Scalable line dynamics in ParaDiS

Vasily Bulatov, Wei Cai, Jeff Fier, Masato Hiratani, Gregg Hommes, Tim Pierce,
Meijie Tang, Moono Rhee, Kim Yates, Tom Arsenlis
Lawrence Livermore National Laboratory[1]

Abstract

We describe an innovative highly parallel application program, ParaDiS, which computes the plastic strength of materials by tracing the evolution of dislocation lines over time. We discuss the issues of scaling the code to tens of thousands of processors, and present early scaling results of the code run on a prototype of the BlueGene/L supercomputer being developed by IBM in partnership with the US DOE's ASC program.

Introduction to dislocation dynamics

The theory of crystal plasticity came into existence in the late nineteenth century [1]. With time it has become a proving ground for many novel methods of applied and computational mathematics. While very useful in practical applications in structural mechanics, geophysics, and other areas, mathematical crystal plasticity developed into a vestige of phenomenology, having little or, sometimes, no connection to the underlying microscopic physics of material behavior. Yet at the same time, metallurgists and physicists continued to ask hard questions: *why* do crystals deform in the ways they do and *what are the mechanisms* by which plasticity comes about?

Realization that the plastic strength of crystalline materials is controlled by the motion of its line defects, dislocations, has firmed up in stages. In 1934, the concept of dislocations as physical agents of crystal plasticity was proposed, simultaneously and independently, by three eminent scientists – Taylor, Polanyi and Orowan. While essentially explaining most of the puzzling phenomenology of crystal plasticity, dislocations still remained only a beautiful hypothesis until the late 1950's when first sightings of them were reported in electron microscopy experiments. Since then, the ubiquity and importance of dislocations for crystal plasticity and numerous other aspects of material behavior has been regarded as firmly established as, say, the role of DNA in promulgating life.

Having largely answered why crystals deform as they do, dislocations provide a natural basis for bringing physical content into the mathematical theory of crystal plasticity. However, blocking the road to a grand unification of dislocation physics and continuum crystal plasticity is a very serious hurdle: in order to be representative of macroscopic plasticity behavior, the motion and interaction of dislocations *en masse* must be considered. Given that the physics of dislocation motion and interactions are very well understood, the nature of this obstacle is purely computational: one needs to be able to

[1] This work was performed under the auspices of the U.S. Department of Energy by the University of California at Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48. UCRL-CONF-203896

0-7695-2153-3/04 $20.00 ©2004 IEEE

trace the simultaneous evolution of millions of dislocation lines over extended time intervals, in order to directly *compute*, as opposed to *fit*, the plastic strength of crystalline materials. This is one exemplary case where *the size does matter*. A mesoscopic approach of Dislocation Dynamics attempts to address this challenge.

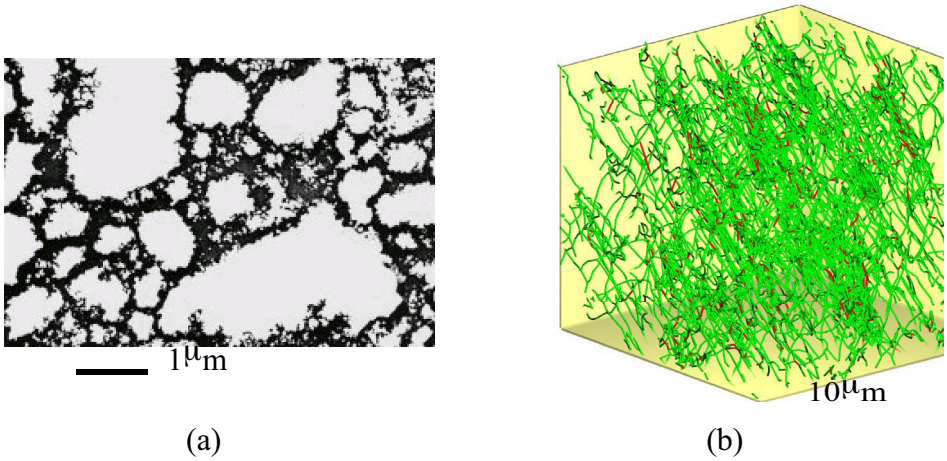


Figure 1. (a) Dislocation microstructure developed in copper during plastic deformation, as observed in an electron microscope [2]. Dark regions contain large numbers of dislocation lines, whereas light regions contain no dislocations. (b) Dislocation microstructure developed during plastic deformation in a dislocation dynamics simulation by ParaDiS.

Dislocation Dynamics: unifying dislocation physics and crystal plasticity

The idea behind the computational approach of Dislocation Dynamics (DD) is simple. One introduces dislocation lines in the computational volume, letting them interact and move in response to forces inflicted by external loads and inter-dislocation interactions. The accuracy of DD simulations is assured by dislocation physics and mechanics that supply methods for computing forces acting on dislocations and dislocation velocities under these forces. Although the DD approach was introduced relatively recently, in the early 1990s, its potential impact on the science of material strength is well recognized [3-5]. Yet, despite multiple pronouncements of inevitable success, the DD approach has not come close to the levels of computational performance required to demonstrate that it can, indeed, predict macroscopic strength from the collective behavior of dislocations. Given the computational complexity of the problem at hand, the limited success DD can claim so far is not surprising. At the same time, the performance levels required for "break-through" DD simulations can be very well quantified.

Among various challenges worthy of large-scale computational attack, understanding the nature of strain hardening and dislocation patterning in metals is one of the most famous and holds a special value among researchers in the area of material strength. If direct DD simulations can be shown to accurately reproduce dynamic hardening transitions that

occur naturally during crystal deformation[2], even skeptics will be convinced that the microscopic physical theory of crystal strength has arrived in the form of DD. With this in mind, we decided to gear our new DD code towards a single large-scale "hero" simulation that will cover the length and time scales sufficient to observe the hardening transitions that occur naturally as a result of collective motion and rearrangement of dislocations. Careful estimates show that to be able to naturally account for hardening and dislocation patterning and avoid "small volume" artifacts, the model should include from 1M to 100M dislocation segments. Furthermore, the evolution of such large dislocation groups will have to be traced over millions of time steps to reach the strain levels at which the hardening transitions are observed. DD capabilities available until now at LLNL and elsewhere stop short of these target performance figures by some 2-3 orders of magnitude. Massively parallel computing, such as will be available with BlueGene/L in 2005, is the only viable pathway to closing this performance gap.

There are several challenges to this program that can be formulated at the outset. The first one is a programming challenge: DD models deal with the ever-changing topology of dislocation lines and line networks requiring rather complex data structures. On parallel machines, this challenge is dramatically amplified due to the need to handle the topology of these line networks across the boundaries of computational domains. The second challenge is a natural tendency of dislocation lines to cluster in space (owing to the long-range interactions among the lines) and develop highly heterogeneous distributions of degrees of freedom making it difficult to achieve a good load balance. The third challenge is to handle multiple time scales observed in the evolution of large dislocation ensembles, where periods of relative calm (slow motion of lines) are interspersed with bursts of very high activity over which small groups of lines move very fast and experience multiple collisions. The code ParaDiS recently written at LLNL has shown much promise in addressing these challenges.

We have developed an interesting strategy for planning and executing simulations. In the course of deformation, dislocations multiply, increasing their numbers by 2-3 orders of magnitude. For this reason, it is possible to start with a relatively small model and let it grow on a small machine. Then, following a steadily growing number of dislocations, the job should be moved to progressively larger machines. This sort of progression worked very well when we scaled our simulation up from 12 to 100, then to 200 and, eventually, to 1,500 cpus. The same strategy will be applied on the Blue Gene/L machine, which is planned to have 131,072 processors when delivered to LLNL in early 2005. Scaling results for up to 4096 processors are shown in a later section of the paper.

The ParaDiS project

ParaDiS stands for *Parallel Dislocation Simulator*. It is a massively parallel dislocation dynamics simulation code that is being developed at the Lawrence Livermore National Laboratory since 2001. ParaDiS is specifically designed for investigating the collective behavior of large numbers of dislocation lines as required for understanding and accurate

[2] Strain hardening is responsible for some well-known facts of everyday life, such as why an aluminum paper clip eventually breaks after bending it back and forth several times.

prediction of plasticity and strength in crystals. The code is mainly written in C and uses the MPI library for communication among the processors.

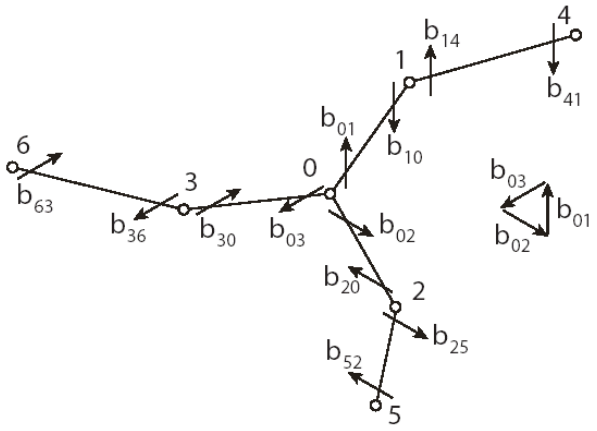


Figure 2. A fragment of the dislocation lines network: each line segment *xy* carries a unit of "vector current" quantified by Burgers vector b_{xy}.

The object of a ParaDiS simulation is a network of nodes connected by line segments. A node can have two or more segments, or arms, connecting it to neighboring nodes. Each segment carries a unit of "vector current" or Burgers vector, which denotes the direction and magnitude of the displacement that occurs when a dislocation moves. Similar to the scalar currents in electric circuits, Burgers vectors are conserved along the lines and satisfy the Kirchoff rule: the sum of Burgers vectors of all arms emanating from a given node is zero, e.g. $\mathbf{b}_{01} + \mathbf{b}_{02} + \mathbf{b}_{03} = 0$ (see Fig. 2). Also similar to electric current, the network can never terminate on a node with a single arm that carries a non-zero Burgers vector. During the simulation the network topology evolves as new nodes are added and some old nodes get deleted. The nodes move in space in response to the forces they see.

Nodal forces and motion

The nodes move in response to nodal forces according the first order equation of (over-damped) motion:

$$\frac{d\vec{r}_i}{dt} = M[\vec{f}_i]$$

$$\vec{f}_i = \frac{E[\{\vec{r}_i\}]}{\vec{r}_i}$$

where f_i is the force on node *i, E* is the energy of the dislocation network [1], and $M[f_i]$ is a mobility function giving the velocity of node *i* as a function of nodal force f_i. The network energy *E* includes the interaction between all network segments and between the segments and applied stress. The segment-to-segment interaction forces are long range, taking the lion's share, typically more than 80%, of the total CPU time to compute.

Topological rearrangements

In addition to moving the nodes, ParaDiS evolves the network topology to reflect the physics of dislocation motion and collisions in real crystals. Handling the evolving topology of moving and intersecting lines is a major bookkeeping challenge, especially in a parallel implementation. It is therefore highly desirable to reduce the logical complexity of the topological switches. Presently, ParaDiS relies on two basic operations: 1) insert a new node and 2) merge two nodes into one. Whenever two nodes become doubly connected (Fig. 3(c)), the double arm is replaced by a single arm with the Burgers vector equal to the sum of the Burgers vectors of its two parents. Yet even this simple algorithm requires considerable care since the nodes participating in a single topological rearrangement may belong to different processors. Even though topological changes consume only a small fraction of the computing time, the associated logic and bookkeeping constitute about 50% of the ParaDiS source code, and are fully implemented.

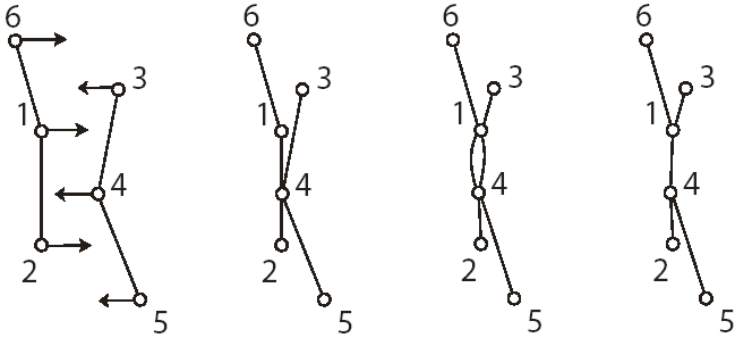


Figure 3. Handling topological changes caused by dislocation collisions: (a) just before the collision; (b) at the collision time a new node is inserted in segment 1-2 and merged with node 4; (c) a similar insert-and-merge sequence involving node 1 and segment 3-4 (d) the double arm connecting nodes 1 and 4 is reduced to a single arm.

Spatial domain decomposition and dynamic load balancing

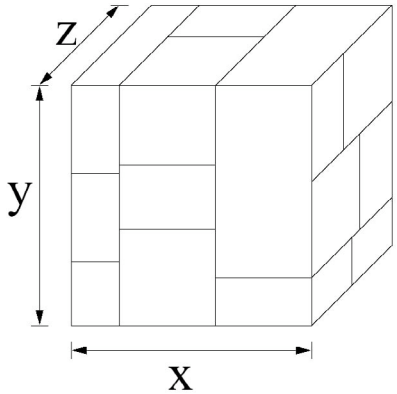


Figure 4. Example decomposition of 3-D simulation space into 3 3 2 domains.

ParaDiS relies on spatial decomposition (Fig. 4) to partition and distribute the nodal data and computational load over the processors. This is essential for good scalability, since most communications take place between neighboring processors. A common approach

to data decomposition is to partition the entire simulation space into equally-sized rectangular domains and to assign each domain to a processor. Unfortunately, this approach leads to poor scalability due to a tendency of dislocation line dynamics simulations to accumulate load imbalance. This is because in the course of simulations, the lines tend to organize themselves spontaneously into spatially heterogeneous (lumpy) arrangements. Associated large variations of the local line density cause computational loads to vary significantly from one processor to another. To maintain scalability ParaDiS uses hierarchical recursive bisection to partition the problem domain first in the X, then in the Y, and finally in the Z dimension [6,7]. The X-partition assigns equal computational load (weight) to each YZ slab. Each slab is then independently partitioned in the Y direction into blocks of equal weight, and similarly for the Z direction. At regular time intervals ParaDiS re-evaluates the computational load and shifts the domain boundaries to maintain good balance, as shown in a later section of the paper (Tab. 1).

Algorithm cycle and communication patterns

Because dislocation interaction is long range, any two line segments interact with each other in a ParaDiS simulation. For computational efficiency, all segment-segment interactions are partitioned into *local* and *remote* contributions, based on proximity of the interacting segments. For this purpose, a static mesh of cubic *cells* is overlaid on the problem space. The interactions between two segments in the same or in the neighboring cells are considered and treated as local. Otherwise, they are considered remote. 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.

The reason for overlaying a regular grid of cubic cells on the problem space is to enable efficient calculation of forces produced by remote segments, using a fast multi-pole algorithm. The choice of the cell size or, conversely, the number of cells in the total volume, is a compromise between the time required to compute short-ranged and long-ranged forces. Forces on a given dislocation segment are computed explicitly only from the segments in its own and in the neighboring cells (27 cells in total). The rest of the forces are lumped into super-segments, and are much cheaper to compute. By some experimentation we find that, depending on the problem size, we can select an optimal sub-division into cells such that the time spent on explicit (short-range) forces is approximately equal to the time spent on the remote interactions. This balance obviously depends on the number of cells chosen and, very likely, on several other issues. For example, since long-range forces require some global communication, that will affect the optimal choice of cell subdivision.

During a given algorithm cycle, local and remote forces on each network node are computed, node positions are advanced, and topological rearrangements are performed, if necessary. A single ParaDiS cycle consists of the following substeps[3].

[3] In the following, a *local* communication is between the neighboring processors (domains) while in a *global* communication, all processors talk to each other. ParaDiS tries to minimize the amount of global communication as much as possible.

1) Each node is assigned to a cell according to its spatial location.
2) Each domain sends/receives the state (including connectivity) of each node assigned to the cells neighboring the domain's own cells. The nodes received from the neighboring domains are called *ghost nodes*.
3) To account for the remote forces, all segments in a given cell are lumped into a single net super-segment and its net charge is communicated via a *global* reduction (communication) operation to all domains.
4) Total force on each node is computed as a sum of *local* and *remote* interactions.
5) Nodal velocity is calculated, based on the nodal force and the mobility function. The velocities are propagated to ghost nodes via *local* communication.
6) Nodal positions are advanced. Line collisions are detected and the topology adjusted accordingly. Any topological changes are propagated via *local* communication.
7) The lines are remeshed: nodes added and removed as line geometry requires. Changes are propagated via *local* communication.
8) Occasionally, load balance is re-evaluated based on the times it takes each processor to compute the nodal forces. The results are used to adjust domain boundaries for improved load balance. A small amount of *global* communication is required.
9) Nodes that migrated across domain boundaries due to nodal motion and/or domain boundary adjustments are transferred to new domains, via *local* communication.
10) I/O for this cycle is performed, potentially requiring some *global* communication.

During a single cycle, local communications take place several times. At any particular time, a given domain overlaps a set of cells, its *native* cells. Since the cell mesh is regular, each cell has 26 neighboring cells. Any other domain that overlaps either a native cell or a neighbor of a native cell of the given domain, is considered to be a neighbor domain. In most of the local communications (the ghost node communication of step 2 is the exception) only a small amount of data is actually transferred. This communication pattern of ParaDiS fits well with the architecture of BGL, with its 3-D torus network for the neighboring processors and a separate tree structure for global communications. Note, however, that due to the somewhat irregular spatial decomposition of ParaDiS domains, nearest neighbor domains needn't always correspond to nearest neighbor processors in BGL's torus network, though on average they will tend to be nearby.

Typical output of ParaDiS

Like any other simulation program, ParaDiS has access to all of its degrees of freedom at every time step. This information is written out periodically and is used to visualize the evolution of the dislocation network, as in Fig. 1(b). In a typical ParaDiS simulation, the initial distribution of dislocations is deformed at a constant strain rate (e.g. 1 s^{-1}). Two important outputs from this type of simulation are stress, Fig. 5(a), and total dislocation density, Fig. 5(b), as functions of strain.

During a simulation, the dislocation line density and, hence, the number of nodes increase by 1-2 orders of magnitude. This multiplication behavior is typical of crystals under stress. An important computational implication is that the size of the problem (the number of network nodes) steadily increases as the simulation proceeds. As is very well known to every dislocation simulator, even a small model of the dislocation network that accumulates strain briskly in the beginning of the simulation, quickly outgrows the computer it ran on bringing the simulation nearly to a standstill. As an example, the results shown in Fig. 5 were obtained in a simulation that initially ran on a 12-cpu Linux cluster but was later moved to a 200-cpu Linux cluster and, finally, to 512 cpus of an IBM-SP3 machine. We anticipate that, by the time the BGL machine grows to its full size of 64K nodes, the line dynamics simulation we report on here will also grow to a size that demands a machine of that size.

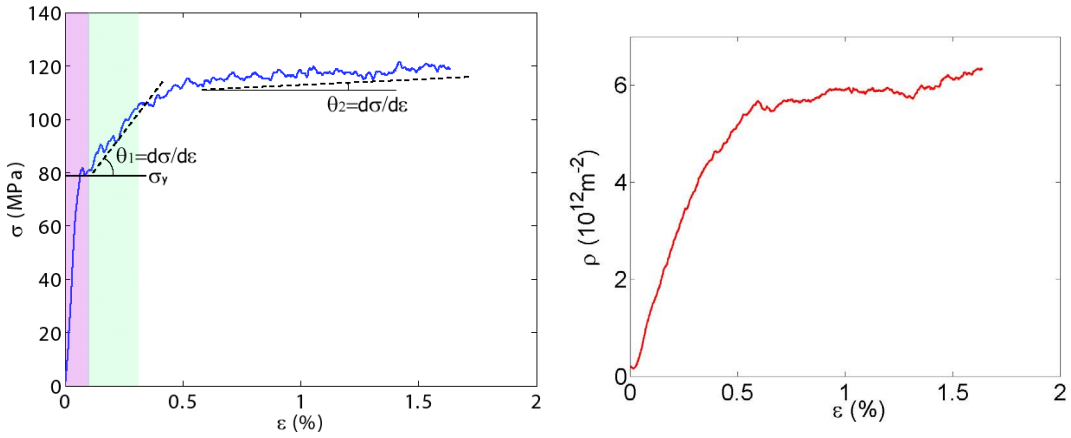


Figure 5. (a) Mechanical strength of single crystal molybdenum computed in a single ParaDiS simulation. The red region is a typical amount of strain a serial dislocation code can accumulate. The blue region shows how much strain ParaDiS was able to generate a year ago. Currently, ParaDiS can generate more than 3% of strain in a single simulation. Variations in the slope of the stress-strain curve reflect dynamic transitions in the collective behavior of dislocation lines. (b) The same transitions are also seen in the behavior of the total density of dislocation lines as a function of strain.

The BlueGene/L ASC platform

BlueGene/L is a massively-parallel computing system designed for research and development in computational sciences. It is targeted to deliver hundreds of teraflops to selected applications of interest to the US Department of Energy/National Nuclear Security Agency's Advanced Simulation and Computing program (ASC). It is an extremely high compute-density system with very attractive cost-performance and relatively modest power and cooling requirements. BGL is being developed by IBM in partnership with ASC, and is planned to be delivered to Lawrence Livermore National Laboratory in early 2005. At this writing there exist a 4096-node 500 MHz prototype using pass 1 chips, and a 1024-node 700 MHz system with pass 2 chips.

BlueGene/L has been extensively described elsewhere [8,9]. The 65,536 compute nodes

of BGL are each composed of just a single moderately-clocked chip, together with their attendant main memory chips, drastically lowering power consumption and space requirements while favoring communication and memory performance. The BGL node is a chip comprising two independent processors, each capable of two floating point operations per cycle (including fused multiply-adds, yielding a theoretical peak of 4 floating point operations per cycle), several independent network controllers, three levels of cache (including 4 MiB L3), and memory controllers. Though each floating point unit is capable of two operations per cycle, they are not independent: the second floating point pipe is usable only by 2-way SIMD instructions, or by 2-way "SIMOMD" (i.e., "single instruction, multiple operation, multiple data") instructions.

The two processors on each chip are identical, with symmetric access to resources. It is expected that most applications will use *communication coprocessor mode*, running a single MPI task per node, with one processor running the application, but with the MPI library offloading much of the work of message passing to the second processor. Some applications may run in *virtual node mode*, running two MPI tasks on each node (one on each processor), or in *dual-core mode*, using a fork-join mechanism to perform computation on the second processor (in which case the user must deal explicitly with the lack of coherence in the L1 caches).

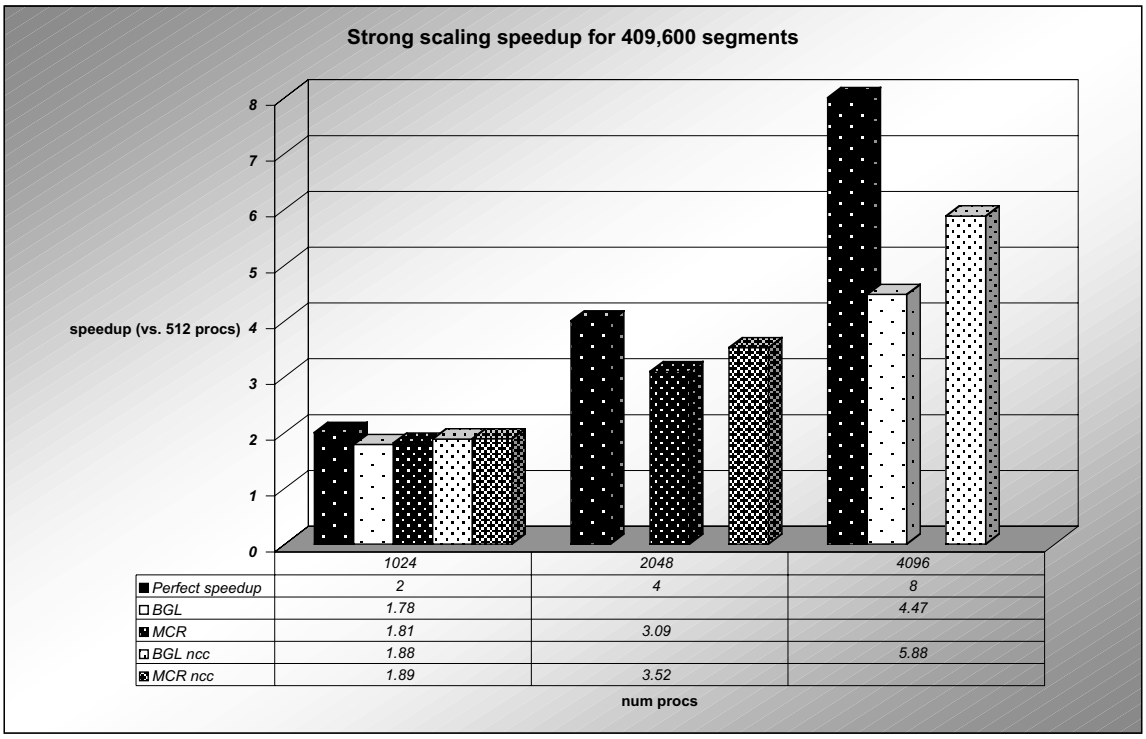
MPI communications are handled by three independent special-purpose networks in BGL. Point-to-point and all-to-all communications are handled by a 32x32x64-node three-dimensional torus, with each node connected to its nearest neighbors via six independent bidirectional links. In addition to the torus, BGL also has tree networks to perform global operations like broadcasts, reductions, and barriers with very low latency and high bandwidth, e.g., the entire 65,536-node machine is targeted to complete an MPI_AllReduce in less than 10 microsec and an MPI_Barrier in 5 microsec.

Performance results

In the results presented below, we have used strong scaling (i.e., the initial problem size is the same for all runs) to investigate ParaDiS performance on 512 to 4096 processors. In addition to running on the 4096-node BGL prototype, we also ran on LLNL's MCR, an 1152-node Linux cluster, in which each node has two Xeon processors running at 2.4 GHz, each capable of two floating point operations per cycle. On MCR we ran two tasks on each dual-processor shared-memory node, one for each cpu. All BGL results reported here were run on the 500 MHz prototype; since the speed of all components of BGL, including memory, scales with processor clock speed, we have adjusted the results to reflect a 700 MHz clock, which is the target frequency of the to-be-delivered BGL. We ran in communications coprocessor mode on BGL, running just one task on each node. At the time of this writing, the IBM xl compiler for BGL does not yet generate instructions for the second pipe of the "double hummer" floating point unit, as it is not a standard component of the IBM 440 core processor. Later versions of the compiler will generate 2-way SIMD instructions to utilize the second pipe, which should significantly increase floating point performance.

Figure 6 shows that the scaling performance of ParaDiS is excellent on both MCR and BGL. This is strong scaling (i.e., the most difficult kind), with the initial problem size being held constant from run to run. We expect even better scaling on BGL eventually, with larger problems sizes (see Fig. 7) and with the maturation of the MPI implementation. (At the time of these tests BGL's MPI was still under development and lacked some planned improvements that will optimize the use of its torus and tree networks.) We have not yet had time to experiment with different mappings of MPI tasks to BGL's torus, which can affect the performance of point-to-point communications.

The performance shown is based on the elapsed time of the final 25 steps of 150-step runs, because the earliest steps of a run include an amount of load balancing activity by ParaDiS that is not typical of later steps (see load balancing in Tab. 1). Two scaling numbers are given in Fig. 6 for each machine and processor count: "BGL" and "MCR" is the total time for the 25 steps, and "BGL ncc" and "MCR ncc" exclude time for the "cell charge" routine, which is dominated by file I/O. The latter is a fairer comparison, because the file system attached to the prototype BGL system is a slow serial system, whereas in production use BGL will be attached to a large, fast parallel file system. Due to time constraints, we were unable to run experiments on 2048 BGL nodes, and of course MCR is limited to only about 2048 processors.



	1024	2048	4096
Perfect speedup	2	4	8
BGL	1.78		4.47
MCR	1.81	3.09	
BGL ncc	1.88		5.88
MCR ncc	1.89	3.52	

Figure 6. Strong scaling from 512 processors to 2048 on MCR and 4096 on BGL, running a problem with 409,600 initial segments.

The elapsed times for the last 25 steps of the 512-processor runs on BGL and MCR were respectively 635 and 305 seconds (a 2.08:1 ratio), or 594 and 288 seconds (a 2.06:1 ratio) when we exclude the time for cell charge (mostly file I/O). In both cases, the ratio is

considerably less than the 3.4:1 ratio of the BGL and MCR processor clock cycle times (at 700 MHz and 2.4 GHz). We surmise that the better per-clock performance on BGL is due mainly to its more balanced memory system, but have not yet investigated that in detail. Moreover, recall that BGL's prototype compiler does not yet generate instructions for the processor's second floating point pipe; it will be interesting to see how performance improves with future compilers.

Figure 7 shows the effect of problem size on strong scaling on MCR. As expected, scaling improves with larger problems, presumably due to a more favorable ratio of computation to communication. We have not yet run these experiments on BGL, but expect a similar improvement in scalability with larger problems. (Note: the scaling shown in Fig. 7 is based on total run time, rather than the run time of the final 25 steps, and so is affected by the greater imbalance of the early steps in each run.)

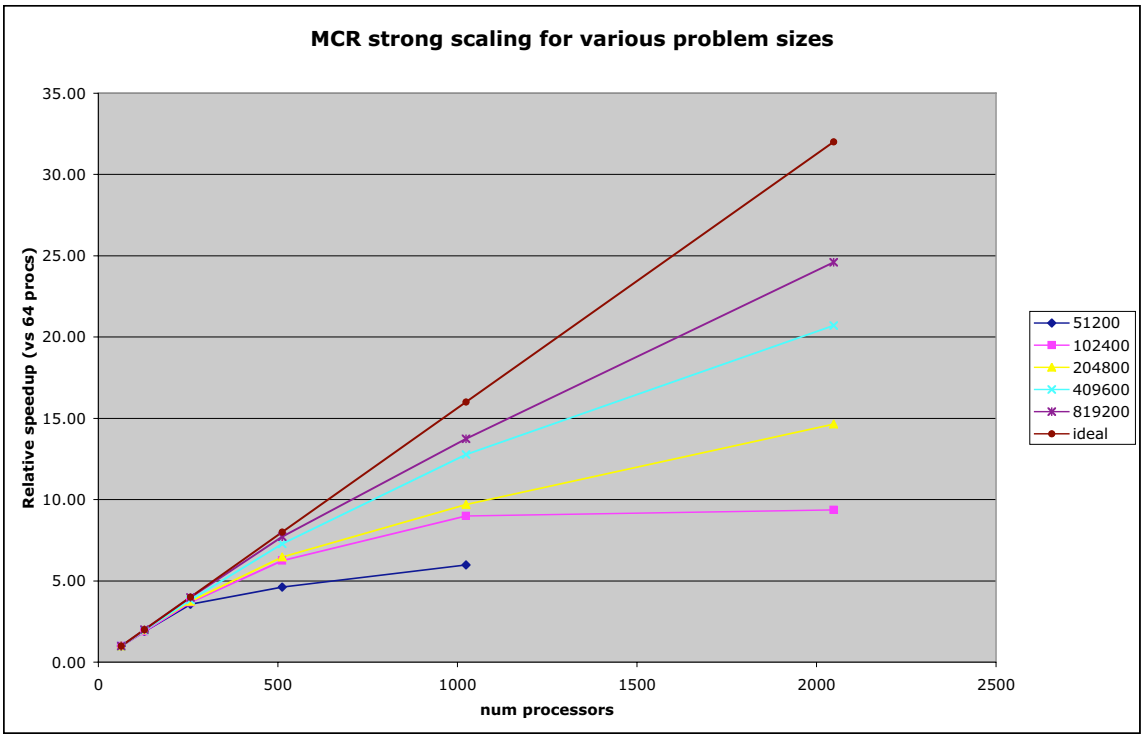


Figure 7. Strong scaling on MCR using different problem sizes (number of initial segments).

Table 1 shows the overall load balance achieved by ParaDiS on the studied problem over 25-step intervals. The load balance value is the time spent in force calculations averaged over all the cpus, divided by the maximum force calculation time for any cpu. During the first 75 steps, load becomes increasingly balanced, then stays fairly balanced.

Step	25	50	75	100	125	150
BGL	82%	89%	92%	93%	94%	95%
MCR	83%	90%	93%	95%	96%	96%

Table 1. Load balance at 25-step intervals on 1024 processors.

It will take more experimentation to discover how well ParaDiS will scale to tens of thousands of processors, but the results up to 4096 are encouraging. Over the coming year we will investigate the performance of ParaDiS on larger and larger BGL systems, also experimenting with virtual node mode (running two MPI tasks per node), different mappings to the torus, and new optimizations in BGL's MPI library and compilers.

Summary

A new code for line dynamics simulations, ParaDiS, is being developed at LLNL. The code shows considerable promise to deliver the long-awaited breakthrough in computational materials sciences: a first direct calculation of material strength based on the underlying behavior of dislocation lines. As more capable machines become available, so grows the modeled dislocation network, promising to reach the target level of complexity through efficient utilization of the new computational resources.

References

[1] J. P. Hirth and J. Lothe, *Theory of Dislocations*, 2nd ed., Wiley, New York, 1982.
[2] H. Mughrabi, T. Ungar, W. Kienle and M. Wilkens, "Long-range internal stresses and asymmetric X-ray line-broadening in tensile deformed [001]-oriented copper single crystals", *Phil. Mag. A* 53, 793 (1986).
[3] 1. K. W. Schwarz, "Simulation of dislocations on the mesoscopic scale. I. Methods and examples", *J. Appl. Phys.* 85, 108(1999).
[4] B. Devincre and L. P. Kubin, "Mesoscopic simulations of dislocations and plasticity", *Mater. Sci. Eng. A* 8, 234-236 (1997).
[5] N. M. Ghoniem and L. Z. Sun, "Fast-sum method for the elastic field of three-dimensional dislocation ensembles", *Phys. Rev. B* 60, 128 (1999).
[6] M.J. Berger and S.H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors", IEEE Trans. Computers V36(5):570-580 (1987).
[7] P. M. Campbell, E.A. Carmona and D.W. Walker, "Hierarchical domain decomposition with unitary load balancing for electromagnetic particle-in-cell codes," in *Proc. Fifth Distributed Memory Computing Conference*, Charleston, South Carolina, April, 9-12, 1990. IEEE Computer Society Press, 1990.
[8] N.R. Adiga et al. "An Overview of the BlueGene/L Supercomputer," in *Proc. of the ACM/IEEE SC2003 Conf.*, Nov. 2003.
[9] http://www.llnl.gov/asci/platforms/bluegenel/