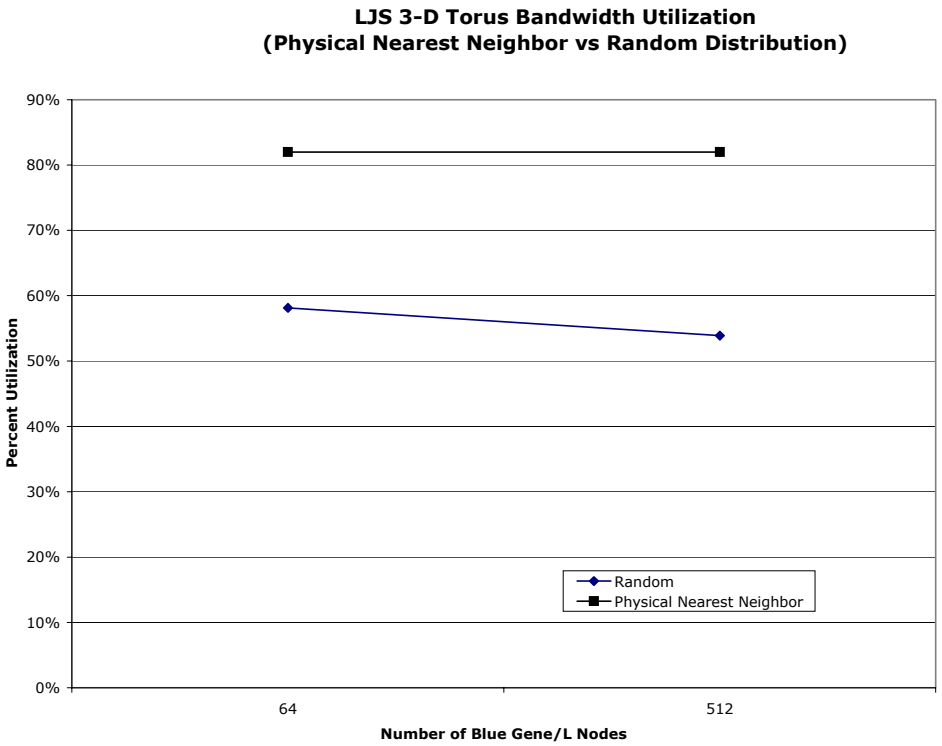

Figure 7. Best vs. Random Distribution of Cells

Scaled Size			
N (atoms)	BG/L Nodes	Compute Time (ms)	Communications Time (ms)
500,000	1	1480	3
256,000,000	512	1480	3
512,000,000	1024	1480	3
1,024,000,000	2048	1480	3
2,048,000,000	4096	1480	3
4,096,000,000	8192	1480	3
8,192,000,000	16384	1480	3
16,384,000,000	32768	1480	3
32,768,000,000	65536	1480	3
Fixed Size			
N (atoms)	BG/L Nodes	Compute Time (ms)	Communications Time (ms)
500,000	1	1480	3.00E+00
500,000	512	2.89E+00	1.88E+00
500,000	1024	1.45E+00	1.17E+00
500,000	2048	7.23E-01	7.32E-01
500,000	4096	3.61E-01	4.58E-01
500,000	8192	1.81E-01	2.86E-01
500,000	16384	9.03E-02	1.79E-01
500,000	32768	4.52E-02	1.12E-01
500,000	65536	2.26E-02	6.98E-02

Table 7. Fixed vs. Scaled Problem Size

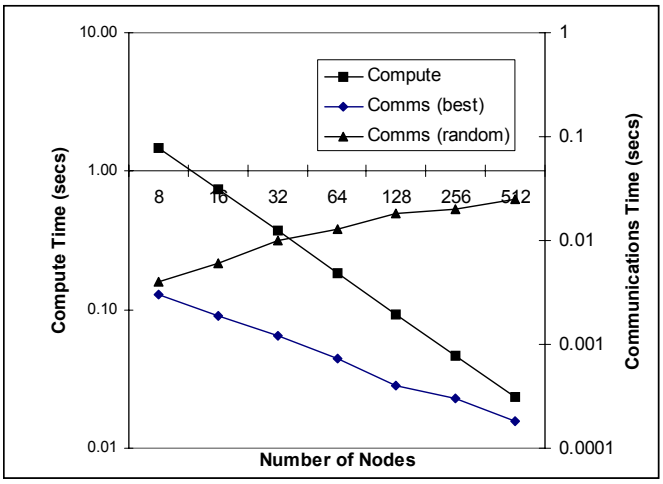

Figure 8. Fixed Problem Size Mapped to Increasing Number of BG/L Nodes

Processor	**Machine**	**CPU sec/atom/timestep**
333 MHZ Pentium*	Intel Tflops	1.67E-5
500 MHZ DEC Alpha EV6*	Sandia CPlant	7.03E-6
450 MHZ DEC Alpha EV5*	Cray T3E	1.7E-5
195 MHZ MIPS R10000*	SGI Origin	1.49E-5
700 MHZ BG/L Processor	**Blue Gene/L**	**2.96E-6**

Table 8. CPU secs/time step Comparisons (*from Plimpton [99])

Packet Delay (us)	**Flush Time (ms)**	**Escapes**	**Avg Transit Time (us)**	**Bandwidth Utilization**
0	24.9	10.20%	85.6	54.10%
3	24.4	9.70%	81.4	55.10%
6	25	8.90%	74.3	53.80%
9	24.2	3.60%	33.5	55.80%
10	21.5	0.50%	10.9	62.80%
11	23.6	0%	7.4	57.10%

Table 9. Throttling Effects for Random Distribution Case on 512 BG/L Nodes