ARL-TR-9255 • Aug 2021

Finite Element for Discrete Dislocation Dynamics (FED3) User Guide

by Joshua C Crone, Kenneth W Leiter, and Jaroslaw Knap

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9255 ● AUG 2021

Finite Element for Discrete Dislocation Dynamics (FED3) User Guide

by Joshua C Crone, Kenneth W Leiter, and Jaroslaw Knap
Computational and Information Sciences Directorate,
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2021	Technical Report	October 2011–September 2021

4. TITLE AND SUBTITLE
Finite Element for Discrete Dislocation Dynamics (FED3) User Guide

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Joshua C Crone, Kenneth W Leiter, and Jaroslaw Knap

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLC-EM
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-9255

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
primary author's email: <joshua.crone.civ@mail.mil>. ORCIDs: Joshua Crone, 0000-0003-0856-9149

14. ABSTRACT
The Finite Element for Discrete Dislocation Dynamics (FED3) simulator was developed to extend the modeling capabilities of discrete dislocation dynamics (DDD) beyond infinite single-crystal domains. It was designed to run concurrently with existing DDD simulators as a separate executable to enable efficient use on high-performance computing resources. This report serves as a user guide, describing how to build and run the simulator. Example simulations are described to highlight the capabilities of FED3.

15. SUBJECT TERMS
discrete dislocation dynamics, DDD, plasticity, microstructure, finite elements

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Joshua C Crone
Unclassified	Unclassified	Unclassified	UU	37	19b. TELEPHONE NUMBER (Include area code) 410-306-2156

Contents

List of Figures

List of Tables

Acknowledgments

The Finite Element for Discrete Dislocation Dynamics (FED3) simulator was developed through the financial support of the US Army Combat Capabilities Development Command Army Research Laboratory's Enterprise for Multiscale Research in Materials. Computational support, through a grant of computer time, was provided by the DOD High Performance Computing Modernization Program. Theoretical discussions, code support, and user feedback were provided by numerous individuals. The authors wish to especially thank DEVCOM Army Research Laboratory researchers (past and present) JJ Ramsey, Lynn Munday, Ben Szajewski, Chi-Chin Wu, and Peter Chung, as well as Lawrence Livermore National Laboratory researchers (past and present) Sylvie Aubry, Tom Arsenlis, Gregg Hommes, Jaehyun Cho, Brett Wayne, Meijie Tang, and Moon Rhee.

1. Introduction

The Finite Element for Discrete Dislocation Dynamics (FED3) simulator was developed to extend the modeling capabilities of discrete dislocation dynamics (DDD) beyond infinite single-crystal domains. This includes accounting for free-surface effects due to both external surfaces (e.g., thin films, micropillars) and internal surfaces (e.g., void) as well as the effects of material heterogeneities (e.g., precipitates, grains, multilayer films). Elastic effects are accounted for through a superposition of stress fields,[1] where the stress fields to correct the infinite single-crystal stress fields are computed by solving a boundary value problem (BVP) via the finite element (FE) method.[2,3] Geometric effects are accounted for by modifying dislocation nodal positions as they encounter surfaces and interfaces. The specific tasks performed by FED3 are the following:

1. Extract free surfaces and interfaces as 2-D manifolds of the 3-D FE mesh.
2. Extract triple lines as 1-D manifolds of the 3-D FE mesh at the boundary of three or more material heterogeneities.
3. Compute boundary conditions and solve BVP for the corrective stress fields.
4. Compute Peach–Koehler (PK) forces on dislocation segments due the corrective stress fields.
5. Detect when dislocations encounter free surfaces or interfaces of material heterogeneities.
6. Write output files for the FE mesh and dislocation segments at specified time steps for visualization and debugging.

The main computational challenges to incorporating FE into DDD simulations are the inherent differences in domain decomposition and load balancing. Due to the rapid multiplication of dislocations during a simulation, dynamic load balancing is desired, where the boundaries of the parallel domain decomposition evolve to maintain an equal number of dislocation segments on each process. In contrast, the FE computations are most efficient when the number of elements on each processor are equal, where a static domain decomposition strategy is preferred. Furthermore, the

relative computational cost of the FE and DDD calculations will vary if, for example, one is modeling a few dislocation segments in a highly refined FE mesh versus modeling hundreds of thousands of dislocation segments in a coarse FE mesh. To address the required flexibility in load balancing and domain decomposition, the FED3 simulator has been developed as a separate, parallel executable, rather than integrating the FE calculations directly into an existing DDD software, following the multiple program, multiple data (MPMD) paradigm. This allows a user to independently select the number of CPU resources for the FED3 and DDD components and enables the use of independent domain decomposition strategies. Coupling of FED3 to the DDD simulator is achieved through the use of an HDF5-based Distributed Shared Memory (DSM) library.[4]

While FED3 should be compatible with any DDD simulator that is modified to send/receive segment information to/from FED3, this work was developed in close collaboration with the ParaDiS team at Lawrence Livermore National Laboratory (LLNL). FED3 has been fully integrated into a development version of ParaDiS. Incorporating FED3 into another DDD simulator would be a significant undertaking; therefore, it is recommended that the ParaDiS coupling be used if possible. Contact LLNL for more information on the requirements to access this developmental version of ParaDiS. FED3 was written to be compatible with two versions of ParaDiS. One being the standard, single-crystal version where dislocations must be contained in a single material (i.e., the matrix material). FED3 is also compatible with a polycrystal version of ParaDiS currently under development[5] that allows for dislocations to be contained in multiple materials/grains with rules for transmission across interfaces. FED3 performs slightly different tasks depending on which version of ParaDiS it is coupled with, but the only change to the user experience is in the input options of Section 4.

2. How to Build

2.1 FED3

FED3 relies on a number of external dependencies that must be compiled prior to building FED3. To simplify the build process, a Python-based build script is included, along with tarballs of all required dependencies except an Message Passing Interface (MPI) implementation, which is also required. A detailed README.md file is included in the distribution, but a brief overview is provided here. Note that

advanced users can compile each dependency and FED3 by hand for greater control of compile and install options, rather than using the Python build script. Directions for building each component manually are also included in the README.md file.

The simplest way to build FED3 is by designating a single install directory, $FED3_INSTALL_DIR, and running the following command from the top directory of the FED3 distribution:

```
$ python ./build-FED3.py --mpicc mpiicc --mpicxx
  mpiicpc --mpifc mpiifort --install-dir
  $FED3_INSTALL_DIR --scratch-dir /tmp/tmpBuild
```

where IntelMPI wrappers have been employed. The MPI wrapper scripts should be changed depending on the MPI implementation used and absolute paths may be provided if multiple MPI implementations are available on the system. Choosing a path in /tmp for scratch directory is not necessary and may be undesirable if the user wants to keep the files generated when configuring and compiling each code. Additional options for the the Python build script are displayed when passing "-h" as a command-line argument to the script. Upon successful compilation, the FED3 executable will be located at $FED3_INSTALL_DIR/bin/fed3.

2.2 ParaDiS

As previously mentioned, a developmental version of ParaDiS has been coupled to FED3. Compiling the FED3-enabled version of ParaDiS is similar to compiling the standard version. For detailed instructions on how to build ParaDiS, users are referred to the ParaDiS documentation. This guide only discusses the FED3-specific aspects of building ParaDiS.

The first step to enabling FED3 in ParaDiS is by setting the ARLFEM preprocessor macro. Throughout the ParaDiS source code, any FED3-specific block of code is surrounded by "#ifdef ARLFEM," which can be enabled through the compiler flag "-DARLFEM". To be consistent with the way other macros are defined in ParaDiS, ARLFEM can be enabled by adding "DEFS += -DARLFEM" to the makefile.setup file. In addition, the line "DEFS += -DDEBUG_ARLFEM" will enable logging and debugging statements specific to the coupling with FED3.

The second step is to "cd fem" into the fem directory, which has been added to

the FED3-enabled version of ParaDiS. In this directory, there are additional makefiles and source files that are only compiled when coupled to FED3. The makefile should be modified so that the include and .lib directories for HDF5 and H5FDdsm are added to the compile line. The following lines of code will accomplish this if HDF5 and H5FDdsm were installed in the same directories. Additional lines will be needed to add all necessary paths when dependencies are installed in separate locations:

```
ARLFEM_PATH = ${FED3_INSTALL_DIR}
INCS += -I$(ARLFEM_PATH)/include
LIBS += -L$(ARLFEM_PATH)/lib
LIBS += -Wl,-rpath,$(ARLFEM_PATH)/lib -lhdf5
LIBS += -Wl,-rpath,$(ARLFEM_PATH)/lib -lH5FDdsm
```

Once the makefile has been updated, running "make" in the fem directory will build a FED3-enabled version of ParaDiS and place the resulting executable (named paradisfem) in the bin directory.

2.3 Additional Information on the External Dependencies

Tarballs of the external dependencies are packaged with FED3. In some cases, newer versions of these dependencies may be used, but compatibility is not guaranteed, particularly if the application programming interface (API) of any of the dependencies has changed. It is recommended to use the version of each dependency that is packaged with FED3 if possible. Particular care should be taken when choosing a version of HDF5. The H5FDdsm code requires a version of HDF5 with support for the virtual file driver (VFD) to enable reading and writing of HDF5 data to memory rather than to a file. The Extensible Data Model and Format (XDMF) also requires HDF5 and can use the same VFD-enabled version that is required by H5FDdsm to avoid confusion. The hierarchy of dependencies is shown in the following multilevel list, where all packages in sublevels are requirements for the package in the level above. Dependencies in blue are not required, but are recommended to significantly reduce the computational cost of solving the BVP.

- FED3
    - H5FDdsm
        - HDF5-VFD

- XDMF
  - HDF5
  - METIS
  - EXODUS II
    - NetCDF
- PETSc
- HYPRE

3. How to Run

3.1 Generating Input Files

The FED3 executable requires two input files in addition to any input files required by the DDD simulator.

1. Text input file with lines containing "keyword = value". Description of the input parameters and possible values are provided in Section 4. Note that the input file parser assumes the file is space delimited and expects spaces on both sides of the equal sign between the keyword and value.
2. An FE mesh using the XDMF[6] file format. This mesh must contain the coordinates of the vertices, connectivity of the elements, nodesets to define boundary conditions (see Section 4), and element sets defining block IDs (see Section 4).

The XDMF FE mesh can be generated through multiple methods. The XDMF library can be called directly in c++ or in Python through Simplified Wrapper and Interface Generator (SWIG) wrappers, which can be built with the XDMF library (not built by default). However, the recommended method is by generating an Exodus II file using a mesh generation tool such as CUBIT[7] from Sandia National Laboratories. Python scripts for generating example meshes in CUBIT are provided in the examples folder included in the FED3 distribution. The following are some helpful tips when generating meshes:

- FED3 supports tetrahedral and hexahedral elements of both linear and quadratic order. Preliminary results suggest that quadratic tetrahedral elements provide the best ratio of accuracy to computational cost.[2]

- Since the largest gradients in the FE stress field are located near surfaces and interfaces, it is recommended that the FE domains are meshed coarsely and surface refinements are performed to concentrate mesh resolution near the surfaces and interfaces.
- Nodeset names are case-sensitive.
- After numerous geometry and meshing operations, gaps may form in the vertex numbering. The Portable, Extensible Toolkit for Scientific Computation (PETSc) solver (see Section 4 for discussion of solver options) requires that the FE vertices are numbered consecutively. In CUBIT, it is recommended that the following command is executed before exporting a mesh to eliminate any issues with non-consecutive numbering of vertices: "renumber node all in volume all start_id 1 uniqueids".
- When block IDs from Exodus II are converted to XDMF, the lowest block ID is mapped to 0, the next lowest is mapped to 1, and so on. So if a mesh is generated with block IDs 1 and 3, these will be converted to 0 and 1, respectively, during the conversion to an XDMF file.

The XDMF build described in Section 2 comes with a binary for converting Exodus II files to XDMF files with the command

```
$ <FED3_INSTALL_DIR>/bin/XdmfExodusConverter input.exo
```

This command will produce "input.xmf" and "input.h5" files, which store the light and heavy data, respectively, for the XDMF file.

When running FED3 across N processes, the FE mesh must also be partitioned across N partitions. The XDMF build includes a mesh partitioner that uses the METIS graph partitioning library[8] to equally distribute the elements across partitions. This is accomplished with the command

```
$ <FED3_INSTALL_DIR>/bin/XdmfPartitioner input.xmf N
```

which produces input_pN.xmf and input_pN.h5 files, where N is an integer ≥ 2.

3.2 Running Locally

The simplest way to run the coupled FED3 simulator is locally on a workstation or single CPU node across N cores, where N $\geq$ 2. The N resources can be partitioned between FED3 and the DDD simulator as desired. Throughout the remainder of this document, it is assumed that the variables $N_FE and $N_DD have been set to the desired number of processes for each executable, where N_FE+N_DD=N. The FED3 executable must be called first to initialize the DSM communicator. By default, a file named ".dsm_client_config" is written to the home directory, which the DSM communicator in the DDD simulator will look for to establish a connection between codes. The following commands are an example of how the coupled applications are executed using the mpirun launcher for MPI applications. The launcher and command-line arguments will vary depending on the MPI implementation used to compile the codes and the options of the specific system. Note the FED3 executable is run in the background so that the DDD executable can be called.

```
$ mpirun -np $N_FE ./fed3 inputFEmesh_p${N_FE}
  input.param &
$ mpirun -np $N_DD ./paradisfem -d input.data
  input.ctrl
```

3.3 Running across Multiple Nodes

When running the coupled applications across multiple nodes, care must be taken to ensure the applications are distributed across all nodes. Otherwise, depending on the behavior of the MPI launcher, it is possible that both FED3 and the DDD simulator would execute on the same subset of available nodes, significantly reducing the performance. Most MPI launchers require a machinefile (often referred to as a hostfile) to identify the available nodes for use. This file is usually passed via command-line arguments or dedicated environment variables. These command-line arguments or environment variables should be modified such that the MPI launchers for FED3 and the DDD simulator receive different sets of hostnames. An example is given below using mpirun and assuming that a machinefile named "nodefile.txt" has been generated that lists each available node repeated n times, where n is the number of processes to use per node. The machinefile is split into two partitions depending on the number of processes requested for the FED3 and DDD executables. These separate machinefiles are then passed to the respective mpirun commands. If an MPI launcher relied on an environment variable, instead of a command-line ar-

gument, that environment variable could be redirected to “FE_Nodefile.txt” before FED3 is executed, and then redirected again to “DD_Nodefile.txt” before the DDD application is executed. Note this step is not required when using “srun” because processes are distributed across all available resources by default.

```
$ head -n $N_FE nodefile.txt > FE_Nodefile.txt
$ tail -n $N_DD nodefile.txt > DD_Nodefile.txt
$ mpirun -np $N_FE --machinefile FE_Nodefile.txt
  ./fed3 inputFEmesh_p${N_FE} input.param &
$ mpirun -np $N_DD --machinefile DD_Nodefile.txt
  ./paradisfem -d input.data input.ctrl
```

3.4 Submitting to a Batch Queue

When running a single FED3 simulation at a time, the default location for the distributed shared memory (DSM) configuration file of “$HOME/.dsm_client_config” is sufficient. However, if multiple simulations may occur at the same time, such as when submitting jobs to a batch queuing system, the DSM configuration files may get overwritten before the DDD simulator is able to complete its coupling to FED3. To avoid this issue, the path to the directory for the DSM configuration file can be overwritten. The DSM communicator is implemented to check the $H5FD_DSM_CONFIG_PATH variable before using the default file path. The following is an example where multiple jobs can be submitted, each with a unique variable $JOBID. This unique identifier could be manually set, or may be automatically available with the batch queuing system.

```
$ mkdir ${HOME}/.DSM_${JOBID}
$ export H5FD_DSM_CONFIG_PATH=${HOME}/.DSM_${JOBID}
```

4. Input Parameters

FED3 requires a space-delimited text file with lines containing “keyword = value”. Next is a description of the various keywords and available options. Default values are shown in parentheses. Keywords that do not have a default value must be specified in the input file.

4.1 Elastic Interactions

The following parameters determine which elastic interactions are included and how often the BVP solver updates the stress fields on the FE domain:

- **IncludeBvpSolve = 0|1 (1)**

  Boolean to indicate if the BVP should be solved. If this flag is set to 0 (false), then FED3 will not compute forces on the dislocation segments and will only perform surface/boundary detection. Note if even if image and polarization stress are not being computed, this flag should still be set to 1 when applying external boundary conditions.

- **IncludeImageStress = 0|1 (1)**

  Boolean to indicate if image tractions on free surfaces should be included in the BVP. If set to 1 (true), then the IncludeBvpSolve flag should also be set to 1; otherwise, the image stress and resulting image forces on the dislocation segments will not be computed.

- **IncludePolarizationStress = 0|1 (0)**

  Boolean to indicate if polarization body forces due to elastic heterogeneities in the FE domain should be included in the BVP. If set to 1 (true), then the IncludeBvpSolve flag should also be set to 1; otherwise, the polarization stress and resulting polarization forces on the dislocation segments will not be computed.

- **BvpSolveFreq = n (1)**

  Frequency at which image tractions, polarization body forces, and BVP solves are computed (when corresponding flags are true). The default value of $n = 1$ computes a new stress field on the FE domain every time a force calculation is initiated by the DDD simulator. PK forces are still integrated every timestep based on the new positions of the dislocation segments, even if the stress field is unchanged. When image and/or polarization stresses are being computed, $n = 1$ is the most accurate scheme since a new stress field should be computed each time a dislocation moves. However, if it is assumed that dislocations do not move significantly from one timestep to the next, it may be acceptable to keep the stress field from a previous configuration for multiple

timesteps. When image and polarization stress are not being computed, but external boundary conditions are applied, n can be made very large such that the BVP solve is only computed on the first timestep. Since the stress field is independent of the dislocations, the stress will be constant and does not need to be recomputed at every timestep, significantly reducing the computational cost.

4.2 Boundary Conditions

Boundary conditions in FED3 are specified through nodesets that must be defined when creating the FE mesh. Each nodeset must have a unique name, which is specified in the following input parameters. Names are case-sensitive.

- **DirichletNodeset = Name** $u_x\ u_y\ u_z$

  Fix the displacement of all nodes in the nodeset to the specified values. For any entry u_i, the word *free* can replace a numeric value to indicate that the displacement in that direction should remain free. For example, "DirichletNodeset = Bottom 0.0 free free" would fix the x component of displacement to be zero but would leave the nodes in the "Bottom" nodeset to be free to move in the y and z directions.

- **TractionNodeset = Name** $\sigma_{xx}\ \sigma_{yy}\ \sigma_{zz}\ \sigma_{yz}\ \sigma_{xz}\ \sigma_{xy}$

  Apply a traction force on free-surface elements where all nodes of the element are contained in the nodeset. The surface normal is automatically computed for each element, multiplied by the specified stress tensor, and integrated over the surface element to compute the nodal traction forces. If only a subset of the nodes within an element are contained in the nodeset, traction forces are not computed for that surface element.

- **ForceNodeset = Name** $F_x\ F_y\ F_z$

  Directly apply a force to nodes within a nodeset. In practice, Dirichlet and traction boundary conditions are used more frequently.

All boundary condition types can have multiple entries with different nodeset names and different boundary condition values. For example, to fix the bottom surface while stretching the top surface along the z direction:

- DirichletNodeset = Bottom 0.0 0.0 0.0
- DirichletNodeset = Top 0.0 0.0 1.0

Nodes can also be contained in multiple nodesets. If both Neumann (traction or force) and Dirichlet bondary conditions are applied to the same node, then the Dirichlet boundary conditions are applied and the Neumann conditions are ignored. If multiple Dirichlet boundary conditions are specified, then the last one in the input file is applied. If multiple Neumann boundary conditions are specified, they are added together.

4.3 Quadrature Rules

The following are the parameters for the quadrature rules:

- **VolumeQuadRule = tet1 | tet4 | tet5 | tet11 | tet15 | hex1 | hex6 | hex2x2x2 | hex13 | hex3x3x3 | hex34 | hex4x4x4**

  Quadrature rule for integrating over the 3-D volume elements. Both tetrahedral and hexahedral elements are supported in FED3 with quadrature rules "tet$N$" and "hex$N$", respectively, where N is the number of quadrature points. When there are no polarization forces, N only needs to be large enough to integrate the element stiffness matrix and will therefore depend on whether linear or quadratic elements are used. However, when polarization forces are computed, the dislocation stress field must be integrated over the elements to compute body forces. This dislocation stress field may have large gradients, where higher quadrature may improve the accuracy of the integration. Further analysis is required to determine the optimal tradeoffs in accuracy and computational cost for increasing the quadrature order versus increasing FE mesh resolution.

- **SurfaceQuadRule = tri1 | tri3 | tri4 | tri6 | tri7 | tri13 | quad1 | quad4 | quad9 | quad4x4 | quad5x5**

  Quadrature rule for integrating over the 2-D surface elements. The element shape (triangle or quadrilateral) and element order (linear or quadratic) are automatically determined by the 3-D volume element type specified in the input mesh. When there are no image forces, the quadrature order only needs to be large enough to integrate the shape functions and will therefore depend on

whether linear or quadratic elements are used. However, when image forces are computed, the dislocation stress field must be integrated over the elements to compute image tractions. This dislocation stress field may have large gradients, where higher quadrature may improve the accuracy of the integration. Further analysis is required to determine the optimal tradeoffs in accuracy and computational cost for increasing the quadrature order versus increasing FE mesh resolution.

- **LineQuadRule = line1 | line2 | line3 | line4 | line5 | line10 | line20 | line100 | line200 | line500 | line1000 (line3)**

  Quadrature rule for integrating the PK force along 1-D line elements representing the dislocation segments. The ideal number of quadrature points will depend on how much the FE stress field varies along a dislocation segment. The FE stress field will be constant or linear across an element, depending on whether the volume element is linear or quadratic, respectively. However, a dislocation segment may span multiple volume elements. Therefore, the choice of LineQuadRule should depend on the order of the volume elements, the expected gradients in the FE stress field, and the ratio of dislocation segment lengths to volume element size. One to five quadrature points is typically sufficient, the higher quadrature rules were developed for a separate application, but are included in FED3 for completeness.

4.4 Solver Options

The following delineate the parameters for solver options:

- **SolverType = hypre | petsc | linearcg (linearcg)**
  - hypre: A parallel, matrix-based iterative linear solver using LLNL's HYPRE library.[9] The preconditioner is set to use the algebraic multigrid with a fixed set of parameters that were determined to optimize the serial and parallel performance. The HYPRE library includes a variety of preconditioners and parameter options, which can be selected by modifying the source code in the "HypreSolver.cc" file. See Crone and Munday[10] for an assessment that was performed to identify the default preconditioner options. To use this solver, FED3 must be compiled with HYPRE.

  - PETSc: A parallel sparse direct solver using MUMPS[11] through the PETSc interface.[12] On the first BVP solve, this solver performs a time- and memory-intensive Gaussian factorization. However, each subsequent solve only requires a matrix-vector multiplication and is therefore significantly faster than iterative solvers. This solver type is best when the simulation will require many BVP solves, where the high computational cost of the factorization will be offset by the speed of many subsequent solves. See Crone and Munday[10] for an assessment and discussion on the tradeoffs between direct and iterative solvers in FED3. Note that the memory requirements increase significantly as the number of degrees of freedom in the FE mesh increase. Therefore, if this solver fails, try to distribute the simulation across more CPU nodes or use other methods to increase the available memory. To use this solver, FED3 must be compiled with PETSc.
  - linearcg: A parallel, matrix-free iterative linear solver employing conjugate gradient with Jacobi (diagonal) preconditioning. This solver is the default since it does not rely on external libraries and has the smallest memory requirement. However, this solver is significantly slower than the PETSc and HYPRE solvers and should be avoided if possible.

- **SolverTol =** f **(**1×10^{-6}**)**

  Convergence tolerance for the iterative solvers. When the two-norm of the residual vector divided by the two-norm of the right-hand-side vector is below the specified tolerance, then the solver has converged.

- **MaxIterations =** n **(**1×10^{4}**)**

  Maximum number of iterations for the iterative solvers. If the convergence tolerance has not been reached by the specified number of iterations, a warning is printed and the FE mesh is dumped to a file named "femMesh_failedSolved.xmf" to assist in debugging.

- **SolverDebugLevel = 0|1|2 (1)**

  Integer to determine how much information is printed to stdout from the solvers: 0 does not print anything, 1 prints minimal information, and 2 prints additional information. Each solver is different in how it determines what information is printed out for values of 1 and 2.

4.5 Debug Options

The following provides the debug options:

- **DebugStatements = 0|1 (1)**

  Boolean to control if debug statements are printed to stdout. If the value is 1 (true), then debug statements are periodically printed throughout the FED3 simulation to indicate what stage of the computation is occurring, the timing of various stages, how many segments were received, and so on. Note the linear solvers have a separate flag to determine the verbosity of that stage of the computation.

- **WriteBoundaryMesh = 0|1 (0)**

  Boolean to control if separate meshes should be written for the extracted boundary features. If the value is 1 (true), then surface meshes of free surfaces and block interfaces are written during initialization. Line meshes are also written containing the triple-lines between three or more blocks, if they exist.

- **FixBoundaryOutputFreq =** n **(0)**

  Integer to control the frequency at which dislocation nodal data is written to an XDMF file before and after calls to the boundary detection algorithm. A value of 0 indicates that the files should never be written, while $n > 0$ writes the dislocation nodal data every n^{th} call to the boundary detection algorithm. These files can be useful in debugging any issues with the boundary detection method. Note this option is only available when using the node-based boundary detection method that is compatible with the polycrystalline version of ParaDiS under development. This output option is not available for the segment-based surface detection algorithm that is employed when coupled to the single-crystal version of ParaDiS.

4.6 Multiple Blocks

Blocks are a way to identify separate regions of the FE domain. They are created during mesh generation and are essentially a list of elements associated with that block. The way blocks are treated, and the required input parameters to FED3, will

depend on whether FED3 is coupled with the developmental polycrystal ParaDiS or the original single-crystal ParaDiS.

When coupling FED3 with the developmental polycrystal ParaDiS, each block should specify a separate grain. The elastic constants and crystal orientation of each grain are then defined within ParaDiS input files. There are no block-specific input parameters that must be specified in the FED3 input file.

When coupling to the single-crystal version of ParaDiS, blocks allow for the introduction of elastic heterogeneities (e.g., hard inclusions inside a matrix material, multilayer thin film). Care should be taken to ensure that dislocations are only contained within blocks that have the same elastic constants as what is specified in ParaDiS. Otherwise, there will be inconsistencies between the dislocation stress fields computed in ParaDiS and FED3. When elastic heterogeneities are harder than the matrix material containing dislocations, the polarization stress can act to naturally keep the dislocations out of the secondary material. However, dislocations will be pulled into the second materials if they are softer. The current implementation of FED3 does not have a way to prevent or indicate when this has occurred.

- **NumberBlocks =** N **(1)**

  Number of blocks contained in the FE mesh. An error occurs if this number is inconsistent with the number of blocks found in the mesh.

- **Mu = [Space-delimited list of length** N**] (DD)**

  Shear modulus values for each block. For any component of the list "DD" can be specified rather than a numeric value. This indicates that the shear modulus for that block should be taken from the value specified in the ParaDiS ctrl file. For example, if the the FE mesh has three blocks and the ParaDiS value should be used for the first and third blocks and a value of 300 GPa for the second block, the input command would be the following: Mu = DD 300.0e9 DD. Note the units must be consistent with what is used in ParaDiS.

- **Nu = [Space-delimited list of length** N**] (DD)**

  Poisson's ratio for each block. Same "DD" option as described for Mu.

Note when coupling with the polycrystal version of ParaDiS, the block information is communicated directly from ParaDiS. In which case, these input parameters are

completely ignored. This section is also unnecessary if the FE domain does not contain elastic heterogeneities (e.g., a cylinder or free-standing thin film). The default values are set for these types of simulations.

5. Output Files

Since the purpose of FED3 is to augment the capabilities of the DDD simulator it is coupled with, any output for quantitative analysis will still come from the DDD simulator. The output from FED3 is intended only for visualization and debugging. Two output options are described in Section 4. The ParaDiS coupling provides an additional output option that is specified in the ParaDiS ctrl file. At specified intervals, XDMF files for the FE mesh and the dislocation segments can be written to a directory called "fem" within the ParaDiS results directory specified by "dirname". The writing of these files are controlled in the ParaDiS ctrl file using the options "savefem", "savefemfreq", and "savefemdt" the same way that "savecn", "savecnfreq", and "savecndt" are used for writing the ParaDiS restart files. All output files can be read into a number of visualization tools such as VisIt from LLNL and ParaView.

The FE mesh contains many fields on the mesh used for the calculation of the image/polarization forces, such as the nodal tractions, stresses, and displacements at a particular snapshot during the simulation. The files are named "femMeshI.xmf" and "femMeshI.h5", where I is the snapshot counter indicating that this is the I^{th} time FED3 is dumping the output mesh. When running FED3 in parallel, separate mesh files are written for each process. The filenames are "femMeshI_n.xmf" and "femMeshI_n.h5", where n is the processor ID. Two scripts are provided in the "utils" directory at the top level of the FED3 distribution to combine the partitioned mesh output files. To combine the files for a range of I values, the script "convertOutputMesh.sh" is provided. This script must be run from within the results directory containing the output mesh files. The two required arguments are the number of processes FED3 was run on and maximum snapshot index. When no other arguments are provided, this will combine and convert every file up to the specified maximum snapshot index. A third command-line argument can be provided to choose a starting snapshot index other than 1. A fourth command-line argument can be provided to choose a stride length other than 1. For example, the following command will combine output files "femMesh0004_*", "femMesh0006_*",

"femMesh0008_*", and "femMesh0010_*" for a simulation in which FED3 was run on 32 processes:

```
$ ./convertOutputMesh.sh 32 10 4 2
```

This scripts requires the Python script "XdmfCollectionGenerator.py" (also included in "utils") to generate an XDMF file that points to all of the partitioned mesh files for that snapshot. This file can then be read by VisIt, ParaView, and other visualization packages. An "unpartitioned" mesh file is also created that combines the heavy data in each of the femMeshI_*.h5 files into a single file. Finally, the XDMF file is converted to an Exodus file, which has the effect of converting all vector and tensor values to scalars, making visualization easier with some visualization packages. The file "FED3_xdmf_visitExpressions.xml" is also provided in "utils", which can be loaded into VisIt expressions to extract the scalar values directly from the XDMF files, rather than using the Exodus converter. Note the user must edit the "convertOutputMesh.sh" file to provide paths to "XdmfCollectionGenerator.py" and the Xdmf utilities that are compiled with XDMF.

The dislocation segment output file contains fields associated with image/polarization forces (both nodal and per-segment forces), Burger's vector, segment length, and surface/boundary constraint. The files are named "femSegmentI.xmf" and "femSegmentI.h5", where I corresponds to the same snapshot counter as the mesh files.

6. Examples

Examples are provided to demonstrate some of the features of FED3. With each example, there is a Python script that employs CUBIT to generate input meshes. The scripts are provided for users who want to modify the dimensions of the FE domain and/or the mesh resolution. The scripts can also be used as starting points to create scripts for more complicated geometries.

6.1 Image Force on Prismatic Loop

The first example contains a circular prismatic loop, parallel to a free surface. Due to the availability of an analytical solution, this example provides a means for assessing the accuracy and convergence of the image force with respect to the FE mesh size as a dislocation approaches a free surfaces.

An analytical solution for the image force perpendicular to the surface, f, was de-

termined by Bastecka[13]:

$$f = \frac{b^2\mu}{16R(1-\nu)}\left\{\frac{-3\alpha^4+7\alpha^2+2}{\alpha[\sqrt{1+\alpha^2}]^5}E_o\left(\sqrt{\frac{1}{1+\alpha^2}}\right) + \frac{3\alpha^3-\alpha}{[\sqrt{1+\alpha^2}]^5}K_o\left(\sqrt{\frac{1}{1+\alpha^2}}\right)\right\} \tag{1}$$

where R is the radius of the prismatic loop, d is the distance from the prismatic loop to the free surface, $\alpha = d/R$, $E_o(k)$ and $K_o(k)$ are the elliptic integrals, μ is the shear modulus, ν is Poisson's ratio, and b is the magnitude of the Burger's vector.

The simulation setup is described in detail in Section 4.2 of Crone et al.[2] In summary, a cubic FE domain of length L is created using the Python script "GenPrisLoop_cubit.py". L should be large enough to minimize boundary effects from the side and bottom surfaces since the analytical solution assumes the dislocation is in a half space. The Python script includes an option to designate a smaller cubic region near the center of the +Z surface with refined mesh resolution when tetrahedral elements are used. The prismatic loop should be parallel to the +Z surface and centered at $(0, 0, \frac{L}{2} - d)$ (Fig. 1 shows an example schematic). The FE mesh provided in the example was generated with the following commands:

```
$ python ./GenPrisLoop_cubit.py --outerL 200000.0
  --innerL 4000.0 --outerH 20000.0 --innerH 250.0
  --elem-type tetra10 -sr 2
  -f PrisLoop_L200K_4K_H20K_250_tet10_sr2
$ <FED3_INSTALL_DIR>/bin/XdmfExodusConverter
  PrisLoop_L200K_4K_H20K_250_tet10_sr2.exo

$ <FED3_INSTALL_DIR>/bin/XdmfPartitioner
  PrisLoop_L200K_4K_H20K_250_tet10_sr2.xmf 8
```

The FED3 input file "input_prisLoop_hypre.param", ParaDiS ctrl file "prisLoop.ctrl", and ParaDiS data file "prisLoop_L200K_R1K_d500.data" are provided to perform a simulation for a prismatic loop with $R = 1000b$ and $d = 500b$. A text file with the solution to Eq. 1 for these parameters is provide in "AnalyticalSoln.txt". When running the FED3 simulation with DebugStatements = 1, each component of the image forces is summed over all dislocation segments. This value is printed to std-

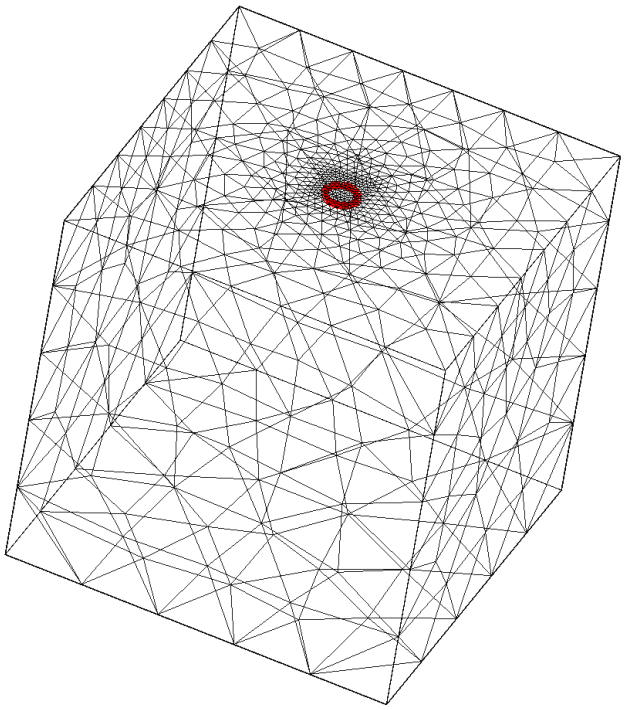

Fig. 1 Schematic of prismatic loop simulation setup. Mesh has been generated with tetrahedral elements and a refined region surrounding the red dislocation loop.

out with the other debug statements and can be used as a quick way to compare the numerical solution to the analytical solution. Template scripts are provided showing how this example can be run with the PBS and Slurm queuing systems.

6.2 Frank–Read Source with Hard Inclusion

The second example demonstrates the effects of elastic heterogeneities through calculation of the polarization force. The setup includes a Frank–Read (FR) source at the center of the domain with a spherical inclusion offset in the direction of the bowing FR source (Fig. 2). The mesh provided in the examples is generated with the following command:

```
$ python ./GenInclusion_cubit.py -Lx 10000
  -Ly 10000 -Lz 1000 -R 200 -x 1000 --outerH 500
  --inclusionH 50 --elem-type tetra10  -sr 1
  -f Inclusion_10Kx10Kx1K_R200_H500_50_tet10_sr1
```

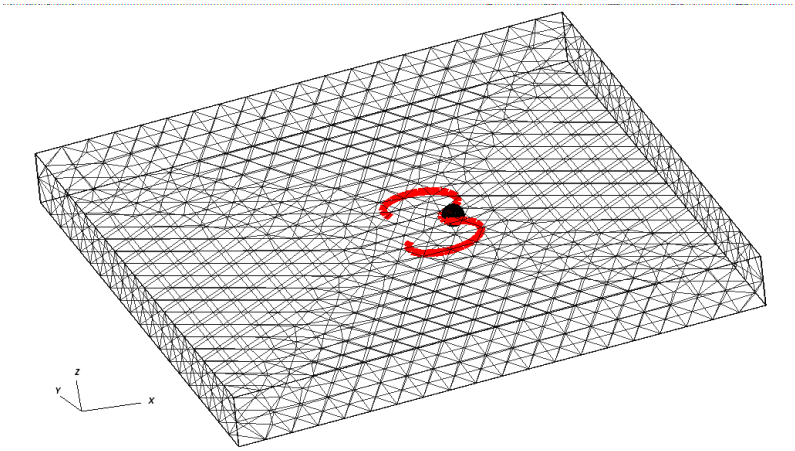

Fig. 2 Schematic of FR source with hard inclusion simulation setup. The bowed out FR source is shown in red.

The element size inside the inclusion is controlled separately from the element size of the matrix material and surface refinements can be applied to the interface between the matrix and inclusion. This ensures the mesh resolution is focused in the regions where stress gradients are largest. An example mesh is shown in Fig. 3 for the parameters indicated in the previous command, with a mesh ratio of 10:1 and one level of surface refinement. Note the matrix and inclusion must be defined by different element blocks so that different elastic constants can be assigned to each region. The colors in Fig. 3 represent the different block IDs.

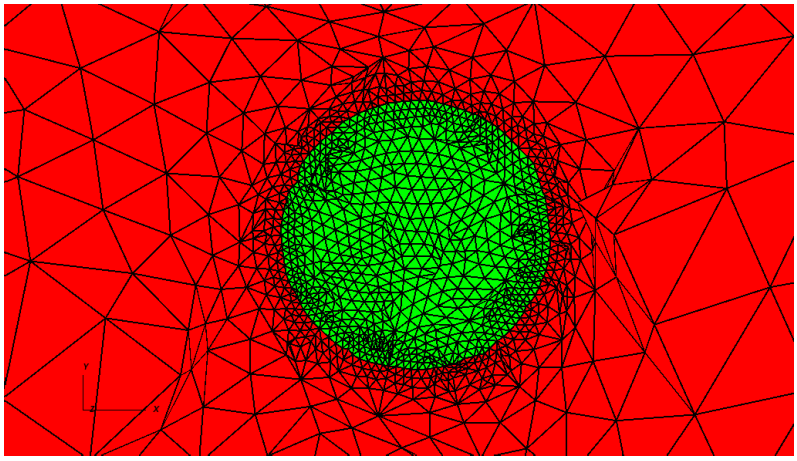

Fig. 3 Mesh refinement at the volume and interface of the inclusion. Distortions in the shapes of the inclusion and elements are due to artifacts of the slicing scheme used for visualization.

The simulation applies a shear stress to force the FR source to bow out in the +X direction. Various combination of stresses can be applied to capture different physics. The applied far-field shear stress can be applied as either a uniform stress in ParaDiS or by applying tractions to the external surfaces of the FE mesh in FED3. The latter will include stress concentrations due to elastic mismatch, which are independent of any dislocations and will not be included if the far-field shear stress is applied directly in ParaDiS. The polarization stress can be turned on or off via the input parameters described in Section 4 to control whether the dislocation interacts elastically with the inclusion. These two options are independent, resulting in four simulation conditions briefly outlined in Table 1. Each simulation condition requires a different pair of input files to ParaDiS and FED3. All input files required to simulate the four conditions are included with the example and listed in Table 2.

Table 1 Simulation options for including stress effects due to elastically mismatched inclusion

No.	Far-field stress	Polarization stress	Resulting condition
1	ParaDis	Off	Uniform stress field throughout simulation domain. FR source sees no effect from inclusion, bows out as it would in infinite single crystal.
2	ParaDis	On	Far-field stress is uniform in domain, but FR source interacts elastically with inclusion through polarization stress computed at each time step in FED3.
3	FED3	Off	Includes stress concentrations due to applied stress of elastically heterogeneous material. This stress field is independent of dislocation configuration and only needs to be computed once at the beginning of the simulation.
4	FED3	On	Includes both stress concentrations and elastic interactions between FR source and inclusion. Stress field evolves at each time step and must be recomputed.

Table 2 Input files corresponding to the simulation conditions listed in Table 1. ParaDiS data file and FED3 mesh files are independent of simulation conditions.

No.	ParaDiS ctrl file	FED3 param file
1	frs_withFarField.ctrl	noFarField_noPolar.param
2	frs_withFarField.ctrl	noFarField_withPolar.param
3	frs_noFarField.ctrl	withFarField_noPolar.param
4	frs_noFarField.ctrl	withFarField_withPolar.param

This example provides a good opportunity to explore parameters related to the computational efficiency. A user may change the mesh resolution to change the number of FE elements and/or change the number of processes the FED3 code is distributed over. Also, since this example requires many timesteps (as opposed to the single timestep in the first example), it is a good opportunity to explore the tradeoffs between the HYPRE and PETSc solvers discussed in Section 4.

6.3 Nanopillar

The final example is a nanopillar undergoing tensile loading at a constant strain rate. In this example, ParaDiS computes the accumulated plastic slip and applies the corresponding applied stress. Options are available to run the code with and without computing the image forces. In the latter, FED3 is only responsible for identifying where dislocations intersect the free surfaces of the cylinder and computing the surface normal at that location.

Two meshes are provided: a coarse mesh, which should be sufficient for the case where image forces are not computed; and a second mesh with two levels of surface refinement to improve the resolution of the higher gradients of the image stress near the free surface. The meshes were created using the Python script provided and the following commands.

```
$ python ./GenCylinder_cubit.py -R 1000 -L 4000
  --elem-size 250 --elem-type tetra4 -sr 0
  -f Cylinder_R1K_L4K_h250_tet4_vr0_sr0_ed1
$ python ./GenCylinder_cubit.py -R 1000 -L 4000
  --elem-size 250 --elem-type tetra4 -sr 2
  -f Cylinder_R1K_L4K_h250_tet4_vr0_sr2_ed1
```

The initial dislocation content is 10 FR sources randomly distributed throughout the nanopillar on randomly assigned FCC slip systems. A Python script, "writeParadisDataFile_RandomFRS_hollowCylinder.py", is provided to generate the ParaDiS data file. Parameters at the top of the script can be modified to change the length and density of the FR sources, as well as the dimensions of the cylinder.

This example is meant as a capabilities demonstration, not necessarily a physically realistic simulation in which quantitative analysis is appropriate. Therefore, the following results are provided for demonstration purposes only. In Fig. 4, the stress–strain curves, provided by ParaDiS, are plotted with and without inclusion of image forces. Qualitatively, these results are consistent with expected trends. The yield stress is lowered when image stresses are included because there is additional force pulling the FR source toward the surface as it bows out. Therefore, the applied stress for the FR source to reach a critical configuration is lowered. Figure 5 contains an example visualization of the FED3 output. The 3-D contours indicate regions with image stresses above 25, 50, and 75 MPa. The mesh is shown in white and the dislocations are shown in red. Note the full Cauchy stress tensor of the image stress is provided in the output mesh. The von Mises stress is shown here as a convenient scalar quantity for visualization purposes.

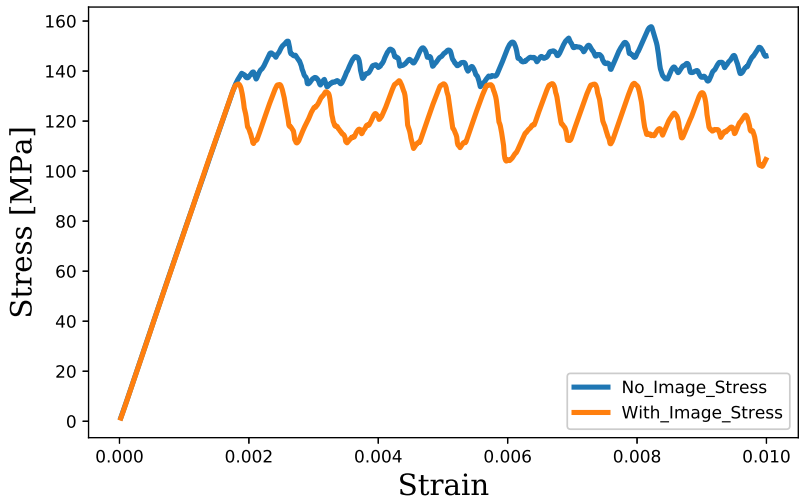

Fig. 4 Stress–strain curves for a nanopillar under tensile loading

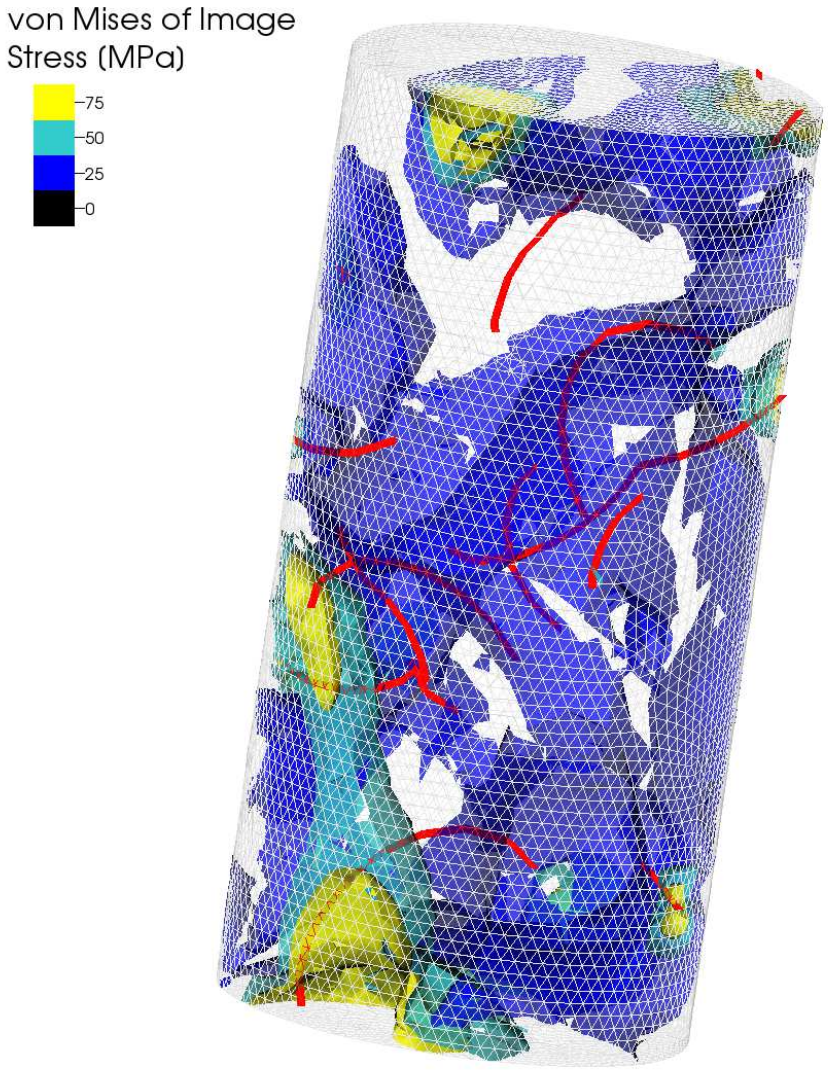

Fig. 5 Visualization from a snapshot of the nanopillar simulation, generated by LLNL's VisIt package; 3-D contours indicate regions with image stress is above 25, 50, and 75 MPa. The mesh is shown in white and the dislocations are shown in red.

7. Conclusion

FED3 provides a parallelized simulator for capturing elastic and geometric interactions between dislocations and arbitrarily shaped material heterogeneities. This user guide provides an introduction to building and running FED3. For a detailed description of the underlying physics and computations implemented in FED3, please refer to the following papers: Crone et al.,[2] Leiter et al.,[14] and Cho et al.[5]

8. References

1. van der Giessen E, Needleman A. Discrete dislocation plasticity: a simple planar model. Modelling and Simulation in Materials Science and Engineering. 1995;3:689-735.

2. Crone JC, Chung PW, Leiter KW, Knap J, Aubry S, Hommes G, Arsenlis A. A multiply parallel implementation of finite element-based discrete dislocation dynamics for arbitrary geometries. Modelling and Simulation in Materials Science and Engineering. 2014;22(3):035014.

3. Weygand D, Friedman L, der Giessen EV, Needleman A. Aspects of boundary-value problem solutions with three-dimensional dislocation dynamics. Modelling and Simulation in Materials Science and Engineering. 2002;10:437–468.

4. Soumagne J, Biddiscombe J, Clarke J. An HDF5 MPI virtual file driver for parallel in-situ post-processing. Recent Advances in the Message Passing Interface. 2010;6305:62–71.

5. Cho J, Crone JC, Arsenlis A, Aubry S. Dislocation dynamics in polycrystalline materials. Modelling and Simulation in Materials Science and Engineering. 2020;28(3):035009.

6. Clarke JA, Namburu RR. A distributed computing environment for interdisciplinary applications. Concurrency and Computation: Practice and Experience. 2002;14(13-15):1161–1174.

7. Sandia National Laboratories. CUBIT: Geometry and Mesh Generation Toolkit. Sandia National Laboratories; 2020. https://cubit.sandia.gov/.

8. Karypis G, Kumar V. METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

9. Falgout RD, Yang UM. HYPRE: a library of high performance preconditioners. In: International Conference on Computational Science; p. 632–641.

10. Crone JC, Munday LB. Parallel performance of linear solvers and preconditioners. Army Research Laboratory (US); 2014. Report No.: ARL-TR-6778.

11. Amestoy PR, Duff IS, L'Excellent JY. MUMPS multifrontal massively parallel solver version 2.0. 1998.

12. Balay S et al. PETSc users manual. 2019.

13. Baštecká J. Interaction of dislocation loop with free surface. Czechoslovak Journal of Physics. 1964;14:430–442.

14. Leiter K, Crone J, Knap J. An algorithm for massively parallel dislocation dynamics simulations of small scale plasticity. Journal of Computational Science. 2013;4:401–411.

List of Symbols, Abbreviations, and Acronyms

TERMS:

1-D – one-dimensional

2-D – two-dimensional

3-D – three-dimensional

API – application programming interface

BVP – boundary value problem

CPU – central processing unit

DDD – discrete dislocation dynamics

DSM – distributed shared memory

FE – finite element

FED3 – Finite Element for Discrete Dislocation Dynamics

FR – Frank–Read

ID – identification

LLNL – Lawrence Livermore National Laboratory

MPI – Message Passing Interface

MPMD – multiple program, multiple data

PETSc – Portable, Extensible Toolkit for Scientific Computation

PK – Peach–Koehler

SWIG – Simplified Wrapper and Interface Generator

VFD – virtual file driver

XDMF – Extensible Data Message Format

MATHEMATICAL SYMBOLS:

α – d/R

b – Burger's vector magnitude

d – distance from prismatic loop to the free surface

f – image force of a prismatic loop in direction of free surface

μ – shear modulus

ν – Poisson's ratio

R – radius of prismatic loop

MATHEMATICAL OPERATORS:

$E_o()$ – complete elliptic integral of the second kind

$K_o()$ – complete elliptic integral of the first kind

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL RDRL DCL TECH LIB
3 (PDF)	DEVCOM ARL FCDD RLC EM J CRONE J KNAP K LEITER