Computational Mechanics: HDF Parallel I/O Implementation in Warp3D

Researchers

Daniel Pledger:
University of Tennessee, Knoxville
dpledger@vols.utk.edu

Carlos Estrada:
New Mexico State University, Las Cruces
carlos1e@nmsu.edu

Rocco Febbo:
University of Tennessee, Knoxville
rfebbo@vols.utk.edu

Mentors

Timothy Truster:
University of Tennessee, Knoxville

Kwai Wong:
University of Tennessee, Knoxville

Abstract:

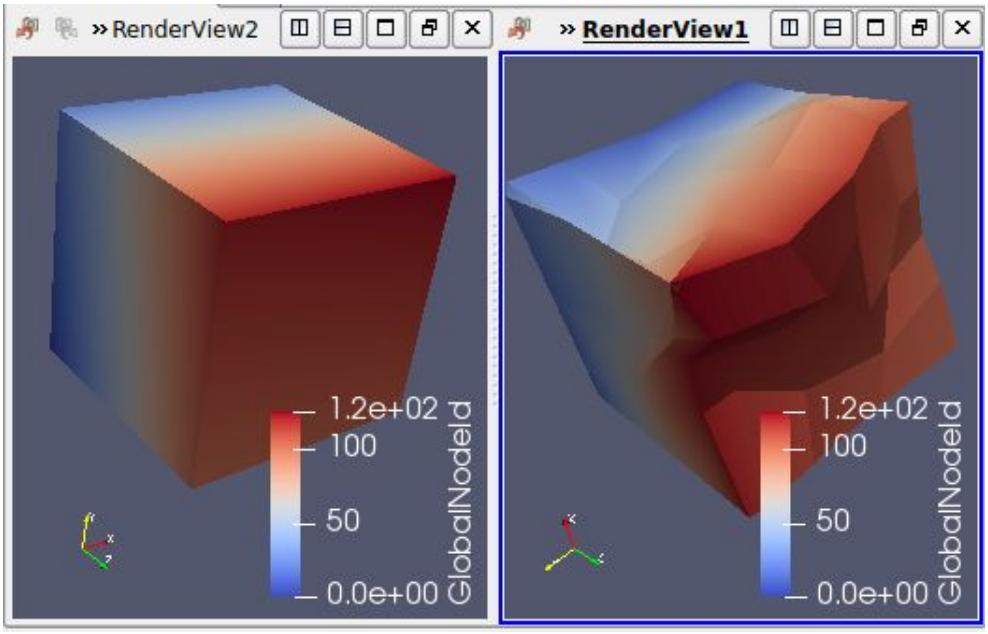
Warp3D is a code developed for the solution of large-scale, 3-D solid models subjected to static and dynamic loads, and capable of fracture and fatigue analysis. A process could be developed for producing more accurate objects to be analyzed while balancing runtime and efficiency by manipulating I/O and implementing a parallel-friendly format. HDF5 is a file format that consists of metadata and datasets and could suit as a

1. Introduction:

This report will outline the developed workflow, and programs utilized for creating a Warp3D input file with complex grain structures and boundaries. In addition, this will illuminate the source code developed for more efficient I/O in Warp3D.

1.0.1 Warp3D:

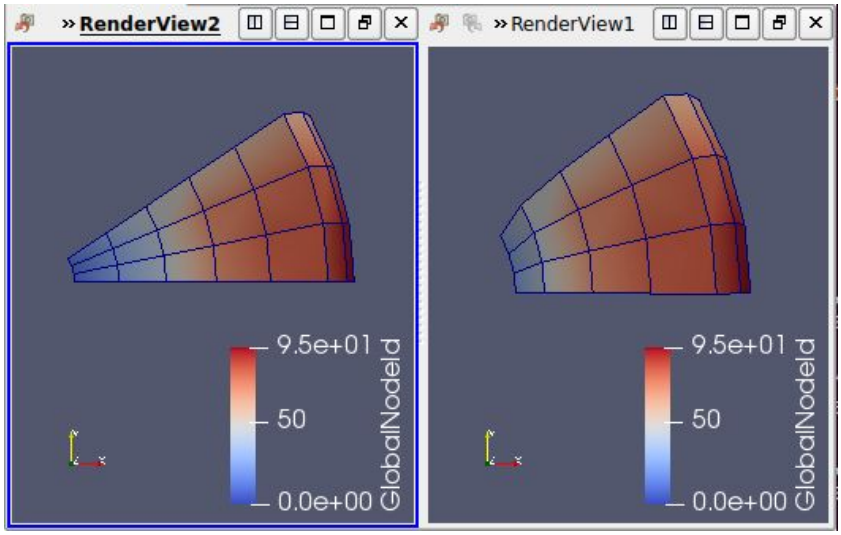
This program is the center of this research. Warp3D takes a geometrical mesh consisting of nodes and incidences/elements, material properties, and applied loads to this geometry to compute stresses, strains, displacements, temperature change, fracture points, etc. in the material. Warp3D systematically performs the physical mathematics behind these changes due to applied loads in parallel with OpenMP.



1.0.2 Paraview:

Paraview is an open-source multiple-platform application for interactive, scientific visualization. It has a client–server architecture to facilitate remote visualization of datasets, and generates level of detail models to maintain interactive frame rates for large datasets.

Paraview was used extensively in the visualization of Warp3D input and output files. XDMF allows Paraview to render from datasets in HDF5 format.



1.1 Goals:

- Construct a workflow for creating a more defined and realistic object through pre-processing software.
- Utilize HDF5 for quicker and more efficient I/O to and from Warp3D.

1.2 Objectives Fulfilled:

- The focus of the research conducted was to make use of Warp3D's full capabilities by creating a workflow for inputting more complex geometries, and maximizing efficiency by modifying output processes.
- A workflow was developed using various open source codes to create Warp3D input files containing material grain structures, and interface elements between them, to allow for the slipping and separation of grains for fracture analysis.
- Warp3D's source code was modified to output an HDF5 file containing simulation results, and write an XDMF file for visualization in Paraview.

2. Overview:

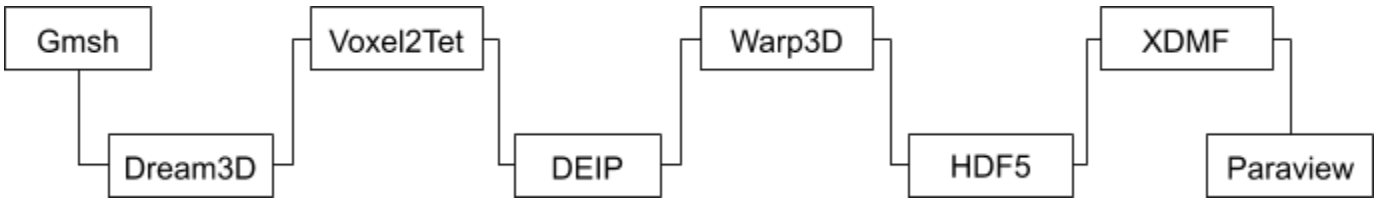


Figure 1: Outline of workflow.

Figure 1 shows a flowchart of the programs and file types used for this project. Gmsh, Dream3D, Voxel2Tet, and DEIP were used for input file creation, while Warp3D source code was written and modified for output in HDF5 and XDMF. Paraview can be used as an intermediate step between most programs to view created geometries before Warp3D input, but is ultimately used for visualizing Warp3D data outputs.

2.1 Front End:

Below are the programs used to create a more complex Warp3D input file containing grain structures. Efforts were aimed toward creating a workflow for inputting geometry models containing grain structures into Warp3D that can be reproduced for other models easily.

2.1.1 Gmsh:

Gmsh is a three-dimensional finite element mesh generator with a built in CAD engine. Gmsh was used to create the geometry used for testing and for model generation for Warp3D simulation. Gmsh can export geometry data to an STL file, commonly used for viewing CAD geometries.

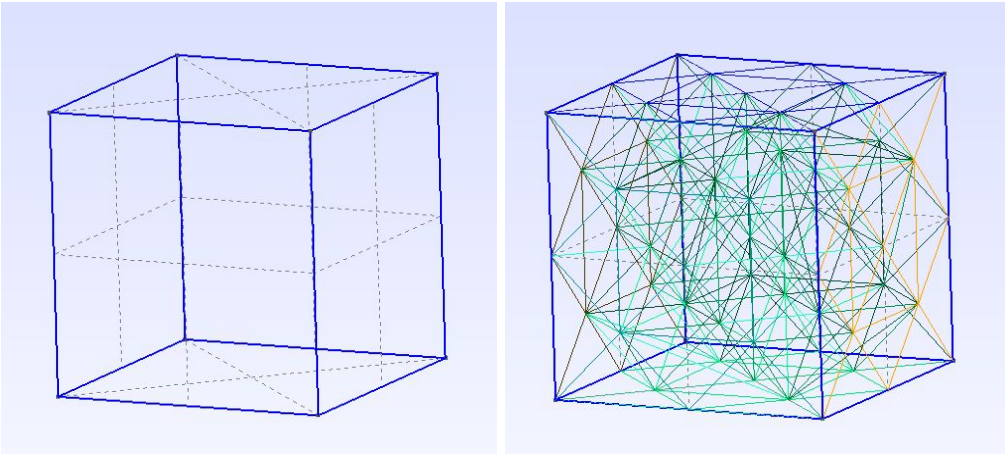


Figure 2: A simple cube created in Gmsh on the left. A three dimensional mesh of the cube on the right.

2.1.2 Dream3D:

Dream3D is an open source, computational microstructure tool. Dream3D allows you to fill a solid CAD model with microstructural grains created from statistics and properties users can input to meet their desired grain size and shape. Microstructural grains are created by constructing a “Pipeline” or Dream3D workflow. Dream3D can take an STL file created by Gmsh and create grain structures in it as shown by Figure 3.

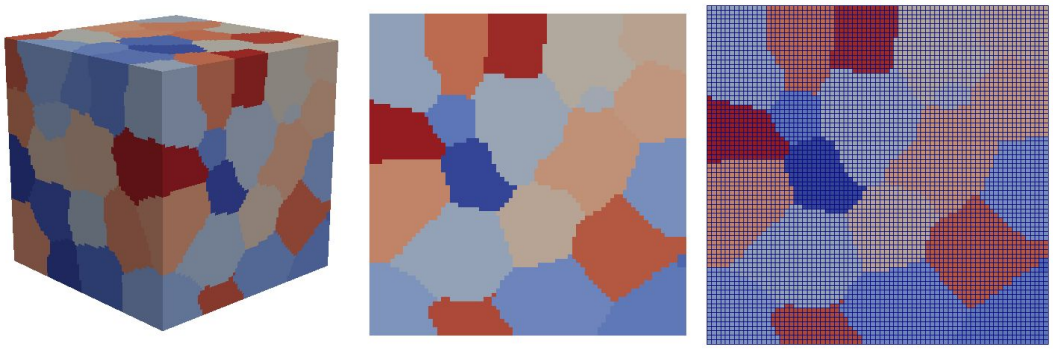


Figure 4: Microstructural grains created using Dream3D in the cube example from Figure 2.

Starting a pipeline creates a data structure describing model geometry and microstructural grains. The fifth step in the pipeline is the Statistics Generator. Statistics such as average grain size and standard deviation are input to create the grain structures. The data structure created describes grain shapes, sizes, Euler angles, and more. Dream3D can output an XDMF file for visualization in Paraview, but outputs model data in a .dream3d file format.

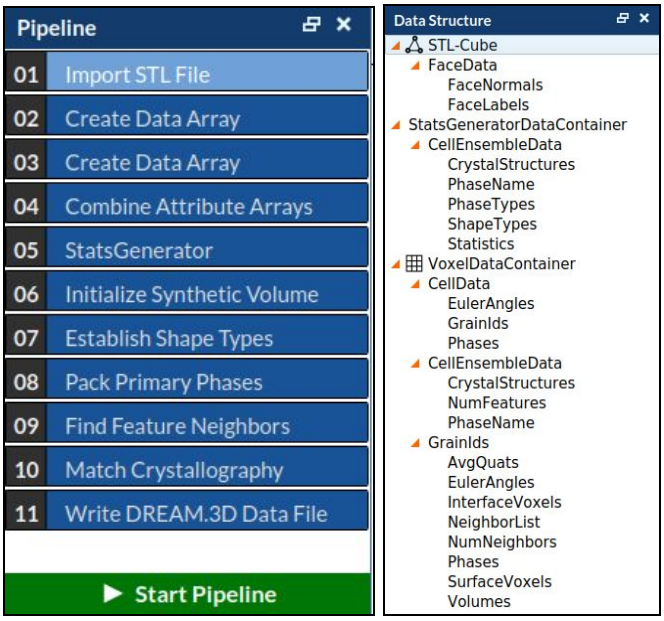


Figure 4: Dream3D pipeline and data structure.

2.1.3 Voxel2Tet:

Voxel2Tet is a code written in C++ that can take voxel (cubic) grain structures and convert them to tetrahedral mesh with smooth interfaces. Voxel2Tet was designed to read .dream3d files, but was last updated in 2016. Due to Dream3D updates, Voxel2Tet required source code modification and Dream3D grain structures to be created in a specific way to run. Grains created by Dream3D in Figure 4 were put into Voxel2Tet and are shown in Figure 4. Voxel2Tet outputs VTK files for visualization in Paraview, and Abaqus .inp files containing model geometry and data. Source code was lightly modified to read Dream3D data after updates.

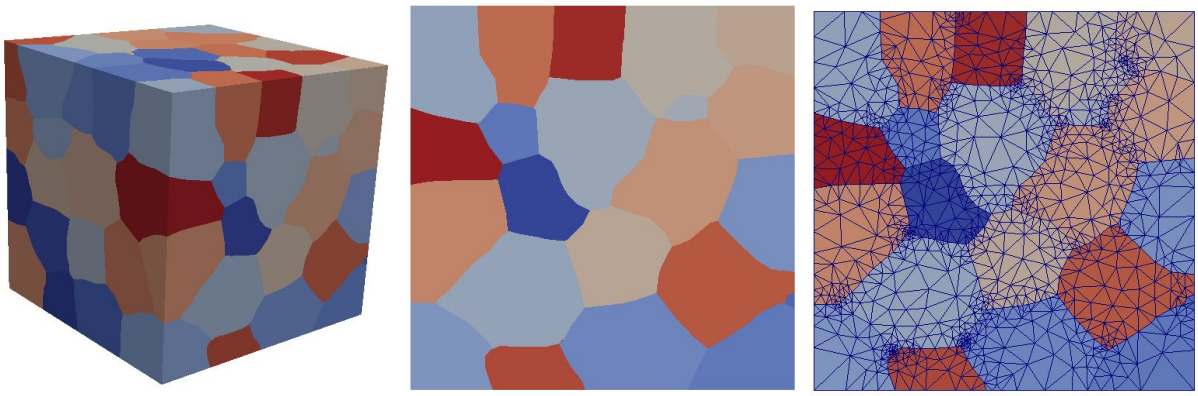


Figure 5: Dream3D grain structures modified by Voxel2Tet to have smoother interfaces.

2.1.4 Discontinuous Element Insertion Program (DEIP):

Discontinuous Element Insertion Program is a program written in MATLAB that inserts zero-thickness elements in between grain surfaces in a finite element mesh in two and three dimensions. Zero-thickness elements are useful in Warp3D simulation because they allow for grain slipping and separation during fracture and fatigue analysis. A MATLAB program was written to read the data in Voxel2Tet Abaqus .inp output file and prepare the data for input to DEIP. For more accurate model representation, the four-node tetrahedral elements were converted to ten-node quadratic tetrahedral elements (Figure 6). DEIP inserts elements for four-node tetrahedral elements are six node surface elements, and twelve-node surface elements for ten node tetrahedral (Figure 7).

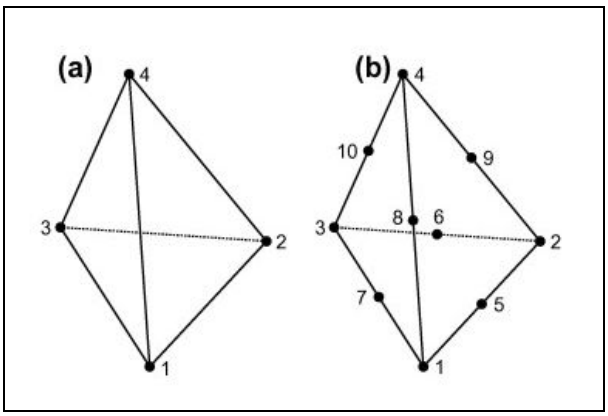


Figure 6: (a) Four node tetrahedral (b) Ten node tetrahedral interfaces

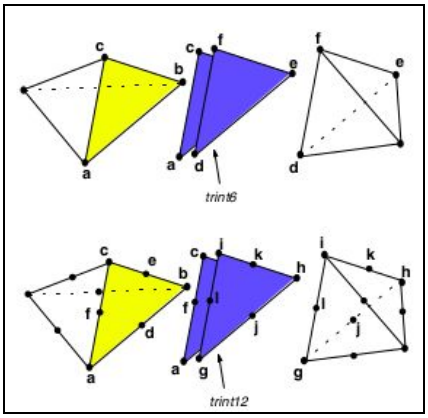


Figure 7: Six node and twelve node interfaces

DEIP has a VTK file writer for visualization in Paraview, as well as a Warp3D input file writer for easy input to Warp3D. The Warp3D input file writer was modified for the new element types. Figure 8 shows the DEIP inserted surface elements between the grain structures from Figure 5.

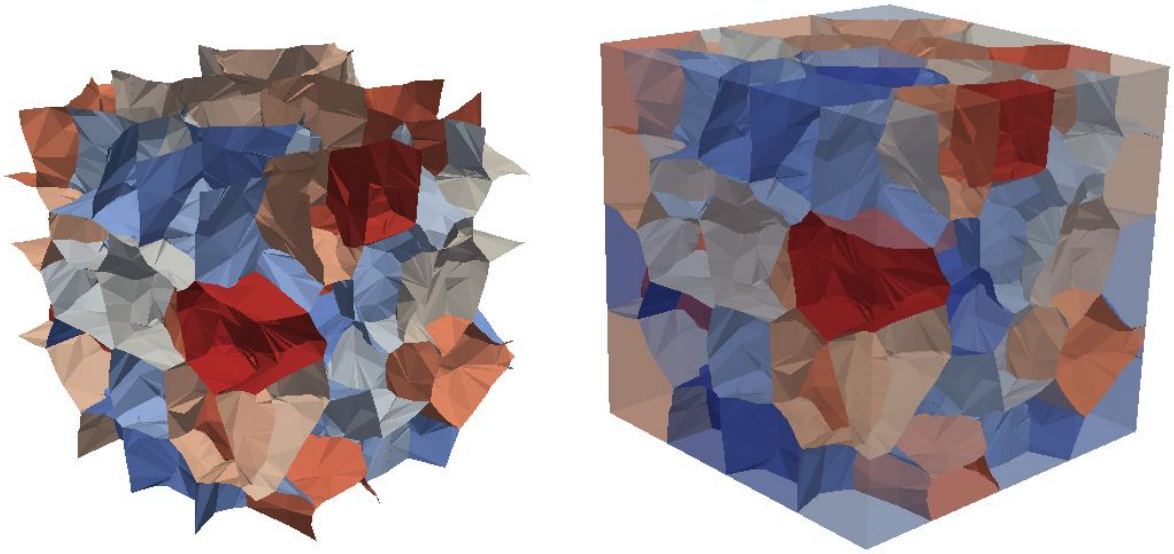


Figure 8: DEIP inserted surface elements between grains.The picture on the right illustrates the interfaces with the grains slightly visible for easier visualization.

New or modified DEIP programs:

- V2TAbaqusInputReader.m: This MATLAB program was created to read the Abaqus .inp model file created by Voxel2Tet. Similar structure to the preexisting

AbaqusInputReader.m was followed with added code to read data specific to Voxel2Tet output format.

- Tet_mesh_l2q.m: This program, written by John Burkardt was added after reading Voxel2tet output. This program converts linear tetrahedron to quadratic tetrahedron.
- S4Warp3d_qMPC.m: This program writes the input file for Warp3D. It was modified to accept quadratic tetrahedral elements and grain boundary surface elements. Modifications were also made to write element incidences on two lines to be read successfully by Warp3d. Warp3D has a character limit for each line, so incidences needed to be written on two lines to avoid this issue.
- PostParaview.m: This program was written to write a VTK file to view the model and inserted elements in Paraview. It was modified to correctly read quadratic tetrahedron and write the data to the created VTK file.

3. Source Development:

See Appendix: Source Contributions Descriptions and Processes for more detailed information on source or compilation development/ modification.

3.1 Additions to Source Code:

Additions were added to the source code of Warp3D in the form of: new subroutines, functions and variables.

3.1.1 mod_main.f:

Variables were added to global_data module:

- h5called: Boolean value used to track the opening of .h5 and .xmf files
- globname: Character array to be allocated when an HDF5 format file is named and called to receive data output

3.1.2 main_program.f:

When detecting lines from input file, if “stop” command is reached and h5called is true, call cstop() to write and close .xmf file.

3.1.3 oudriv.f:

- A catch for user input flag “hdf5” was added.
- If caught, the conditional branch calls added subroutine “oudriv_hdf5_file”.
- oudriv_hdf5_file deciphers the input line to procure a user-given file name and calls a function from ouhdf5.c, “chdf5”. The global boolean value, h5called, is then updated.

3.2 Developed Code:

Code was developed to work with the newly developed additions to create HDF5 and XDMF files for Warp3D I/O.

3.2.1 ouhdf5.c:

- Function chdf5_(...): Takes data from oudriv.f and writes .h5 and .xmf based on data given and step number computed.
- Function MakeH5Set(...): Creates an HDF5 set and fills it with given data.
- Function WriteXDMF(...): Writes XDMF format descriptor file.
- Function cstop_(...): Writes closing lines when “stop” is read to leave room for more data to be written to the .h5 and .xmf files per displacement “step”.

3.3 Manipulation of Compilation Process:

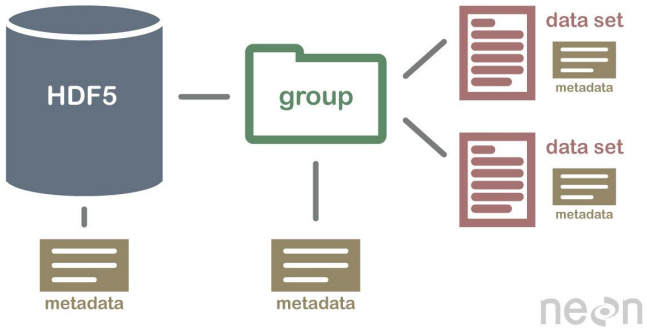
- Makefile.C.HDF5: executable created for compiling ouhdf5.c into object code and moving ouhdf5.o into the correct directory for the larger scale source compilation. Uses compiler wrapper h5cc for HDF5 libraries.
- Makewarp.bash: executable Makefile.C.HDF5 call to run before large scale compilation process.
- Makefile.linux_gfortran.omp: Changed compiler, gfortran, to wrapper, h5fc, for access to HDF5 libraries. Added ouhdf5.o to MOD object file list. Added ouhdf5.c to dependencies of object compilation.

4. Back End:

All Warp3D output contributions can be seen in ouhdf5.c and better described in Appendix: Source Contribution Descriptions and Processes.

4.1 HDF5:

- Use for parallel I/O.
- Store in HDF5 format for efficient memory sizes.
- Use metadata to identify the composition of datasets via XDMF.



4.2 XDMF:

- Descriptor file accompanying HDF5.
- .xmf describes .h5 datasets and metadata for labeling.
- Step number is appended to the name of the attribute.

Example: test87.xmf describing datasets in test87.h5

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf Version="2.0">
        <Domain>
        <Grid Name="Warp3D_1" GridType="Uniform">
                <Topology Type="Hexahedron"
                NumberOfElements ="59018">
                <DataItem Format="HDF"
                        Dimensions ="472144">
                        test87.h5:/Inc
                </DataItem>
                </Topology>
                <Geometry GeometryType="XYZ">
                <DataItem Dimensions="3 65403"
                        Format="HDF">
                        test87.h5:/Coor
```

```
                    </DataItem>
                    </Geometry>
                    <Attribute Name="GlobalNodeId" AttributeType="Scalar">
                    <DataItem Dimensions="65403"
                              Format="HDF">
                              test87.h5:/GNID
                    </DataItem>
                    </Attribute>
                    <Attribute Name="Displacement_00015" AttributeType="Vector">
                    <DataItem Dimensions="3 65403"
                              Format="HDF">
                              test87.h5:/U_00015
                    </DataItem>
                    </Attribute>
                    <Attribute Name="Load_00015" AttributeType="Scalar">
                    <DataItem Dimensions="3 65403"
                              Format="HDF">
                              test87.h5:/L_00015
                    </DataItem>
                    </Attribute>
          </Grid>
          </Domain>
</Xdmf>
```

4.3 Run Time Analysis:

Test87 (given in $WARP3D_HOME/verification):
number of nodes 65403 elements 59018

HDF5:

- No Parallel:

9.940s F
10.023s
10.386s S
10.199s
10.014s

avg: 10.1088

- Parallel (OpenMP):

10.172s
10.390s S
10.010s
10.296s
9.985s F

avg: 10.1706

Exodus:

13.984s
14.641s S

14.058s
13.975s
13.880s F

avg: 13.992

5. Conclusion:

Front end progression of realistic geometrical object brings a level of success and accuracy to Warp3D that was unrealized beforehand. Implementing grain structures into geometric models will be useful in analyzing crack propagation, material fatigue, and grain shift and separation. Accuracy will also be increased by the conversion to quadratic tetrahedron, due to the increase in node points and element description. Back end source code manipulation and development has shown a great increase in efficiency and a well-received decrease in run times. While crude and unfinished, if polished, HDF5 implementation in Warp3D I/O could evolve to be an integral aspect of large-scale projects. This could help material sciences and structural engineering to better understand and simulate objects for greater analysis in fracture points, stress displacement and much more.

5.1 Future Work:

- Compare Warp3D results for models with and without grain structures and grain boundary interfaces.
- Implement parallelism in Voxel2Tet and DEIP to lower run times.
- Test input workflow on large scale, complex geometries to find real world solutions.

Appendix:

I. Intermediate/ Non-essential Codes

- geotohdf5.c: Takes .geo (Gmsh format) and converts data to HDF5 format to be read by XDMF also created in this program.
- pattohdf5.cpp: Takes .text Patran convention form and converts data to HDF5 format to be read by XDMF also created in this program.

II. Source Contribution Descriptions and Processes

!!!!! Intro !!!!!

This is a log of all of the locations and meaning of the development that has taken place to Warp3D by JICS Computational Mechanics Team 2019

!!!!! Problems !!!!!

Created an anaconda environment that ran on Python 2 for compatibility issues

!!!!! Source and Compilation Code !!!!!

!!! mod_main.f:

Added:

globname (user specified file name for output, dynamically allocated at first hdf5 output call)

h5called (logical to track if hdf5 file has been open and written to)

!!! Makewarp.bash:

line 519: Runs Makefile.C.HDF5 before the larger scale make is processed

!!! Makefile.C.HDF5:

Compiles ouhdf5.c into object code and moves it to the large scale object-holding directory

h5cc: compiler wrapper for C to use HDF5 seamlessly

-fopenmp: flag to use OpenMP

$WARP3D_HOME == Main directory for all of warp3d distribution being used Ex. warp3d_distribution_18.0.0…

!!! Makefile.linux_gfortran.omp:

line 83: h5fc (changed from gfortran)

line 291: Tells compiler to use ouhdf5.o object code

line 379: ouhdf5.c added to dependencies in object compilation call

!!! oudriv.f:

lines 61-63: Checks for hdf5 output command

lines 400-466: New subroutine to deal with hdf5 output "oudriv_hdf5_file"

!!! ouhdf5.c:

Function called upon by oudriv_hdf5_file subroutine

Receives data from oudriv_hdf5_file (c u v a incid etc.)

Creates datasets in the user specified file name .h5

Writes XDMF descriptor file for Paraview to read HDF5 file with accuracy

!!! main_program.f:

cstop called if h5called == true after "stop" is detected

integer i and cstop added to call cstop

!!! cstop:

Function to write close of XDMF file

Inside of ouhdf5.c

called in ***main_program.f*** after "stop" is detected (passed globname)

!!!!! Executing !!!!!

Pre-Execution:

Add in input file line after desired step: output hdf5 file "filename"

pls:

Runs compilation with "help" specifications (answers to user interface 24 threads)

Runs "do"

do:

Runs Warp3D processes with "it" specifications for .text to use for .exo

do#: Just reskins of do but different test # from the test examles in $WARP3D_HOME/verification/

clean:

cleans up all the unnecessary/intermediate files to run .xmf with .h5 from Warp3D output

Pattoh5.cpp: // Independent of Source Code. Extra back end in case source HDF5 is not cooperating

Run make and run exe with argument of patran neutral .text file

Glossary:

Acronyms

I/O - **I**nput/**O**utput

HDF - **H**ierarchical **D**ata **F**ormat

DEIP - **D**iscontinuous **E**lement **I**nsertion **P**rogram

XDMF - e**X**tensible **D**ata **M**odel and **F**ormat

Program Descriptions

Dream3D - Dream3D allows you to fill a solid CAD model with microstructural grains created from user input statistics.

Voxel2Tet - Voxel2Tet converts voxel representations to tetrahedral mesh with smooth interfaces.

Gmsh - A three-dimensional finite element mesh generator with a build-in CAD engine.

Warp3D - a code developed for the solution of large-scale, 3-D solid models subjected to static and dynamic loads, and capable of fracture and fatigue analysis.

HDF5 - Hierarchical Data Format is a set of file formats designed to store and organize large amounts of data.

Paraview - An open-source multiple-platform application for interactive, scientific visualization. It has a client–server architecture to facilitate remote visualization of datasets, and generates level of detail models to maintain interactive frame rates for large datasets.

XDMF - Uses XML to store Light data (metadata) and to describe the data Model. HDF5 is used to store Heavy data. The data Format is stored redundantly in both XML and HDF5. This allows tools to parse XML to determine the resources that will be required to access the Heavy data.

References:

1. C. Geuzaine and J.-F. Remacle. *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*. International Journal for Numerical Methods in Engineering, 2009

2. Truster TJ. *Discontinuous element insertion algorithm. Advances in Engineering Software*, 2015

3. Dodds, R. (2019, July 01), et al. *WARP3D-Release 18.1.5 3-D Dynamic Nonlinear Fracture Analyses of Solids Using Parallel Computers*. University of Illinois at Urbana-Champaign

4. Sandstrom, Carl. *A Novel Tool for Converting the Voxel Representation of Microstructures ToSmooth Tetrahedral Meshes*. 2016.

5. *XDMF: Main Page*. (2017, March 13). Retrieved July 31, 2019, from http://www.xdmf.org/index.php/Main_Page