LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-644292

Dislocation microstructure analysis of multi-junctions in BCC single crystals for dislocation dynamics simulations

M. Tang, R. Cook, M. Rhee, G. Hommes, A. Arsenlis

September 26, 2013

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Dislocation microstructure analysis of multi-junctions in BCC single crystals for dislocation dynamics simulations

Meijie Tang, Rich Cook, Moone Rhee, Gregg Hommes, Tom Arsenlis
Lawrence Livermore National Laboratory, Livermore CA 94551

Abstract

A new algorithm is developed to uniquely and completely trace through the whole dislocation network without ambiguity and to identify the dislocation multi-junctions. As a result of such analysis, the multi-junction spatial and temporal evolution is identified and the characteristic dislocation lengths between multi-junctions are obtained in order to construct constitutive relations at large strains.

1. Introduction

Large-scale dislocation dynamics simulations performed by ParaDiS have generated vast amount of information on dislocation microstructure evolutions and mechanical responses during plastic deformation process in the simulations. The general outputs from such simulations are stress vs. strain relation, dislocation density vs. strain relation, as well as dislocation flux vs. strain [1]. While some of these outputs can be compared to experimental data and used to build constitutive relations for materials strength, the underlying mechanisms for yield stress and strain hardening arise from the key features of dislocation microstructure such as strong pinning points, which is not straight-forwardly available from ParaDiS standard outputs. Post processing algorithms are usually best suited for such analysis.

For BCC single crystals, the controlling mechanism for strain hardening is due to the formation and pinning of the dislocation multi-junctions [2]. The multi-junctions are formed due to the close interactions of three or more dislocation lines. They can form under certain conditions and also possible to dissolve. The multi-junctions present themselves in a few variants and evolve in space and time. There is no priori knowledge as to where to look for them except tracing through the whole dislocation network. The goal of the algorithm is to identify the multi-junctions and reconstruct the dislocation network as linked meta-arms between the multi-junctions. Information obtained through such analysis will connect to constitutive models. In addition, the algorithm can be integrated with Visit to enhance the visualization tools of the dislocation microstructures.

2. Dislocation Dynamics Simulations

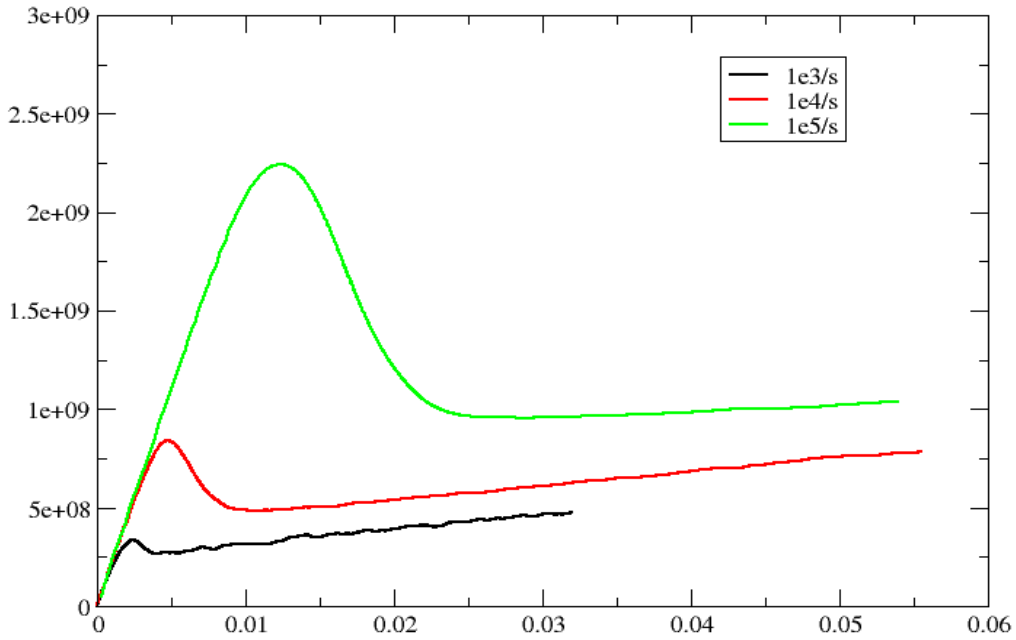

Figure 1. Stress (Pascale) vs. total strain curves obtained from the DD simulations for single crystal Fe using linear mobility laws.

A series of DD simulations for single crystal Fe using linear mobility laws at different strain rates has been performed. It has 3 sets of simulations at the applied strain rate of 10^3/s, 10^4/s, and 10^5/s. The loading direction is along [001]. The simulation box size is 2 micrometers each side. We use this set of simulations as an example for the microstructure analysis. The stress strain curves are shown in Figure 1.

3. Algorithm for identifying dislocation multi-junction nodes

The algorithm to look for multi-junctions can be very complicated. It must take into considerations of all possible variants of multi-junctions and their complex connectivity with the surrounding dislocation neighboring environments.

The detailed steps in the algorithm are discussed in the Appendix. We highlight the important components in the algorithm in this section. The algorithm traces through the whole dislocation network following the dislocation segments' Burgers vectors. The basic Burgers vectors in BCC are of [111] type. Their interactions can lead to formation of binary junctions with Burgers vectors of [200] type. In addition, we have observed segments that have high energy

Burgers vectors such as [113], or [220]. The formation of segments with these high energy Burgers vectors is not expected and deserves further investigation. However, considering the high strain rates in these simulations, dislocations can be over driven to form temporary high energy configurations. These segments with high energy Burgers vectors are very few in quantities (less than 1% of the total number of segments in the simulations). However, their existence can break the ability to trace through and spoil the analysis. The solution is to decompose the high energy Burgers vector into two or more lower energy Burgers vectors, which are given by the Burgers vectors of the neighboring segments. For instance, a triple node has 3 segments and their Burgers vectors are [113], [-1-1-1], and [00-2]. We then decompose the Burgers vector [113] as the following: [113]=[111]+[002]. We replace the original segment [113] with two overlapping segments with Burgers vector of [111] and [002] respectively. After the decomposition, the triple node now becomes a quadruple node with Burgers vectors of [111], [002], -[111], and –[002].

Further more, we realize that if we continue to decompose the [200] type of Burgers vectors into the basic [111] type, the whole dislocation network then consists of only [111] type of lines. The advantage for this decomposition scheme is that all variants of multi-junctions now reduce to the same simple form. That is a dislocation node with 4 neighbors that include all 4 [111] Burgers vectors, i.e., [111], [11-1], [1-11], [-111]. This node is a multi-junction node.

The dislocation line between any two of the multi-junction nodes is called meta-arms. The whole dislocation network now consists of multi-junction nodes (or M node) and meta-arms between the M nodes. The dislocation line length of the meta-arm is called meta-arm length. The straight distance between any two M nodes is called EP distance.

4. Analysis results

Using the algorithms discussed above, we analyzed the dislocation configurations obtained from the set of simulations in Section 2.

Figure 2 shows the dislocation loops density evolution compared with the total dislocation density evolution. Initially, there are no M nodes and meta-arms, the whole dislocation network consists of dislocations in giant loops. When multi-junctions start to form, M nodes start to appear,

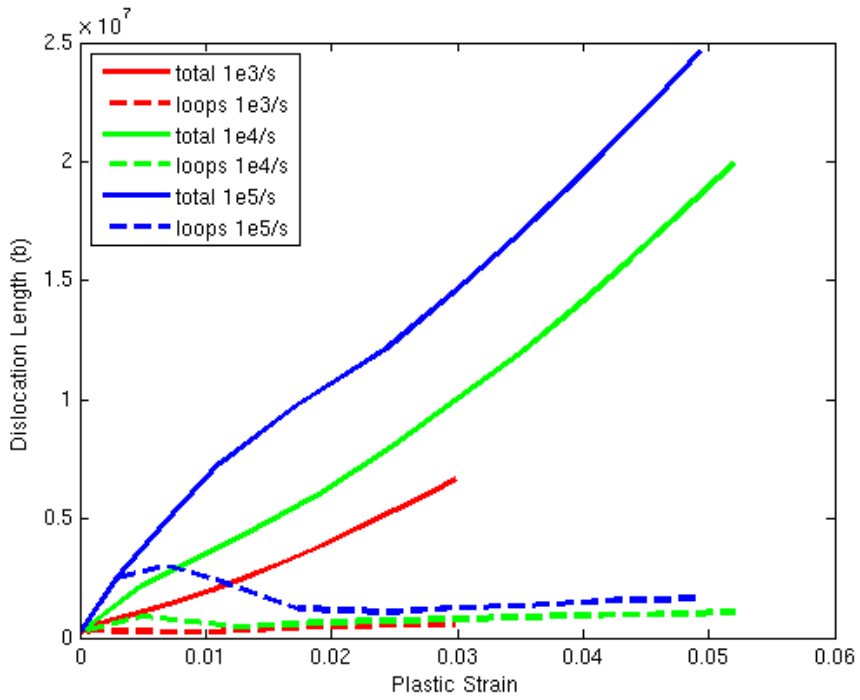

Figure 2. The evolution of total dislocation line length and the total dislocation loop lengths. The latter clearly changes slowly after the initial hump and saturates early.

the dislocation lines are broken into meta-arms connecting between M nodes. Meanwhile, as strain gets larger, the dislocation loops (mostly stand alone ones) start to form. These loops are seen as debris. It is checked that these loops contribute very little to the dislocation flux and plastic strain. Therefore, they do not play an important role for the plastic flow.

Figure 3 shows the number of M nodes and their evolution as a function of plastic strain. Figure 4 shows the total meta-arm lengths for all meta-arms in the system and Figure 5 shows the total EP distances between all pairs of M nodes. Figure 6 shows the average meta-arm length, which is obtained by dividing the total meta-arm length by the number of meta-arms. Figure 7 shows the average EP distance between all M nodes, which is obtained by dividing the total EP distance by the number of meta-arms (or pairs of M nodes). As a reference, we also plot the mean distance based on total dislocation density $1/\sqrt{\rho}$ in Figure 8.

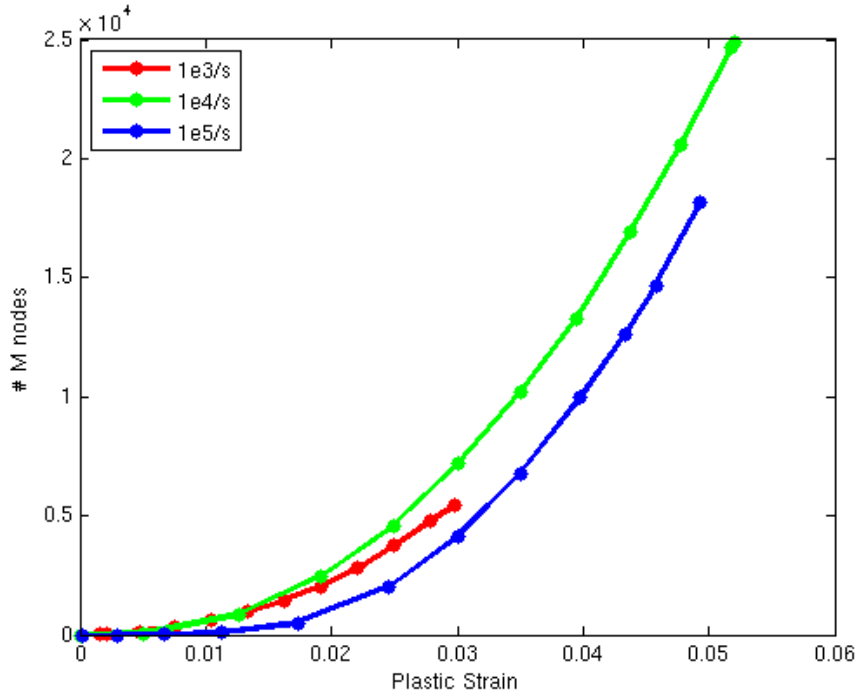

Figure 3. The number of M nodes as a function of plastic strain.

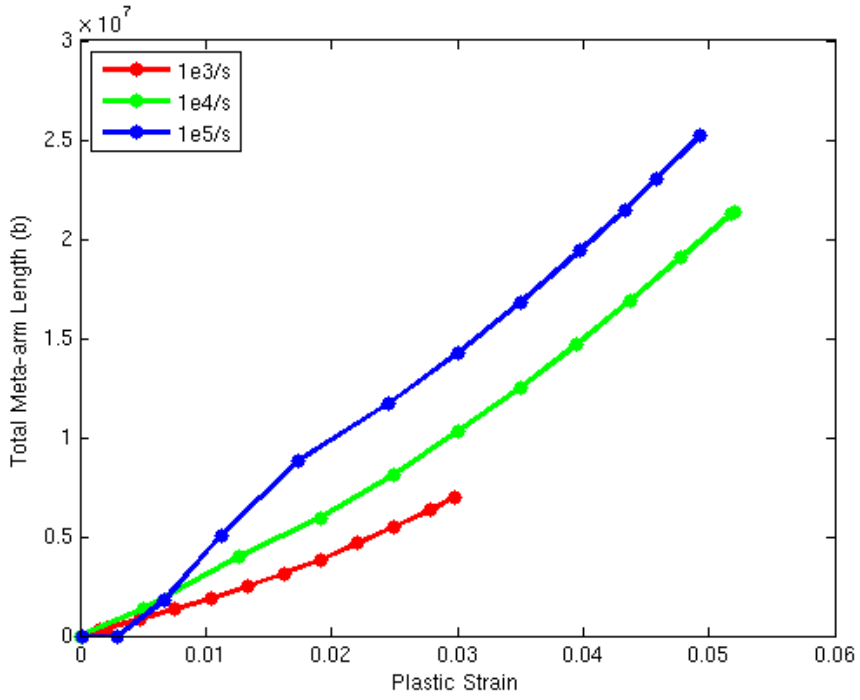

Figure 4. The total meta-arm lengths between all pairs of M nodes.

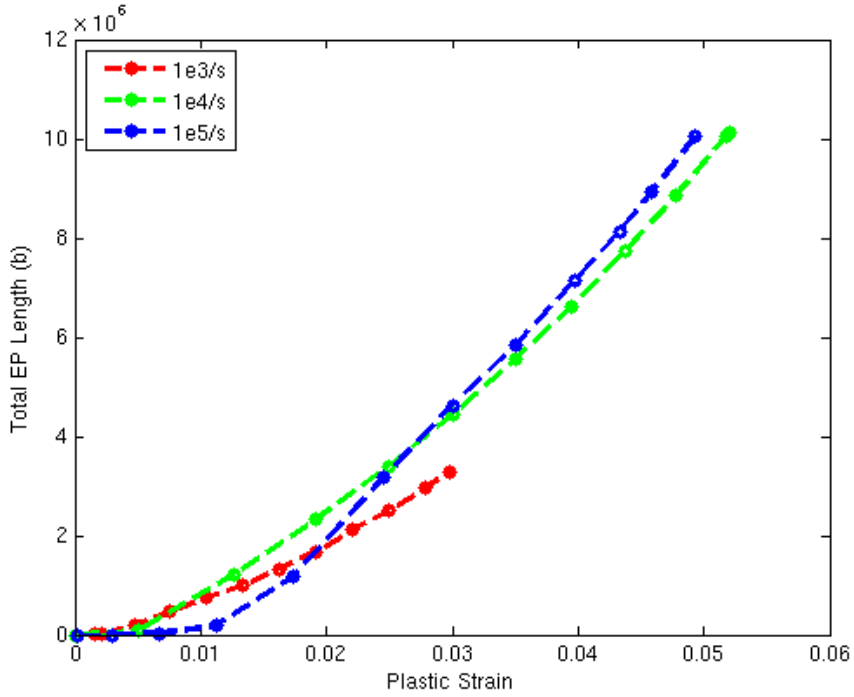

Figure 5. The total EP straight distances between all pairs of M nodes.

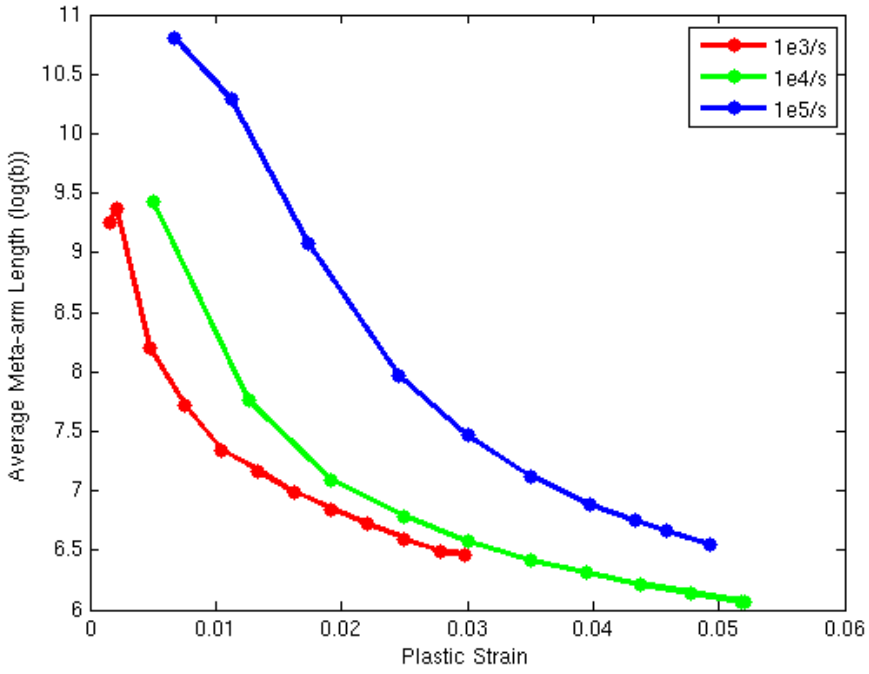

Figure 6. The average meta-arm length. The number of meta-arms are obtained directly from the output of the analysis.

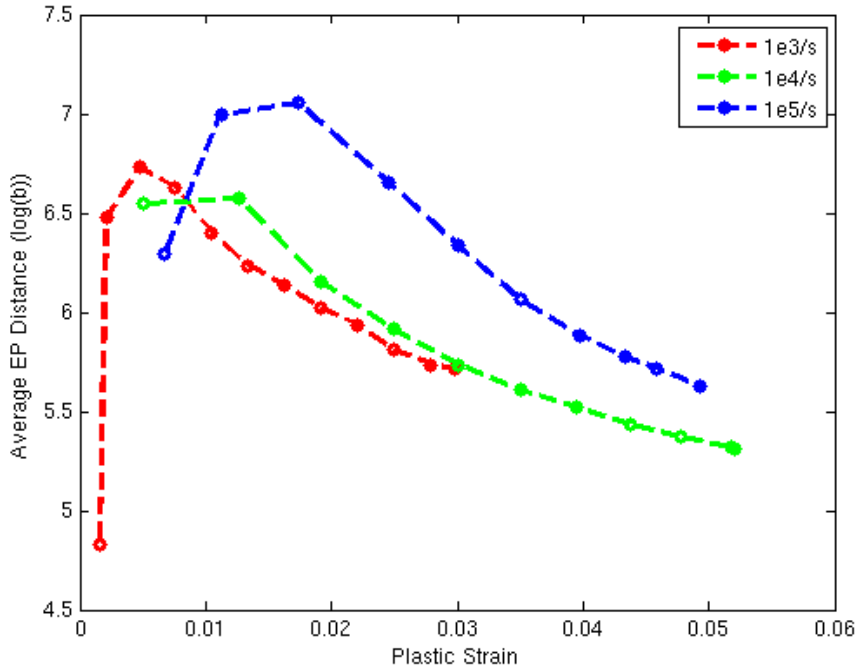

Figure 7. The average EP distance.

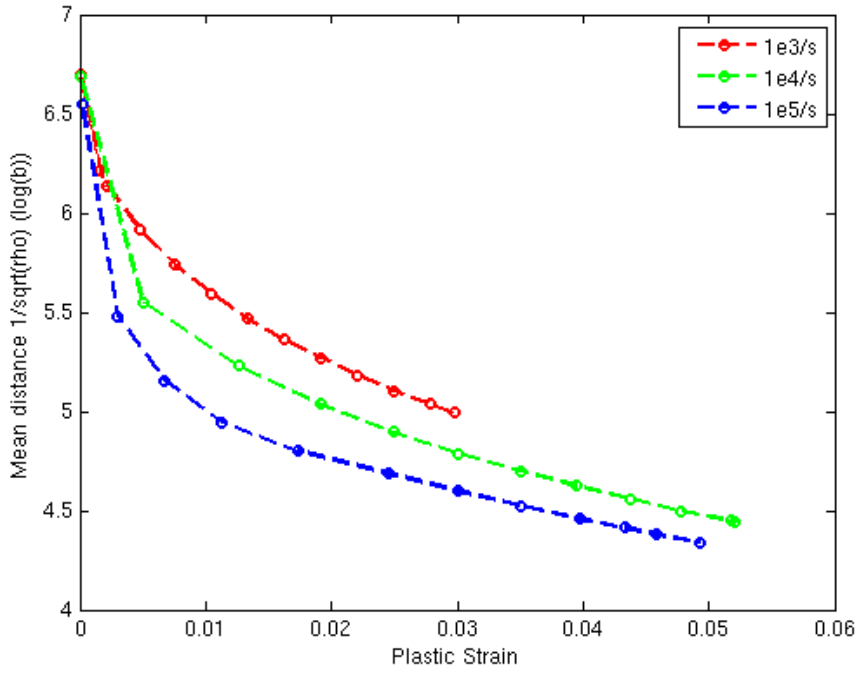

Figure 8. The mean distance between dislocations obtained by $\lambda = 1/\sqrt{\rho}$

5. Discussion and future work

The analysis algorithm developed is useful to identify dislocation multi-junction nodes in BCC single crystals and obtain the characteristic dislocation lengths between multi-junctions. Similar approach can be applied to non-BCC single crystals. The analysis algorithm is integrated into the Visit visualization tools as a routine tool to analyze dislocation configurations. See the Appendix on how to use the tools. In order to use the results obtained from such analysis, we need make connections with the constitutive models that can present a framework where the characteristic lengths can be utilized for strain hardening models.

Acknowledgements

This work is performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. M. Tang would like to thank Vasily Bulatov for discussions during the early stage of this work.

References

1. Arsenlis A, Cai W, Tang M, Rhee M, Oppelstrup T, Hommes G, Pierce T G, and Bulatov V V, 2007 *Enabling strain hardening simulations with dislocation dynamics* Modelling Simul. Mater. Sci. Eng. 15 553-595

2. Bulatov V. V., Hsiung L. L., Tang M., Arsenlis A., Bartelt M. C., Cai W, et al, 2006 *Dislocation multi-junctions and strain hardening* Nature, 440, 1174-1178.

Appendix

The ParaDIS Visualization Toolset

The ParaDIS toolset has three components.

paraDIS_lib -- the underlying library for analyzing and visualizing paraDIS dumpfiles.
analyzeParaDIS -- a command-line tool built on paraDIS_lib which outputs various files for scientific analysis and visualization
paraDIS VisIt plugin -- a reader that imports paraDIS data into the LLNL VisIt mesh visualization tool. Built on paraDIS_lib, the plugin also reads the paraDIS parallel file format which is optimized for speedy reading and visualization.

1A. paraDIS_lib -- A Library for paraDIS Dumpfiles

The paraDIS_lib library is a simple library that parses paraDIS dumpfiles and analyzes them. It is able to create statistics, detailed information, and graphics primitives about those dumpfiles. The algorithm can be found in the ReadData() function call in paradis.C. Dumpfile analysis proceeds in stages:

0. Dumpfile parsing and creation of Nodes and Segments. Nodes represent discretization nodes from the paraDIS simulation, and Segments capture neighbor relationships between the nodes as described by the dumpfile.
1. Creation of Arms. An Arm is a chain of connected Nodes and the Segments forming their connections. The Nodes in an Arm that have two neighbors are called "interior" Nodes, while any Node(s) with more than two neighbors are called "terminal Nodes." An Arm has either zero, one or two terminal Nodes. See Figures 1, 2 and 3.
2. Decomposition of higher-energy burgers vector Arms into type 111 Arms.
3. Classification of Nodes as multijunctions or non-multijunctions.
4. Classification of Arms. (See section 1B)
5. Wrapping boundary Segments to prevent visual artifacts for periodic data.
6. Discovery of MetaArms [sic]. A MetaArm is a collection of connected Arms in which all Arms share the same burgers vector and each Arm shares a terminal Node with another Arm in the MetaArm. A MetaArm, like an Arm, may have zero, one or two terminal Nodes, but not all its interior Nodes have two neighbors, as it includes multiple Arms. See Figure 4.

After the data structures described above have been created, ReadData() optionally writes out files containing statistics, as well as files that can be used for

visualization and debugging. These files are described in the analyzeParaDIS section below. An API for integration into other codes is also available in the file paradis_c_interface.h.

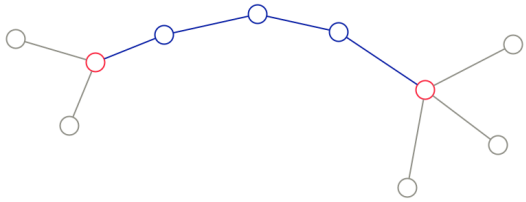

Figure 1: An arm with three interior nodes (blue circles), four segments (blue lines), and two terminal nodes (red circles). Grey segments and nodes belong to neighboring arms.

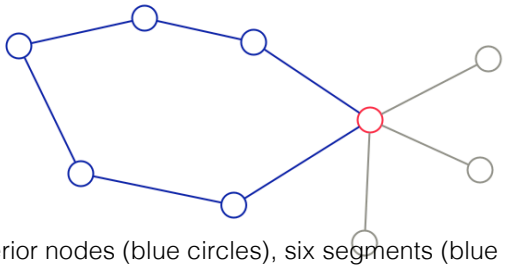

Figure 2: A looped arm with five interior nodes (blue circles), six segments (blue lines), and one terminal nodes (red circle). Grey segments and nodes belong to neighboring arms.

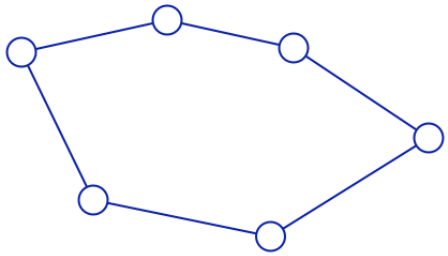

Figure 3: A looped arm with six interior nodes and six segments.

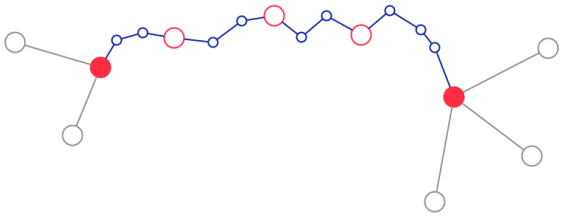

Figure 4: A meta-arm consisting of four arms. Red circles are terminal nodes for arms. Filled red circles are the terminal nodes for the meta-arm. Blue circles are interior nodes.

1B. Segment, Arm and MetaArm type definitions

The following code extracted from paradis.h shows the values and types for segment, arm and meta-arms.

// Segment BURGERS TYPES: (P = plus(+) and M = minus(-))
// These are valued in order of increasing energy levels, corresponding to the sum of the square of the components of the burgers vector.

```
BURGERS_DECOMPOSED  -2  // for segments that are decomposed
BURGERS_UNKNOWN     -1  // analysis failed
BURGERS_NONE         0  // no analysis done yet
BURGERS_PPP         10  // +++  BEGIN ENERGY LEVEL 1
BURGERS_PPM         11  // ++-
BURGERS_PMP         12  // +-+
BURGERS_PMM         13  // +--
BURGERS_200         20  // BEGIN ENERGY LEVEL 2
BURGERS_020         21
BURGERS_002         22
BURGERS_220         30  // BEGIN ENERGY LEVEL 3
BURGERS_202         31
BURGERS_022         32
BURGERS_311         40  // BEGIN ENERGY LEVEL 4
BURGERS_131         41
BURGERS_113         42
BURGERS_222         50 // BEGIN ENERGY LEVEL 5
BURGERS_004         60 // BEGIN ENERGY LEVEL 6
```

// Arm MN types:

```
ARM_EMPTY        -1 // marked for deletion after decomposition step
ARM_UNKNOWN        0
ARM_LOOP         2
ARM_MM_111         3
ARM_MN_111         4
ARM_NN_111         5
ARM_SHORT_NN_111   6 // for "Bucket" analysis.
```

// MetaArm types:

```
METAARM_UNKNOWN     0  // Not defined, error, or other odd state
METAARM_111         1  // Non-loop, type 111 arms.
METAARM_LOOP_111    2  // Loop of type 111 arms.
METAARM_LOOP_HIGH_ENERGY    3  // Loop of type 200 arms or higher.
```

2. analyzeParaDIS -- Command Line Tool

The analyzeParaDIS command line tool presents a convenient way to parse paraDIS dumpfiles and glean statistics about them. The options for analyzeParaDIS are documented by running analyzeParaDIS with the --help option. A brief example is the best way to illustrate usage of analyzeParaDIS. The easiest way to run analyzeParaDIS is with the --full option, like so:

/usr/global/tools/IMG_private/paraDIS/chaos_5_x86_64_ib/bin/analyzeParaDIS --full rs0476.data

The --full option generates basically all available outputs. Here are all the output files from the above command, placed in a subdirectory named rs-0476-output.

```
rs0476-Arms-debug.txt
rs0476-FullNodes-debug.txt
rs0476-MinimalNodes-debug.txt
rs0476-debug-before-decomp.arms
rs0476-metaarms-debug.txt
rs0476-nodes-0000.vtk
rs0476-nodes-0001.vtk
rs0476-nodes-0002.vtk
rs0476-nodes-0003.vtk
rs0476-nodes.visit
rs0476-segments-0000.vtk
rs0476-segments-0001.vtk
rs0476-segments-0002.vtk
rs0476-segments-0003.vtk
rs0476-segments-0004.vtk
rs0476-segments.visit
rs0476-tagged.data
rs0476.arms
rs0476.metaarms
rs0476.out
```

The files with “debug” in their name are detailed output for debugging issues with the library and with paraDIS itself. Files ending with “vtk” VTK files containing line and point meshes representing the nodes and segments in the dumpfile, with variables at each mesh element describing the results of the analysis. The files ending with “visit” are for viewing the VTK files in VisIt. See “Viewing paraDIS Data With VisIt” below. The file “rs0476-tagged.data” is a copy of the input file, but with the Node tags changed to indicate whether each Node is part of a looped Arm or not. This file is used in restarting paraDIS in some special mode. The files ending with “arms” and “metaarms” contain neatly formatted summaries of the Arms and MetaArms in the data set. The file rs0476.out is a very detailed output containing a blow-by-blow list of all actions taken by the algorithm. Not for the faint of heart, but can be very instructive!

In addition to these files, you can generate snapshots of particular arms, known as “trace files”, by using the --trace-arm, --trace-node, and --trace-depth options.

3. Viewing paraDIS Data With VisIt

VisIt is a custom, open-sourced mesh viewer developed originally by computer scientists at Lawrence Livermore Laboratory. There are three ways to visualize, or view, paraDIS data with VisIt.

In the "serial dumpfile" method, you open a paraDIS dumpfile directly, the paraDIS_lib function completes its lengthy analysis of the file using a single CPU thread, and VisIt then displays the results.

In the second, "parallel paraDIS dumpfile" method, you open a special paraDIS output known as a parallel data file. This is a simplified output that can be read and displayed in parallel. It is less detailed than the output of the serial dumpfile method but scales in size and speed for larger data sets.

The third method is a hybrid approach: run analyzeParaDIS with the --full or --vtk-file options, let it do its analysis, and then view the resulting .visit files in parallel using VisIt. The result is a fuller set of results and scalable visualization.

For support in viewing paraDIS data with VisIt, please contact Richard Cook at 925-423-9605.

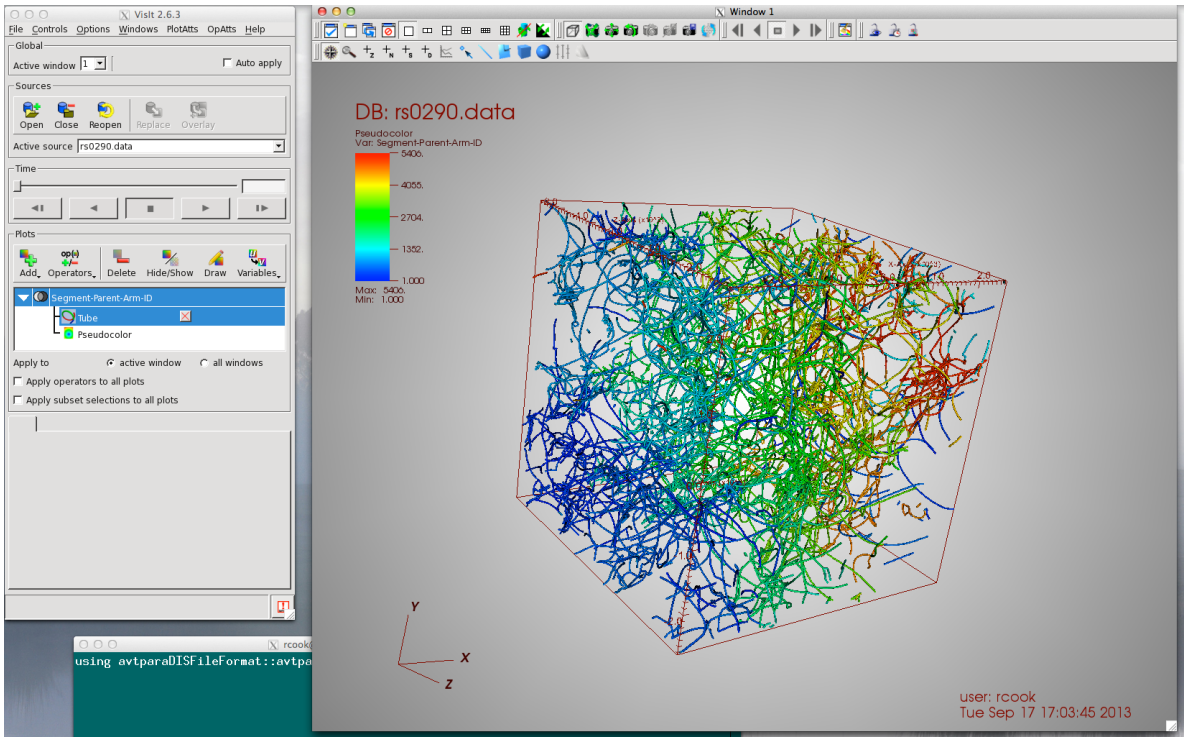

Figure 5: VisIt rendering of a 22,801 node paraDIS dumpfile, using the Tube operator and coloring by Parent Arm ID.