# Analysis, Tracing, Characterization and Performance Modeling of Select ASCI Applications for 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

## 1. Introduction

Caltech's Jet Propulsion Laboratory (JPL) and Center for Advanced Computer Architecture (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.

The direct approach of cycle-by-cycle level simulation a 64K node system running parallel applications is infeasible due to limitations of existing computer systems. In lieu of this, the JPL/CACR 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 will be explicitly represented, but message traffic will be aggregated to reduce simulation run time while retaining statistical effects of adaptive routing and network contention as a function of network load and size.

Commercial sequential discrete event simulation packages are incapable of handling systems of 64K nodes. To cover the full range of BG/L sizes (up to 64K nodes) it was necessary to exploit parallel discrete event simulation software that could then be run on parallel machines for scalability.

Parallel discrete event simulation (PDES) has been successfully used for battlefield simulations which may involve large numbers of complex simulation objects (millions) where processing of individual events is compute intensive. Parallel simulations of MPP architectures and computer networks in which a large number simple objects and small compute time events are processed have not been as widely studied. A parallel discrete event statistical model was developed to scale to the full 64K nodes. Both optimistic and time step methods are being used to speed up the parallel simulation. An optimistic time step model was developed using the SPEEDES (Synchronous Parallel Environment for Emulation and Discrete-Event Simulation) framework developed at JPL[2]. Results of this approach are reported in this paper.

We expect 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.

## 2. Approach

Select ASCI and related applications were analyzed 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. In particular, the applications and kernels analyzed are:

The Lennard-Jones Spatial Decomposition (LJS) target code [3] 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. We examine the spatial decomposition case 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 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) of timesteps. 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.

The Quantum Monte Carlo (QMC) target code [4] calculates material properties within chemical accuracy. Quantum Monte Carlo methods are used by many scientific disciplines because the stochastic nature of the Monte Carlo method is easily parallelized and scalable. QMC calculations are typically broken into two computationally intensive phases: initialization and statistics gathering. Obtaining statistically independent data on each processor is typically done with each processor performing a time consuming initialization procedure. The parallel implementation of this QMC code uses a manager/worker paradigm. One node is the manager and n-1 nodes are workers. The manager sends MPI_isend messages directly to each worker to gather statistics. Subsequent statistics gathering is done with the MPI_Reduce command . Workers check the incoming buffer with a polling, MPI_Iprobe, command. The code is written in C++ and MPI.

The 3-D Adaptive Mesh Refinement (AMR3D) target code [5] is a parallel fluid dynamics code performing Richtmyer-Meshkov shock simulation using GrACE (Grid Adaptive Computational Engine) a data management library developed by Manish Parashar at Rutgers University for mesh generation and automatic load balancing. The computational domain of an application integrated with GrACE can be irregularly partitioned into a number of n-dimensional (with dimensionality determined by the nature of the problem) patches, also called "bricks" or "zones". Each of the patches can further spawn sets of more finely subdivided meshes (contained within the boundary of the parent patch) – with the maximal depth of such formed hierarchies of refinement limited explicitly by the user. AMR3D's behavior is highly dynamic with respect to CPU utilization and memory usage. Communication patterns include: point-to-point nearest-neighbor updates of boundary regions; global reductions to determine the time step value to be used throughout the next iteration; collective and point-to-point message bursts generated by grid re-composition events. AMR3D is written Fortran and GrACE is in C++.

The Magnetic Hydro Dynamics (MHD) target code [6] is a fluid solver for equations of hydrodynamics and resistive Maxwell's equations. The physical problem of interest is one of magnetic reconnection in two dimensions and the parallel implementation of the MHD code decomposes the physical domain into slices of rectangular subdomains with each subdomain assigned to an MPI process. Every subdomain has two extra cells along each Cartesian direction, which defines the area of overlap between physically neighboring subdomains. Data exchanges in the overlapping regions takes place at the start of each time step, using nearest neighbor non blocking send and receive communications. A global reduction, *MPI_allreduce*, of the minimum time step also happens at the beginning of each time step. MHD is written in Fortran 90 and MPI.

The multiscale polycyrstalline target code[7] computes the effective behavior of polycrystalline materials incorporating microstructural information. The simulation strategy promises to achieve a better understanding of the effective behavior of engineering materials in terms of the basic physical processes

occuring at the microstructural scale. Mesh partitioning allows each grain with its unique orientation to be assigned a different processor. Increasing the number of grains directly improves the accuracy of the mechanical stress field resolution.

The CONTACT [8] target code employs a new Lagrangian-Eulerian Shell-Fluid coupling algorithm for simulation of a compressible flow interacting with a thin-shell undergoing large deformations (for example an air bag). An Eulerian finite volume formulation is adopted for the fluid and a Lagrangian formulation based on subdivision finite elements is adopted for the shell response.

## 3. Analysis, Tracing, Experiments and Results

Due to space limitations, results from analysis, tracing and modeling are reported for LJS. Results for the full set of applications will be reported in the full paper.

### 3.1 Analysis

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 $\mathbf{\Delta T}$. 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. $\alpha$ is the cost of integrating the equation of motion for an individual particle over (small) time step.
4. $\beta$ is the cost of computing a single inter-particle force $F_{ij}$
5. $\delta$ is the cost of finding/enumerating particles in the neighborhood of current particle of interest.

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 is assigned to a processor, and a processor is responsible for performing the force calculations and state updates for all particles within the cell.
3. Force computation requires state information for some particles owned by other processors. 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. Reassignment of particles to processors according to the cell boundaries is done periodically.

The communications phase is a number of pairwise data exchanges between (logically) neighboring processors. In terms of diagram, the steps are as follows.

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.

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 other cases, we can approximate

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

For some scale factor $\lambda$,

$\lambda$= Fraction of local particles interesting across a single boundary.

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

The cost/time for the computational phase can be written as

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

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 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 Spatial Decomposition MD algorithm. The communications message size estimates the total byte count for each message in terms of one parameter ($\lambda$) and the Computation cost is a simple two-parameter representation.

## 3.2 Tracing LJS

### 3.2.1 Configuration

*The runtime parameters of the simulation along with their values are listed in the table below:*

| Name | Value | Description |
|---|---|---|
| *Physics* | | |
| *Dt* | 0.00442 | Timestep size in reduced units |
| $T_0$ | 1.444 | Initial temperature in reduced units |
| $\rho$ | 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* | $(4/\rho)^{1/3} \cong 1.68$ | 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 |

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 our 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. 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 our 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 timesteps 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), and we limited the tracing to the first five iterations.

### 3.2.2 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 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.

### 3.2.3 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*)

### 3.2.4 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. The tables below present the counter values collected for the setup, intermediate phases of the fourth timestep of the simulation (which is representative for other iterations as well) and finalization phase for different numbers of processors.

| **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 |

### 3.2.5 Communication Profile

The following tables characterize the communications between neighboring (logical) processing nodes for various numbers of nodes:

| Configuration | Comm. phase | Destination rank | Message size (bytes) |
|---|---|---|---|
| 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 |

### 3.2.6 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 are shown in the following tables:

| 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 |

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, and agree 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.

## 3.3 BG/L Parallel Simulation

The BG/L model was developed using SPEEDES, an optimistic parallel simulation framework developed in the early 1990's[2]. By default, SPEEDES uses a synchronization algorithm called *breathing time warp* based on the concept of virtual time developed by Jefferson[9]. 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.

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.

### 3.3.1 Model Development

Blue Gene/L uses a 3-D torus based network for point-to-point communications between nodes[10][1]. 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.

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 $m * [1 + 1/4 * (x + y + z)]$.

### 3.3.2 Experiments and Results

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.

Two cases were modeled. In the first case, ideal placement of cells is assumed, so that 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. We expected to make maximum use of the bandwidth, and to see good scaling, and we were not disappointed. 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 nodes, respectively. Because not all messages were exactly the same size, the upper limit of bandwidth utilization we could expect to see was 82%. For each model size, we saw 81% or better (see Figure 1).

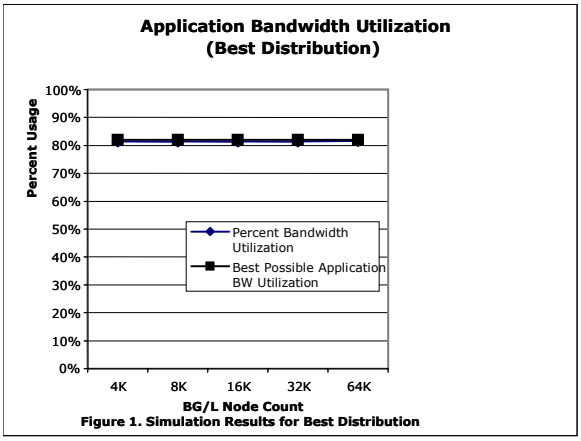

Figure 1. Simulation Results for Best Distribution

Scaling was fairly flat, except for the 64K node case running on 128 Origin nodes, which we again attribute to system overhead (see Figure 2). For the largest size, we simulated transmission of almost 700 million packets.

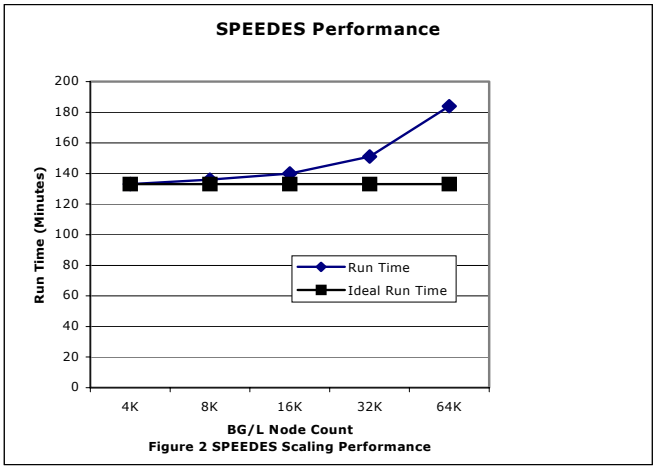

Figure 2 SPEEDES Scaling Performance

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 3).

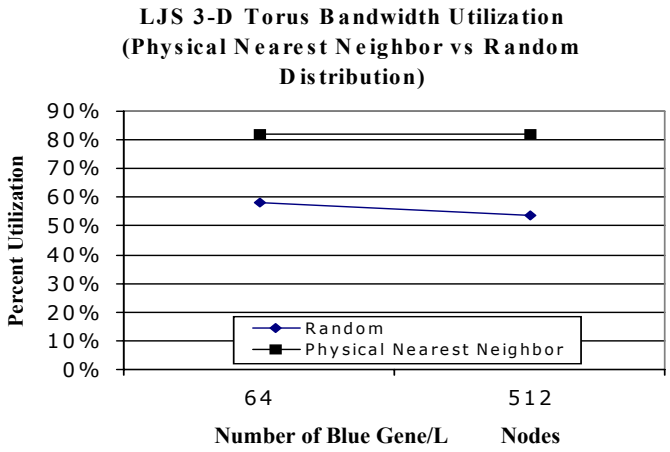


**Figure 3. Best vs Random Distribution of Cells**

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 1 and 2 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 $2.56 \times 10^8$ atoms.

Figure 4 shows the preliminary results for the case where the problem size was held constant at $3.2 \times 10^7$ 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.

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 3. Table 1 shows that slightly better performance can be obtained by "throttling" the injection of packets into the network.

| 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 1. Effects of Packet Throttling for Random Distribution Case on 512 BG/L Nodes**

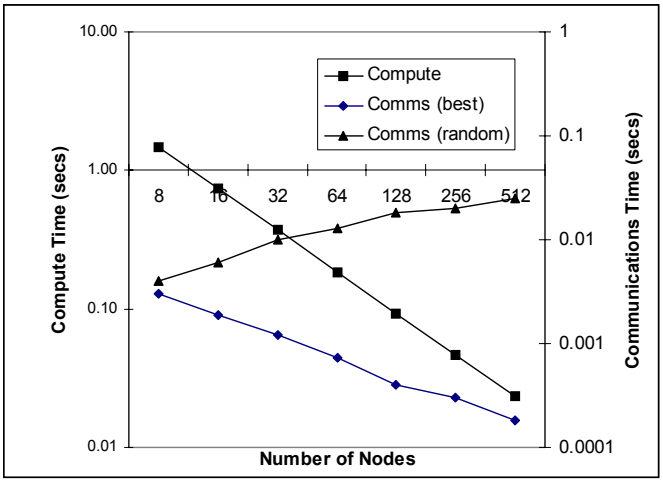


**Figure 4. Fixed Problem Size Mapped to Increasing Number of BG/L Nodes**

## 4. Discussion and Conclusions

Several observations can be made from the experiments:

- 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 can increase overall communications performance

## Acknowledgment

This work was done by the Center for Advanced Computer Research at the California Institute of Technology under contract with Lawrence Livermore National Laboratory, Contract No. B520721 (Applications Requirements Machine Model Simulator).The 128 node SGI Origin 2000 used in this investigation was provided by funding from 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] 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.

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

[4] Michael T. Feldmann, David R. Kent IV, Julian C. Cummings, Richard P. Muller, and William Goddard III, "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.

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

[6] Julian Cummings, Michael Aivazis, Ravi Samtaney, Sean Mauch and Dan Meiron, A Virtual Test Facility for the Simulation of Dynamic Response in Materials, LACSI Symposium 2001, Santa Fe, NM (October 2001).

[7] 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.

[8] 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.

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

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