

Massively-Parallel Dislocation Dynamics Simulations

Wei Cai, Vasily V. Bulatov, Tim G. Pierce, Masato Hiratani, Moono Rhee, Maria Bartelt and Meijie Tang

*Lawrence Livermore National Laboratory, University of California,
Livermore, CA 94551*

Abstract. Prediction of the plastic strength of single crystals based on the collective dynamics of dislocations has been a challenge for computational materials science for a number of years. The difficulty lies in the inability of the existing dislocation dynamics (DD) codes to handle a sufficiently large number of dislocation lines, in order to be statistically representative and to reproduce experimentally observed microstructures. A new massively-parallel DD code is developed that is capable of modeling million-dislocation systems by employing thousands of processors. We discuss the general aspects of this code that make such large scale simulations possible, as well as a few initial simulation results.

Keywords: parallel computation, dislocation dynamics, plasticity

1. Introduction

It has been known for a long time that crystal plasticity is produced by the motion of many dislocation lines [1]. Consequently, *a priori* predictions of the strength of a single crystal against plastic deformation must be possible, at least in principle, by modeling the dynamics of dislocation lines under the influence of external stress and mutual interactions. Such has been a dream of the computational materials scientists for several decades. Yet, it remains a grand challenge even to date. The major difficulty lies in the fact that, to have a representative model of crystal plasticity, the dynamics of a *large enough* number of dislocations needs to be followed for a *long enough* time interval. The length and time scales required have remained beyond the reach of the existing simulation codes.

To understand why, let us consider a typical dislocation microstructure spontaneously developed in copper during plastic deformation [2]. The structure exhibits patterns over the length scale of microns. To model this behavior, a simulation box of about $L = 10\mu\text{m}$ would be necessary. Given that the experimental estimates of dislocation density in such conditions are around $\rho = 10^{12}\text{m}^{-2}$, the total length of dislocation lines in the simulation box is about $\Lambda = \rho L^3 = 10^{-3}\text{m}$. In a dislocation dynamics (DD) simulation, dislocations are discretized into segments. If the average segment length is $d = 1\text{nm}$, then the total number of segments in this simulation would be $N = \Lambda/d = 10^6$, i.e., simultaneous

treatment of a million segments is required. This is a rough estimate but serves to identify the order of magnitude of the complexity of simulation required to address the crystal plasticity problem. Here in this paper we refer to a simulation with million dislocation segments as our target problem.

Because the interactions between dislocation segments are complex and long-ranged, dislocation dynamics codes, when running on a single processor, can only handle up to 10^4 segments. Beyond that size the simulation becomes very slow and no longer useful. Notice that this is two orders of magnitude away from the target size stated above. Yet another computational limit exists: when a reasonable initial dislocation density is used, the total plastic strain one can accumulate using the sequential DD codes is on the order of 0.1%, another two orders of magnitudes below the levels of strain where dislocation patterning and strain hardening behaviors are observed (typically at around 10% plastic strain).

To extend our simulation capability in both length and time scales by two orders of magnitude and to meet the requirements for faithful modeling of crystal plasticity, massively parallel computing appears to be a natural solution. For example, imagine a simulation where 10,000 processors are used simultaneously, each handling on average only 100 dislocation segments. Because the load on each processors is relatively light, a million dislocation segments can be simulated at a reasonable speed in order to accumulate large enough plastic strain. However, developing a DD code that is scalable up to $10^3 - 10^4$ processors is a highly nontrivial task. In this paper, we describe a few general features of our new massively-parallel DD code and present a few initial results from runs on up to 200 processors.

2. Simulation Methodology

The development of the DD3d code began at the Lawrence Livermore National Lab (LLNL) in 2001. To date (two years later) the first version is completed while further developments are still on-going, mainly focused on further enhancing the simulation efficiency and the accuracy of the physical models. The main objective for the DD3d code is to be able to take advantage of massively-parallel computers effectively. To achieve this goal, there are two basic design principles to which we have adhered during the entire development of DD3d. First, whenever possible, we choose algorithms that are *conceptually and logically simple*. Second, we intend to make this code as *generic* as possible. Keeping

these design principles in mind should be helpful in understanding the aspects of code development described below.

The reason that we are only interested in conceptually simple algorithms is obvious. A complex algorithm with many *ad hoc* rules is not only aesthetically less appealing, but necessitates complex book-keeping that can be disastrous if one tries to implement (and debug) it in a massively parallel setting. On the other hand, dislocations are known to be peculiar objects: they are topological line defects with a singular elastic field. Not surprisingly, the algorithms for simulating dislocations are necessarily more complex than those for simulating point objects, such as atoms in molecular dynamics (MD). Therefore, our choice of algorithms in DD3d is usually a compromise between conceptual simplicity and computational efficiency.

If the code is *generic* then it can be easily applied to simulate various materials after it is developed and fully tested for one test case. In our development work on DD3d we find that almost all elements of the algorithm deal with various generic issues that are independent of the specific physical system. These include, for example, meshing the dislocations into segments, computing driving forces, and communication between the processors. The system specific parts, on the other hand, can be grouped into one place — the mobility module (details later) that specifies how individual dislocations move in response to the driving force it sees. The separation of the system-specific parts from generic parts is rather similar to that in the commercially available finite element (FEM) codes. This way the code will be able to model a new material once the user defines his/her own material module of interest.

2.1. DATA STRUCTURE

In DD3d dislocations are represented as a set of “nodes” connected with each other by straight line segments, as shown in Fig. 1. The position of nodes, together with their connectivity, is our fundamental degrees of freedom. If a node is connected with n other nodes, we call it a n -node, or a node with n arms, or n neighbors. In Fig. 1, node 1, 2 and 3 are 2-nodes, or “discretization” nodes, while node 0 is a “physical”-node, indicating the position where three dislocations meet.

The Burgers vectors are defined on every arm emanating from the node, with the line direction always pointing away from the node. For example, \vec{b}_{01} is the Burgers vector of the arm going from node 0 to node 1, and \vec{b}_{10} is the Burgers vector of the same arm going in the reverse direction. Therefore the sum rule $\vec{b}_{01} + \vec{b}_{10} = 0$ follows. Furthermore

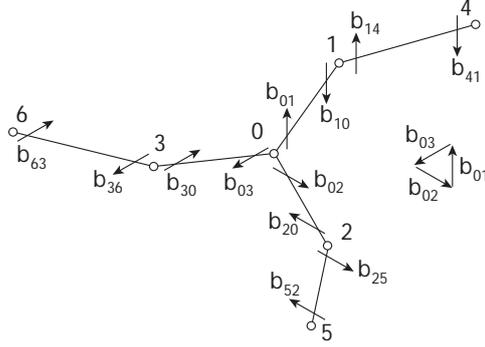


Figure 1. Dislocation network represented as a set of “nodes” (empty circles) interconnected by straight segments (see text).

the total Burgers vector of all arms going out of any given node is also zero, e.g., $\vec{b}_{01} + \vec{b}_{02} + \vec{b}_{03} = 0$.

Under this convention, an arbitrary dislocation network can be uniquely specified by a set of nodes $\{\mathcal{N}_i\}$, each described by its location \vec{r}_i , its connectivity and Burgers vectors of its arms, i.e.,

$$\mathcal{N}_i = \left[\vec{r}_i; I_{ij}, \vec{b}_{ij}, (j = 1, \dots, n_i) \right], \quad (1)$$

where I_{ij} are the indices of the neighboring nodes of node i , and n_i is its total number of neighbors. The node set $\{\mathcal{N}_i\}$ is the data the code deals with.

2.2. GENERIC ALGORITHM

In general, a DD3d computational cycle goes as the following.

1. Compute driving force \vec{f}_i on each node.
2. Compute velocity \vec{v}_i of each node based on \vec{f}_i and local dislocation character.
3. Determine suitable time step Δt .
4. Evolve all dislocation nodes to time $t + \Delta t$, handling topological changes occurring during $[t, t + \Delta t]$.
5. $t := t + \Delta t$. Go to 1.

Except step 2, all the steps above are generic aspects of DD simulations that are not dependent on the material of interest. They will be

discussed in this section. The mobility module (step 2) will be discussed in the following section.

2.2.1. Nodal Force Calculation

The driving force on any given node i can be rigorously defined as (minus) the derivative of the total elastic energy of the dislocation network $E(\{\mathcal{N}_i\})$, with respect to a virtual displacement of the nodal position \vec{r}_i , i.e.,

$$\vec{f}_i = -\frac{\partial E(\{\mathcal{N}_i\})}{\partial \vec{r}_i} \quad (2)$$

In the elasticity theory of dislocations [1], the total elastic energy can be written as the sum of self energies between each segment pairs, such as

$$\begin{aligned} E(\{\mathcal{N}_i\}) &= W_S(01) + W_S(14) + W_S(02) + \dots \\ &\quad + W_I(01, 14) + W_I(01, 02) + W_I(02, 25) + \dots \\ &= \sum_{\langle i,j \rangle} W_S(ij) + \frac{1}{2} \sum_{\langle i,j \rangle; \langle k,l \rangle} W_I(ij, kl), \end{aligned} \quad (3)$$

where $W_S(ij)$ is the self energy of segment (i, j) , and $W_I(ij, kl)$ is the interaction energy between segments (i, j) and (k, l) . Contrary to several earlier claims that the driving force could be infinite due to the existence of sharp corners at the nodes, it can be shown [3] that the driving force in Eq. (2) is well defined, well-behaved and numerically converges to the known analytical solutions for smooth dislocation curves as the discretization becomes finer. It is also shown in [4] that the contributions to nodal driving force \vec{f}_i due to segment interactions, such as $W_I(ij, kl)$ can be evaluated by numerically integrating the stress field of segment (k, l) on segment (i, j) , with proper weights. The contributions from self energies ($W_S(ij)$) on the other hand, are obtained by analytical differentiation. Most of the computational time in DD3d is spent on nodal force calculations, most of which is the evaluation of stress field of one segment on another segment (assuming isotropic linear elasticity). Because periodic boundary condition [5] is used, for every segment the stress field due to an infinite array of its images is also included. The image stress contribution is pre-computed and stored in a table for interpolation during the simulation [6].

2.2.2. Moving the Nodes

For simplicity, we integrate the first order equation of motion describing the over-damped motion of dislocations. This implies that there exists a mobility function (\mathcal{M}), which determines the instantaneous velocity

(\vec{v}_i) of a node, given its instantaneous driving force and local geometry:

$$\dot{\vec{r}}_i \equiv \vec{v}_i = \mathcal{M}(\vec{f}_i) \quad (4)$$

The mobility function \mathcal{M} will be discussed in more detail in the next section. For now let us simply assume that \mathcal{M} is available and can be used to compute all the nodal velocities \vec{v}_i .

At this point, one can imagine the following simple algorithm for a DD simulation. With a pre-selected time step Δt , we can update the position of all nodes by (the forward Euler method),

$$\vec{r}_i := \vec{r}_i + \vec{v}_i \cdot \Delta t \quad (5)$$

However, in practice the velocities of the nodes could vary significantly during the simulation. For accuracy and numerical stability, it is better to use a variable Δt for each integration step. One approach is to put an upper bound (r_{\max}) on the distance any node is allowed to travel during one simulation step. Let $v_{\max} = \max_i |\vec{v}_i|$ be the maximum velocity of all nodes. Then the maximum allowed time step becomes $\Delta t = r_{\max}/v_{\max}$.

One more complication still remains. Remember that our nodes are not simple point objects, instead they are interconnected by dislocation segments. If we simply update nodal positions according to Eq. (5), certain dislocation segments may pass through each other without notice, which would be an unphysical artifact. The segment-segment collisions are accounted for in DD3d in the following way. For every pair of segments, e.g., (1, 2) and (3, 4), given the positions of all four participating nodes at time $t = 0$ ($\vec{r}_1, \vec{r}_2, \vec{r}_3, \vec{r}_4$), and assuming their respective velocities ($\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4$) remain constant, we developed an algorithm to predict whether or not these two segments will collide during period $[0, \Delta t]$, and if they do, when and where will the collision occur. Let us call this algorithm,

$$[\text{col}, t_p, \vec{r}_p] = \text{predictcollision}(\Delta t; \vec{r}_1, \vec{r}_2, \vec{r}_3, \vec{r}_4; \vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4) \quad (6)$$

If there is a collision, `col` returns 1, and $t_p \in [0, \Delta t]$ and \vec{r}_p are the predicted time and location of the collision, respectively. If there is no collision between the two segments, `col` returns zero.

At every time step, after we compute the velocities of all nodes, we use the `predictcollision` algorithm to check for possible collisions between all segment pairs. If there are no collisions at all during $[0, \Delta t]$, then we can safely update the positions of all nodes, and proceed to the next iteration. Otherwise, we have to perform a few sub-iterations to reach the desired time step Δt . Let δt be the time of the first collision. We will move all the nodes to time δt , ($\vec{r}_i := \vec{r}_i + \vec{v}_i \cdot \delta t$), and perform

the necessary topological changes (details below) at that time. After that, we increment the time once again to the next collision time. This procedure is then repeated until the desired time step Δt is reached.

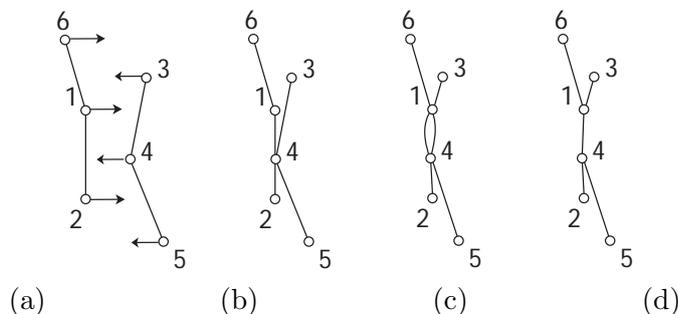


Figure 2. A simple approach to handling the topological changes during dislocation collision by creating a new node at the collision point. (a) Initial state at time 0. (b) At collision time t_p , we replace node 4 with a new node that connects to all four nodes 1 – 3 and node 5. (c) At a later time segment 1-6 and 3-4 collide. Again we create a new node with 4 arms to replace node 1. At this time nodes 1 and 4 become doubly connected. (d) This is resolved by replacing the two arms connecting nodes 1 and 4 with a single arm with Burgers vector equal to the sum of that of the two original arms.

To take into account the topological changes when two dislocation lines meet each other, we adopt the following simple approach. A new node is created at the collision point that connects with all four nodes participating in the collision. Therefore, the new node has 4 arms, as shown in Fig. 2(b). It is interesting to note that by following this very simple algorithm, several different dislocation reaction scenarios are reproduced naturally. To see this, let us follow this algorithm for a few more steps.

As shown in Fig. 2(c), the second collision occurs at a later time, between segments 1-6 and 3-4. Following the above procedure, we introduce a new node with 4 arms to replace node 1. However, this would result in a double connection between nodes 1 and 4, which is obviously redundant. If the sum of the Burgers vectors of these two arms is non-zero, we replace them with one arm with the Burgers vector equal to the sum. This makes a new dislocation segment (a junction), connecting two “physical” nodes – now each with three arms. If, on the other hand, the sum of the two Burgers vector is zero, nodes 1 and 4 are disconnected. Hence, dislocation annihilation occurs.

2.2.3. Parallelism

The all important feature of DD3d is its capability to utilize a large number of processors efficiently in parallel. To date, an efficient usage of 1500 has been demonstrated. To make this possible, all processors are treated equally during the simulation, i.e. there is no distinction such as “master” versus “slaves” between the processors. The total simulation box is divided into rectangular “domains”, each assigned to one processor, as shown in Fig. 3. This way, the communications are mostly *local*, that is, each processor can obtain most of the information it needs by communicating with its nearest neighbors.

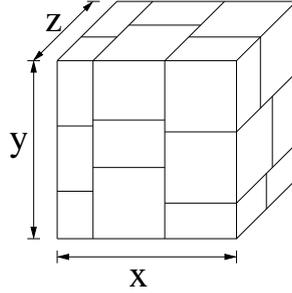


Figure 3. Decomposition of total simulation space into $3 \times 3 \times 2$ domains along x , y , z axes.

Because dislocation microstructures can be highly inhomogeneous, dividing the total simulation box into equally sized domains may lead to severe load imbalance, since some processors may contain a lot more nodes than others. To reach a better load balance, we use the following data decomposition procedure. The total simulation box is first divided into N_x domains along the x direction such that each domain contains equal number of nodes. Each domain is then further divided along y direction by N_y times, and the resulting domains again divided along z direction by N_z times. In the end, we obtain $N_x \times N_y \times N_z$ domains, all containing the same number of nodes, as shown in Fig. 3. However, because the dislocation structure evolves during the simulation, we need to re-partition the problem among processors from time to time, in order to maintain a good load balance. It is found that the optimal number of nodes per domain is in the range from 200 to 1000. In this case, the computational load on each processor is relatively light, while most of the computing time is still spent on computation instead of communication. If and when the total number of dislocation segments increases significantly (e.g. due to dislocation multiplication), we stop and restart the simulation with more processors, to maintain a reasonable simulation speed.

2.3. MOBILITY MODULE

The mobility module is the only material specific part of the DD3d code. It specifies how fast a node should move in response to its driving force. The effects of crystallography and temperature on dislocation mobility are both accounted for here. It is expected that the users will develop their own (possibly very sophisticated) mobility modules to simulate materials of their interest. Here as an example, we describe a simple mobility module that mimics the generic behavior of dislocation in body-centered-cubic (BCC) metals at high temperatures. We call it “pencil-glide” mobility module and use it in the simulation described in the next section.

The “pencil-glide” mobility module is specified by three parameters: the edge mobility M_e , the screw mobility M_s , and a critical angle θ_c . For simplicity, we will only discuss “discretization” nodes here, i.e., nodes with only two neighbors. Let \vec{r}_1 and \vec{r}_2 be the position of the two neighbors of node i , and let $L = |\vec{r}_2 - \vec{r}_1|/2$. Then \vec{f}_i/L is the average Peach-Koehler force around node i . Unit vector $\vec{\xi} = (\vec{r}_2 - \vec{r}_1)/|\vec{r}_2 - \vec{r}_1|$ approximates the dislocation line direction. The dislocation character angle θ is defined through $\cos \theta = |\vec{\xi} \cdot \vec{b}|$. If $\theta < \theta_c$, the dislocation is locally “screw”, otherwise it is “non-screw”. The velocity of “screw” segments is simply $\vec{v}_i = M_s \vec{f}_i/L$. Because it follows the direction of the driving force and is not confined to any plane, this mobility function describes the well-known “pencil-glide” behavior observed in BCC metals at high temperatures. The velocity of “non-screw” segments, on the other hand, is confined within the glide plane, with normal vector $\vec{n} = \vec{b} \times \vec{\xi}/|\vec{b} \times \vec{\xi}|$. Let $\vec{v} = [M_e \sin^2 \theta + M_s \cos^2 \theta] \vec{f}_i/L$, the velocity for “non-screw” dislocation is simply $\vec{v}_i = \vec{v} - (\vec{v} \cdot \vec{n})\vec{n}$.

3. Results

Here we describe the results of initial benchmark simulations using DD3d. The mobility law parameters chosen here are intended to mimic the behavior of BCC metal Mo. For example, screw dislocations have a lower mobility than edge dislocations. Specifically, $M_e = 10b \cdot (\text{s} \cdot \text{Pa})^{-1}$, $M_s = 1b \cdot (\text{s} \cdot \text{Pa})^{-1}$, $\theta_c = \arccos(0.95)$, where $b = 0.27\text{nm}$ is the magnitude of the smallest Burgers vector in BCC Mo.

In these simulations, we used a cubic simulation box with edges along [100], [010] and [001] directions and $10\mu\text{m}$ in length. The initial configuration consists of 8 long screw dislocations (with Burgers vectors along $\frac{1}{2}\langle 111 \rangle$ directions) randomly positioned in the simulation box. The initial dislocation density is around $1.2 \times 10^{11}\text{m}^{-2}$. A uniaxial

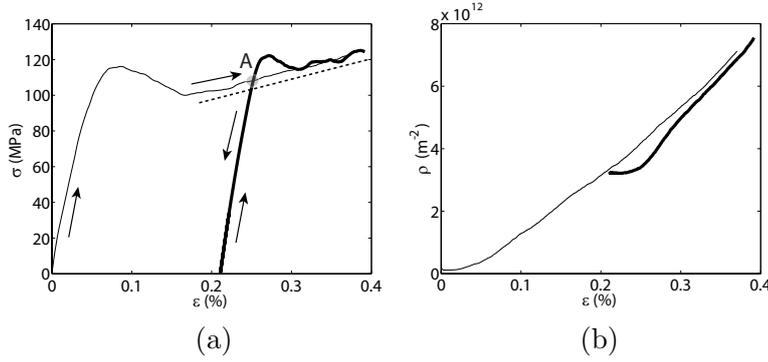


Figure 4. (a) Stress-strain curves produced by DD3d simulations of two uniaxial loading tests along [100] with strain rate $\dot{\epsilon} = 1\text{s}^{-1}$ (see text for more details). The slope of the dashed line indicates the rate of strain hardening. (b) Dislocation density in these two simulations as a function of strain.

tension is then applied along [100] direction under a constant strain rate of $\dot{\epsilon} = 1\text{s}^{-1}$. The thin lines in Fig. 4 are the results from a 12-processor Linux Beowulf cluster after about 6 weeks of the wall-clock time. A total strain reaches about 0.4% at the end of this simulation, while the dislocation density has increased by more than 70 times. The stress-strain curve also exhibits three different behaviors. Initially the response is almost elastic and stress increases linearly. When the upper yield point around 120MPa is reached, the stress drops upon further loading, which indicates a strain softening behavior. Then, after a lower yield point around 100MPa is reached, the stress starts to increase again, this time exhibiting a strain hardening behavior. The slope of the stress-strain curve in this region, i.e., the strain hardening rate $d\sigma/d\epsilon$, as indicated by the dashed line, is around 10GPa.

The thick lines in Fig. 4 correspond to a separate simulation of the same specimen, but with different loading history. After the original specimen was deformed to point A [in Fig. 4(a)], it was unloaded and relaxed under zero stress. We then reloaded it using the same strain rate and temperature, and run the new simulation on a 200-processor Linux cluster for 3 days. It is interesting to note that the new simulation does not follow the original trajectory. Instead, it develops its own upper and lower yield points. However, after the lower yield point, the new simulation enters a strain hardening regime with about the same strain hardening rate as before. This indicates the robustness of the strain hardening behavior observed here.

4. Concluding Remarks

In this paper, we give a brief overview of the new massively parallel dislocation dynamics simulation code DD3d. Our description here is intended to be brief, so that the reader can get a general appreciation about the overall structure of the code, without being distracted by many technical details. To save the space, many general and important aspects are left out, such as remeshing, fast-multipole stress calculations, realistic mobility laws, patterning analysis of simulated dislocation micro-structures, etc. Other issues, such as more efficient parallel collision handling and time-stepping algorithms on 10^4 or more processors, are not completely resolved yet: DD3d is constantly evolving to better address these challenging problems. We expect that DD3d will soon become powerful enough to provide a statistically representative model for dislocation patterning and crystal plasticity. We hope that by exercising this explicit, large scale model, one can obtain new insights that will help the development of more reliable physics-based continuum theories of crystal plasticity.

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48. Benchmark runs of DD3d on 200 to 1500 processors were performed on the MCR cluster of LLNL.

References

- [1] J. P. Hirth and J. Lothe, *Theory of Dislocations*, (Wiley, New York, 1982).
- [2] H. Mughrabi, T. Ungar, W. Kienle and M. Wilkens, *Phil. Mag. A* **53**, 793 (1986).
- [3] Wei Cai, *Atomistic and Mesoscale Modeling of Dislocation Mobility*, Ph.D. Thesis, M.I.T., May (2001).
- [4] Wei Cai, et al. unpublished.
- [5] V. V. Bulatov, M. Rhee and W. Cai, *Mater. Soc. Proc.* **653**, Z1.3.1 (2001).
- [6] Wei Cai, Vasily V. Bulatov, Jinpeng Chang, Ju Li and Sidney Yip, *Philos. Mag. A*, **83**, 539 (2003).

