Module 3: ParaDiS Walkthrough

Frank-Read source and strain-hardening

We will be using the `ParaDiS` code for discrete dislocation dynamics in FCC Al. We will explore this using both the compiled C-code version of `ParaDiS` as well as a simpler MATLAB implementation called `DDlab` to see dislocation dynamics in action. These codes are freely available (after signup) from the ParaDiS website; there are copies of these on the EWS linux workstations, including the compiled versions of the `ParaDiS` executables. There is also documentation for both available on EWS in the `/class/mse404pla/ParaDiS/doc` directory. The directories of interest are:

- Documentation for `ParaDiS` and `DDLab`: `/class/mse404pla/ParaDiS/doc`
- ParaDiS: `/class/mse404pla/ParaDiS/2.5.1`
- DDLab: `/class/mse404pla/ParaDiS/DDLab`

Note: DDLab is really designed for testing, and makes reference to an older style of input format for ParaDiS (v2.2).

Initial setup

First, we will create a directory structure with the files we need to run both `ParaDiS` and `DDLab`. In your home directory, create a new subdirectory for this walkthrough (in this example below, it will be `~/MSE404pla/ParaDiS`):

```
mkdir -p ~/MSE404pla/ParaDiS
cd ~/MSE404pla/ParaDiS
cp -av /class/mse404pla/ParaDiS/DDLab .
cp -av /class/mse404pla/ParaDiS/2.5.1/bin .
mkdir Runs
```

This will (1) make new subdirectory called `MSE404pla/ParaDiS` in your home directory, and change into it, (2) copy over all of the `DDLab` code into a subdirectory, (3) copy over the `ParaDiS` executables, and (4) make a new directory to store your input and output files, called `Runs`.

DDLab: Frank-Read source

`DDLab` is set up to run a discrete dislocation dynamics simulation in MATLAB, with live visualization: you can see the dislocation evolving, as well as new nodes being created, destroyed, etc. It uses very similar algorithms to `ParaDiS`, so you can use it to see what you expect to happen in initial steps of a run. It doesn't support everything that `ParaDiS` does (no periodic boundary conditions, no constant strain rate) nor does it use the fastest possible algorithms, but can give you a flavor of how the code runs, and what types of input is necessary to run a discrete dislocation dynamics simulation.

First, let's run something! Change to your `DDLab` directory, and launch MATLAB:

```
cd DDLab
```

```
module load matlab
matlab
```

If you look in the directory, you will see all of the different functions used to run the DDD simulation. You will also see some utilities to write paradis input files, too. There is a subdirectory called `Inputs` that are some example input files for `DDLab`. We'll look at these files in more depth shortly, but first let's run a Frank-Read source simulation. In MATLAB:

```
run('Inputs/input_frank_read.m')
dd3d
```

Note: after you run the first command, look at your Workspace: the input file has loaded the parameters for the DDD simulation as parameters, which means that you can also make changes to these before running the dd3d simulation if you choose.

When you run `dd3d`, you will see a warning about compilation (it is compiling the `SegSegForces.c` file); you can safely ignore this. It will open up a live visualization window, so that you can see the Frank-Read source in BCC under stress. You will see it grow very quickly, and the number of nodes will increase. The output in the main MATLAB window will be the step number and the time step, `dt` at each iteration. The code uses a "trapezoid" time integrator which is a linear combination of an implicit and explicit Euler scheme; the implicit Euler requires multiple evaluations of forces, but does allow for larger time steps than an explicit Euler scheme. It also allows for an adaptive time step: the integrator takes the largest time step it can for a given amount of error in the trajectory. The velocity `v` is the velocity of a particular node (selected by the variable `printnode`).

The `totalsteps` parameter determines how many steps the simulation will run; in this case, it is set to 200. After the simulation finishes, you can continue the run by typing `dd3d` again. Note: this Frank-Read source does not pinch off in a straightforward way, in part, because it uses a BCC mobility law where the screw dislocation segments can crossslip at will (so-called "pencil glide"); if you look close to the location of the pinch-off, you will notice that the loop has not remained in the slip plane. This is due to the rotational forces on the dislocation segments.

DDLab: Input files

Next, let's look at the input file for the Frank-Read source, to understand the structure, and how to make changes. **Note:** both `DDLab` and `ParaDiS` use SI units for all parameters *except* distances, which are all normalized to the dislocation Burgers vector.

This is `input_frank_read.m`:

```
%Frank-Read source

global Bscrew Bedge Beclimb Bline

rn    = [ 1000 1000 1000  7;
         -1000  -1000 -1000  7;
             0    0     0  0];
```

```
links = [1 3 0.5 0.5 0.5 -1 1 0;
         3 2 0.5 0.5 0.5 -1 1 0];

MU = 1;
NU = 0.305;
maxconnections=8;
lmax = 1000;
lmin = 200;
areamin=lmin*lmin*sin(60/180*pi)*0.5;
areamax=20*areamin;
a=lmin/sqrt(3)*0.5;
Ec = MU/(4*pi)*log(a/0.1);
totalsteps=200;
dt0=1e7;
mobility='mobbcc0';
%Drag (Mobility) parameters
Bscrew=1e0;
Bedge=1e0;
Beclimb=1e2;
Bline=1.0e-4*min(Bscrew,Bedge);

integrator='int_trapezoid';
rann = 0.5*a;
rntol = 0.5*rann;
doremesh=1;
docollision=1;
doseparation=1;
plotfreq=1;
plim=10000;
appliedstress =1e-3.*[2 0 1; 0 2 -1; 1 -1 0];
viewangle=[45 -45 ];
printfreq=1;
printnode=3;
rmax=100;

% simulation box size (for paradis)
L = 40000;
minSideX = -L/2;
minSideY = -L/2;
minSideZ = -L/2;
maxSideX =  L/2;
maxSideY =  L/2;
maxSideZ =  L/2;

% boundary conditions (for paradis)
```

```
xBoundType = 1; % free
yBoundType = 1; % free
zBoundType = 1; % free

% output and communication settings
paradis_dir = '../..';
paradis_input_dir = strcat(paradis_dir,'/Runs');
paradis_output_dir = strcat('Outputs/frank_read_results');
```

There's a lot here, but let's go through it systematically. There are a series of variables defined near the end (all marked with comments `(for paradis)`) that are only used for output to create ParaDiS input files; they plan no role in the DDD simulation. Let's go through the rest of the input file. First:

```
global Bscrew Bedge Beclimb Bline
```

identifies the global mobility parameters (drag coefficients) that will be parameterized below. Then,

```
rn    = [ 1000 1000 1000  7;
         -1000  -1000 -1000  7;
             0    0     0  0];
links = [1 3 0.5 0.5 0.5 -1 1 0;
         3 2 0.5 0.5 0.5 -1 1 0];
```

The first array, `rn`, are the nodes. Each row is a single node; the entries in the row are

```
[ x y z constraints ]
```

where it's the position, followed by either 0 for a free node, or 7 for a constrained node. A similar format will follow for ParaDiS. The second array, `links`, specify how two nodes are connected with a dislocation:

```
[ node1 node2 bx by bz nx ny nz ]
```

which connects node1 (index) to node2 (index) with the Burgers vector in Cartesian coordinates, and the normal of the slip plane. The dislocation line starts at `node1` and ends at `node2`. So, our simulation has three nodes, two of which are fixed (the "ends" of the Frank-Read source) and a single node in the center that is mobile.

Next, material parameters μ (shear modulus) and ν (Poisson ratio)

```
MU = 1;
NU = 0.305;
```

followed by remeshing parameters (max and min length, max and min "area"):

```
lmax = 1000;
lmin = 200;
areamin=lmin*lmin*sin(60/180*pi)*0.5;
areamax=20*areamin;
```

The lengths are for segments (whether they should be split, or joined), while the areas are for nodes (looks at the triangular area of a node with its two neighbors), and hence is a measure of curvature.

More material properties (a is the "core radius" for the non-singular force calculation, and E_{core} is the core energy per length per Burgers vector squared—energy per volume):

```
a=lmin/sqrt(3)*0.5;
Ec = MU/(4*pi)*log(a/0.1);
```

Next, we specify the mobility law and parameters:

```
totalsteps=200;
dt0=1e7;
mobility='mobbcc0';
%Drag (Mobility) parameters
Bscrew=1e0;
Bedge=1e0;
Beclimb=1e2;
Bline=1.0e-4*min(Bscrew,Bedge);
```

We already discussed the `totalsteps`; `dt0` is the upper limit on the size of the timestep. Then, `mobility` is the name of the function that will calculate node velocities based on node forces, and then followed by the parameters used by that function. In particular, the velocity of node i is proportional to the force as

$$\mathbf{v}_i = \frac{\mathbf{f}_i}{B \sum_j L_{ij}/2}$$

where the sum is over all nodes j connected to i, and L_{ij} is the length of the segment. The first three parameters are self-explanatory: drag for screw and edge segments, and climb drag. The final parameter, `Bline`, is the drag coefficient contribution *along the direction of the dislocation line*; it is small but nonzero to help prevent the matrices from becoming singular in pathological cases.

Next are the evolution parameters,

```
integrator='int_trapezoid';
rann = 0.5*a;
rntol = 0.5*rann;
doremesh=1;
docollision=1;
doseparation=1;
```

The parameter `rann` control the distance for annihilation of dislocations; `rntol` is the solution tolerance in our trapezoidal integrator. The last three parameters control what sort of updating is done on the dislocation evolution.

Finally, there are the output parameters,

```
plotfreq=1;
plim=10000;
appliedstress =1e-3.*[2 0 1; 0 2 -1; 1 -1 0];
```

```
viewangle=[45 -45 ];
printfreq=1;
printnode=3;
rmax=100;
```

which controls the plotting, viewing angle, command line printing, and finally how far a node can travel in one time step (`rmax`), and the applied stress.

To see what happens when we change some parameters, reload the Frank-Read simulation, turn off remeshing, and run:

```
run('Inputs/input_frank_read.m')
doremesh=0;
dd3d
```

Before running: what do you expect to happen?

Now, try reloading the file, and increasing the applied stress by a factor of 4:

```
run('Inputs/input_frank_read.m')
appliedstress = 4*appliedstress;
dd3d
```

Does it behave in a way that you expect?

ParaDiS: Input files

For the remainder of the walk-through, we'll work with `ParaDiS` simulations. We can start by creating input files from `DDLab`. First, let's reload our initial geometry, then write out the input files:

```
run('Inputs/input_frank_read.m')
paradis_input_dir = '../Runs'
write_paradis_input
```

Note: we specified the directory where the new input files would be created. If successful, you should see two files in your `Runs` subdirectory:

```
matlab_input.cn
matlab_input.data
```

The first file is the "control" file that tells `ParaDiS` what to do, while the second is the node file. *Note:* this is not the only way to construct these files! We will explore some other scripts to create the data file.

We're not quite ready to run yet; we need some support files to help with the treatment of the boundary conditions. You will have different files that depend on (1) the treatment of the elastic interaction (Fast-Multipole Method or not), (2) the boundary conditions (PBC or free), and (3) the material parameters (μ and ν).

Let's first consider the operation without the FMM method. Create a new subdirectory, in the same level as your `Runs` subdirectory, called `inputs`. We'll make two of these for our run, using the program `bin/stresstablegen`; one for PBC, and one without. The following commands will create those inputs (run from your `ParaDiS` directory):

```
bin/stresstablegen -nopbc -outfile inputs/Rijm.cube.out
bin/stresstablegen -pbc -outfile inputs/RijmPBC.cube.out
```

Note: these commands take a while to run. To save time, feel free to copy the corresponding files out of the `/class/mse404pla/ParaDiS/2.5.1/inputs` directory into your `inputs` directory.

```
cp -av /class/mse404pla/ParaDiS/2.5.1/inputs .
```

This will also copy over some other control files, like `paradis.xdefaults`, which controls how the visualization window will be displayed. Feel free to edit this file; the parameters are fairly self-explanatory.

We will need to do a little housekeeping before running. First, we will see in the control file that it will specify where to put the final output, and that will be in a directory called `Outputs`; so, create that directory:

```
mkdir Outputs
```

ParaDiS: Frank-Read source input

Next, let's edit the control file `Runs/matlab_input.cn` to make sure everything is as we expect. Edit the file, and make sure that the lines:

```
dirname = Outputs/frank_read_results
Rijmfile = "inputs/Rijm.cube.out"
RijmPBCfile = "inputs/RijmPBC.cube.out"
```

point where you expect (i.e., do you have an `Outputs` directory? Is your `inputs` directory correctly named, and the files there that you expect? Case matters! `Inputs` is not the same as `inputs`). Don't worry too much: if you've messed this up, `ParaDiS` will complain that it can't find corresponding files, or can't create the new directory.

We'll go through the rest of the file now. First, note that # is used to identify comments. Any line that starts # is ignored, and once # appears on a line, the rest of the line is ignored. Next, let's look at the other input parameters you'll see the following blocks.

```
#Total simulation steps
maxstep = 200

#The total number of CPUs should be numXdoms * numYdoms * numZdoms
#numXdoms = 1
#numYdoms = 1
#numZdoms = 1
```

```
#Cells for dislocation grouping (cannot be less than 3)
numXcells = 3
numYcells = 3
numZcells = 3
```

The first specifies how many time steps you'll use. The next block is for running in parallel; we are working with the serial version of ParaDiS, so this will always stay with the default of $1 \times 1 \times 1$. Next, the number of "cells" that is used to break up the dislocation-dislocation interaction calculation: inside a cell, interactions are all direct, while outside a cell, a multipole expansion is used.

The next parameters specify if (and how) the Fast Multipole Method is used:

```
#Fast multipole method specs.
fmEnabled = 0  #disable fast multipole
fmMPOrder = 2
fmTaylorOrder = 4
fmCorrectionTbl = "Inputs/fm-ctab.m2.t4.DAT"
```

Note: the FMM table is generated by using `ctablegen` and needs to be consistent with the order of expansion listed here. If `fmEnabled = 0`, then we don't use it.

Next, the integrator:

```
timestepIntegrator = "backward-euler"
```

Change this to "trapezoid". The only recognized options are `"trapezoid"` and `"forward-euler"`, and it will default to trapezoid.

Next, the geometry of the simulation box

```
#Simulation box size
xBoundMin = -20000.000000
xBoundMax = 20000.000000
yBoundMin = -20000.000000
yBoundMax = 20000.000000
zBoundMin = -20000.000000
zBoundMax = 20000.000000

#Boundary conditions
xBoundType = 0
yBoundType = 0
zBoundType = 0
```

The boundary conditions are either periodic (`0`) or free (`1`). Set the boundaries to periodic for now.

Next are the materials properties:

```
#Fundamental length unit
burgMag = 1.0

#Elastic constants
```

```
shearModulus = 1.000000
pois = 0.305000
```

followed by the mobility laws

```
#Mobility law function
mobilityLaw = "BCC_0"

MobScrew = 1.000000e+00
MobEdge  = 1.000000e+00
MobClimb = 1.000000e-02
```

Note: we do not need the "line" mobility that was previously defined.

```
#Discretization
maxSeg = 1000.000000
minSeg = 200.000000

#Maximum nodal displacement at each time step
rmax = 100

#Error tolerance in determining time step
rTol = 14.4338

#Maximum time step
maxDT = 1e+07
```

These parameters have the same meaning as in `DDLab`. Note: `rTol` should be changed to 0.5. Next, we have the core size and energy. Change `rc` to 1, and add the parameter for `rann` with a value of 0.5 (the distance for annihilation).

```
#Core cut-off radius
rc = 57.735027

#Core energy
Ecore = 5.059893e-01

#Turn on elastic interaction
elasticinteraction = 1
```

Next, we have the applied stress:

```
#Applied stress in Pa (xx,yy,zz,yz,zx,xy)
appliedStress = [ 0.002 0 0.001 0 0.002 -0.001 ]
```

Finally, we have information about how often and what to write as output:

```
#Save files
savecn = 1
savecnfreq = 100
```

```
preserveOldTags = 1
```

Next, we'll run our simulation!

ParaDiS: Frank-Read source run

The Frank-Read simulation can be started with

```
bin/paradis Runs/matlab_input.cn
```

This will then begin the simulation, and will open an X window visualizer of the evolving dislocation. **Note:** paradis runs significantly faster than the MATLAB version does. You can control the visualization window with a few commands:

- `p` pauses and restarts
- `r` enables rotation
- `s` enables scaling

There are other commands described in the `ParaDiS` documentation. Note also that the default is for the window to disappear after 3 seconds of the simulation finishing.

The `Outputs/frank_read_results` directory now contains restart files, too.

If we want to output different properties, we need to add the command

```
saveprop = 1
```

which will create a `properties` subdirectory with our properties.

Try rerunning the simulation with different stresses, and different maximum times, to see how it performs.