NASA/TM–2016-219180

Scalable Implementation of Finite Elements by NASA _ Implicit (ScIFEi)

James E. Warner and Geoffrey F. Bomarito
Langley Research Center, Hampton, Virginia

Gerd Heber
The HDF Group, Champaign, Illinois

Jacob D. Hochhalter
Langley Research Center, Hampton, Virginia

April 2016

NASA/TM–2016-219180

Scalable Implementation of Finite Elements by NASA _ Implicit (ScIFEi)

James E. Warner and Geoffrey F. Bomarito
Langley Research Center, Hampton, Virginia

Gerd Heber
The HDF Group, Champaign, Illinois

Jacob D. Hochhalter
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

April 2016

Acknowledgments

The development of this software was initiated through funding from the NASA Aeronautics Research Institute Seedling Fund. The NASA Langley Research Center Comprehensive Digital Transformation initiative also supported the development of the software and teaming with NASA Ames Research Center on performance tuning.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

Scalable Implementation of Finite Elements by NASA (ScIFEN) is a parallel finite element analysis code written in C++. ScIFEN is designed to provide scalable solutions to computational mechanics problems. It supports a variety of finite element types, nonlinear material models, and boundary conditions. This report provides an overview of ScIFEi ("Sci-Fi"), the implicit solid mechanics driver within ScIFEN. A description of ScIFEi's capabilities is provided, including an overview of the tools and features that accompany the software as well as a description of the input and output file formats. Results from several problems are included, demonstrating the efficiency and scalability of ScIFEi by comparing to finite element analysis using a commercial code.

1 Introduction

The Scalable Implementation of Finite Elements by NASA (ScIFEN) package is a parallel finite element analysis code written in C++. It is designed to enable scalable solutions to computational mechanics problems and supports several different finite element types, nonlinear material models, and boundary conditions. Within ScIFEN are both implicit and explicit time integration procedures called ScIFEi and ScIFEx, respectively. This report provides an overview of ScIFEi.

ScIFEi is developed with scalability and usability as the two primary design goals. It leverages several open-source high peformance computing libraries in an effort to satisfy the former, see Figure 1. The Mesh-Oriented datABase (MOAB) [1] and Hierarchical Data Format (HDF5) [2] libraries are used to enable parallel I/O operations for reading input data as well as writing simulation results. An HDF5 file is a smart data container with rich data annotations and structuring capabilities, as well as performance enhancements such as compression and parallel I/O. A frequently used analogy depicts HDF5 files as "file systems in a file" where HDF5 groups are viewed as directories and HDF5 datasets are viewed as "files" that store typed, multidimensional arrays. Like directories, HDF5 groups can be nested. Furthermore, HDF5 attributes provide a convenient mechanism for adding descriptive metadata such as comments, units, or calibrations to HDF5 groups and datasets. MOAB defines a specific schema for the finite element groups and datasets stored in HDF5.

For scalability, the Portable, Extensible Toolkit for Scientific Computation (PETSc) [3–5] provides the parallel numerical linear algebra routines used by ScIFEN. Both ScIFEi and ScIFEx utilize parallel PETSc vectors and matrices. Additionally, ScIFEi utilizes the Krylov subspace methods, parallel Newton-based nonlinear solvers, and scalable parallel preconditions [4]. ScIFEi exposes all available solver and preconditioner options offered by PETSc to its users for customization. Furthermore, PETSc supports the inclusion of additional solvers such as MUMPS [6] and SuperLU [7].

In terms of the usability design goal, ScIFEN development aims to provide user-friendly access and to enable a convenient transition from at least two popular finite

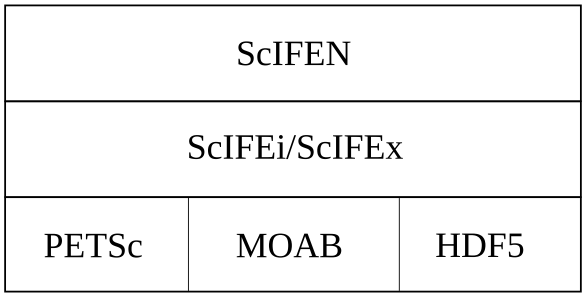

Figure 1: ScIFEN stack diagram with first-level dependencies.

element analysis codes: 1) Sierra Mechanics by Sandia National Laboratories [8] and 2) Abaqus by Simulia [9]. The model input file (containing all input data except mesh data) structure used by ScIFEN is organized similar to that of Sierra Mechanics. Hence, Sierra Mechanics users should find it easy to master the ScIFEN syntax structure. Mesh data, unlike Sierra Mechanics, is stored in a MOAB file [1]. ScIFEN also provides direct support for Abaqus user subroutines (user-defined elements and materials) and offers tools to help execute existing Abaqus models in ScIFEN. Furthermore, the ScIFEN distribution includes both a Python module and a graphical user interface for generating input files to make running analyses more convenient. Such ScIFEN tools provide a high-level interaction with the underlying libraries, *e.g.* HDF5, MOAB, and PETSc, such that users need not be familiar with them.

The goal of this report is to provide an overview of ScIFEi's features and capabilities and to demonstrate its parallel performance. A brief formulation of the analysis performed by ScIFEi is first provided in the following section. An overview of the features and tools that are included in the ScIFEi distribution is given next. The following sections describe ScIFEi input and output files. Finally, the results of a parallel performance study are discussed before the document is concluded in the summary section.

2 Implicit Finite Element Formulation

This section provides an overview of the formulation and solution of the governing equations for ScIFEi. The resulting system of equations is presented and the solution procedure is briefly discussed. The presentation is intentionally kept brief and the reader is referred to [10] for more details of the finite element formulation and PETSc [3] for the solution of the resulting nonlinear system of equations.

ScIFEi assembles and PETSc solves the nonlinear equilibrium equations governing a three-dimensional solid under a specified load where inertial forces are negligible. The system of the governing equations after employing the principle of virtual work and a suitable finite element discretization are written as

$$\mathbf{F}(\mathbf{u}) = \mathbf{P} - \mathbf{I}(\mathbf{u}) = \mathbf{0} \tag{1}$$

where $\mathbf{F}$ is the force imbalance, $\mathbf{u}$ is the displacement vector, and the external ($\mathbf{P}$)

and internal ($\mathbf{I}$) force vectors are given by

$$\mathbf{P} = \int [\mathbf{N}]^T \boldsymbol{\tau} d\Gamma + \int [\mathbf{N}]^T \mathbf{b} dV \tag{2}$$

$$\mathbf{I}(\mathbf{u}) = \int [\mathbf{B}]^T \boldsymbol{\sigma}(\mathbf{u}) dV \tag{3}$$

where $[\mathbf{N}]$ and $[\mathbf{B}]$ are the matrix of finite element shape functions and their spatial derivatives, respectively, $\boldsymbol{\tau}$ is an applied traction, $\boldsymbol{b}$ is a volumetric body force, and $\boldsymbol{\sigma}$ is stress. Γ denotes the portion of the surface of volume, V, with an applied traction. Note that here the assembly over elemental quantities is implied and it is has been assumed that the system nonlinearity is a result of history-/displacement-dependent material models.

ScIFEi uses the PETSc implementation of Newton's method (see the SNES nonlinear solvers in the PETSc documentation [3]) to solve the nonlinear finite element equations $\mathbf{F}(\mathbf{u}) = \mathbf{0}$. Starting from an initial guess $\mathbf{u}^0$, Newton's method progresses from iteration $\mathbf{k}$ to $\mathbf{k+1}$ as follows:

$$[\mathbf{J}(\mathbf{u}^k)]\Delta\mathbf{u}^{k+1} = -\mathbf{F}(\mathbf{u}^k) \tag{4}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \Delta\mathbf{u}^{k+1} \tag{5}$$

where $[\mathbf{J}]$ is the Jacobian of $\mathbf{F}$, given by

$$\begin{aligned} [\mathbf{J}(\mathbf{u})] &= \frac{\partial \mathbf{F}}{\partial \mathbf{u}} \\ &= \int [\mathbf{B}]^T [\mathbf{L}(\mathbf{u})][\mathbf{B}] dV \end{aligned} \tag{6}$$

Here, $[\mathbf{L}(\mathbf{u})]$ is dependent on the material model. At each time increment, Newton's method (Equations 4 and 5) is iterated on until a convergence criterion is met, indicating that the system is in equilibrium.

3 Features and Tools

The ScIFEN source code is distributed with several accompanying features and tools intended to make it easier and more convenient for users to carry out their analyses. This section provides a brief and broad overview of some of these tools. The interested reader is encouraged to consult the full user guide included with the ScIFEN distribution for more in-depth coverage, including tutorial examples. Below is a summary of tools to assist in building, running, and pre- and post-processing results with ScIFEN.

CMake Build System

CMake [11] is a cross-platform, open-source build system that can be used to manage the software compilation process. It automatically generates appropriate makefiles and can greatly simplify compiling complex software projects with multiple subdirectories and external library dependencies. ScIFEN uses CMake to increase its portability and simplify the build process for users. After installing ScIFEi's dependencies (PETSc, HDF5, etc.), the user is only responsible for modifying one configuration file with appropriate variables for their environment. Complete build instructions for both ScIFEi's dependent libraries and running CMake are included in the user guide.

Regression Test Suite

The ScIFEN source code comes with a suite of example problems and regression test scripts both for ongoing software development and for users to ensure their ScIFEN build is working properly. At the time of publication, the test suite includes 34 problems that test a majority of ScIFEi's capabilities (boundary conditions, element types, material models, etc.), while new tests are continually being added as development progresses. The example problems in the suite may also serve as a tool to help new users learn the ScIFEN input file structure.

`scifenpy` Python Module

To simplify the preprocessing of ScIFEN models and postprocessing of simulation results, the ScIFEN distribution has an accompanying Python module called `scifenpy`. `scifenpy` is built on the Python module `h5py` for manipulating the HDF5 and MOAB (which is HDF5-based) files that are used to store ScIFEN input and output. It includes a submodule `scifenpy.sierrahdf5` (referring to Sierra Mechanics) that is used to generate the model input files. All ScIFEN input file commands and data have accompanying functions in `sierrahdf5` to prescribe them and are documented in the user guide. An additional submodule `scifenpy.moab` is included for pre- and post-processing of the MOAB mesh and results data. The `scifenpy` module provides several other capabilities such as simple model checks to ensure proper specification and querying simulation results at particular mesh entities.

ScIFEN GUI

For those ScIFEN users with less programming experience, a graphical user interface (GUI) is included in the code distribution as an alternative to the `scifenpy` module for generating ScIFEN input files. The GUI offers a slightly reduced set of functionality in return for the added convenience, but can still be used to perform a majority of common analyses. The GUI can be used to generate new ScIFEN input files or to view/modify the contents of existing files.

Importing & Using Abaqus Models

As a component of the ScIFEN development goal for improved user-accessibility, direct support is provided for Abaqus users who are interested in running analyses with ScIFEN. Note that although SciFEi only supports a subset of Abaqus' capabilities, a user may find ScIFEN's improved computational efficiency helpful for large-scale analyses (see Section 6). ScIFEN offers a capability for translating an existing Abaqus finite element mesh into a ScIFEN-compatible MOAB mesh using the *Write ScIFEN* Abaqus plug-in included in the source code distribution. ScIFEN also provides support for Abaqus user subroutines so that researchers developing material models via (V)UMATs or finite element formulations via (V)UELs can easily run an analysis with ScIFEN.

4 Input Files

There are two input files required for any ScIFEN simulation, a model input file and a mesh input file. The input files are organized with the model (light) data in a file, which references the mesh (heavy) data stored in a separate MOAB file. ScIFEN model input files are HDF5 files, which are organized similarly to Sierra Mechanics in terms of structure and nomenclature [8]. Model data consists of material model parameters, boundary conditions, results specifications, and any other requisite input other than mesh geometry and topology.

In addition to defining a common ground for organizing finite element mesh data in a HDF5 file in an optimized manner, MOAB can store structured and unstructured mesh data and provides a library which wraps the HDF5 library for automating the reading, writing, and querying of mesh data. The MOAB data model consists of entities, entity sets, and tags. Entities are the basic topological descriptors, such as vertices, edges, faces, and volumes. Entity sets are a collection of those entities' objects. Tags are a mechanism for attaching an arbitrary dataset to entities or entity sets, which have a user-defined name, size, and type. Furthermore, MOAB includes a utility called `mbconvert`, which allows translation to and from other finite element mesh file formats. The mesh input files read by ScIFEN are expected to abide by the MOAB specifications, which are documented elsewhere [1] and are not repeated here for brevity.

The concept of separating the model data from the mesh data provides an important performance enhancement when creating and analyzing models in ScIFEN. ScIFEN can read and write large contiguous datasets stored in MOAB to achieve improved parallel I/O performance. The physical complexities attached to the finite element mesh are then described in the light data portion, which is relatively small and easily handled.

ScIFEN, in a fashion similar to Sierra Mechanics, uses the concept of "scope" to group similar input commands. Scope is implemented in ScIFEN model input files via HDF5 groups. Data with global scope (*i.e.* that can be referred to from anywhere within the file) are placed within the root group of the model input file. Information such as functions or material definitions that could be relevant to both ScIFEi and ScIFEx analyses are defined at the root level. The ScIFEN distribution is accompanied by both a Python module (`scifenpy.sierrahdf5`) and a GUI to assist in generating the model input file.

Figure 2 illustrates the hierarchical scheduling of a ScIFEN simulation. A single

ScIFEi or ScIFEx analysis from start (t_0) to termination time (t_f) is referred to as a *procedure*. Each *procedure* consists of *regions* and *time blocks*. As shown in Figure 2, multiple regions can be nested within a procedure, while multiple time blocks can occur within each region. Furthermore, multiple *time blocks* can be used in a region to turn on and off boundary conditions and adjust time incrementation within a particular time step. Note that while Figure 2 depicts the general case, a single *region* and *time block* is often sufficient to fully define a ScIFEN analysis.

Figure 3 shows an open ScIFEN model input file, scifen_model_input.h5, in HDFView [12]. The root level of the file is shown at the top of the image, with the SCIFEI PROCEDURE and FINITE ELEMENT MODEL groups expanded at the bottom of the image. The expanded SCIFEI PROCEDURE group illustrates the use of multiple *procedures*, *e.g.*, Procedure-1 and Procedure-2. Procedure-1 is expanded and consists of a SCIFEI REGION group and TIME CONTROL group. Each region, *e.g.*, Region-1, contains a referenced finite element model (in the USE FINITE ELEMENT MODEL dataset), boundary conditions (*e.g.*, FIXED DISPLACEMENT and TRACTION), SOLVER specifications, and RESULTS OUTPUT. The TIME CONTROL group contains multiple *time blocks*.

The USE FINITE ELEMENT MODEL dataset in Procedure-1 references Model-1 in the FINITE ELEMENT MODEL group. Within Model-1, the MOAB mesh input file is specified in the DATABASE NAME dataset. The PARAMETERS FOR BLOCK subgroup of Model-1 can contain multiple *element blocks*, *e.g.*, Elem_Block-1 and Elem_Block-2, each of which attach a MATERIAL and SECTION to a list of ELEMENT SETS. Finally, Figure 3 illustrates that the definitions for the referenced MATERIAL, Material-1, and SECTION, Section-1, are defined within the root groups, PROPERTY SPECIFICATION FOR MATERIAL and SECTION, respectively. Each referenced MATERIAL defines necessary material model and parameter data and each SECTION specifies the required finite element integration and formulation types. Further discussion of the available material models and element formulation and integration options are beyond the scope of this paper and the reader is referred to the user guide available through the ScIFEN code release.

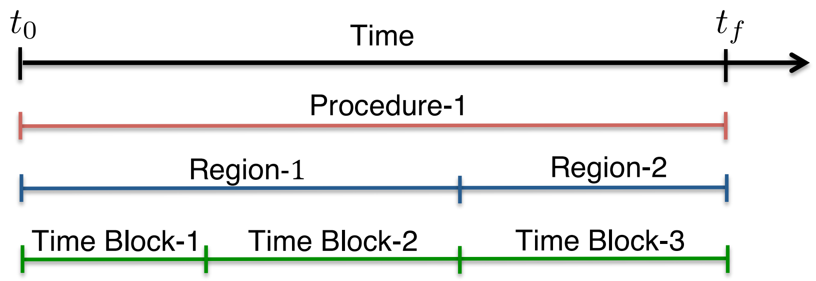

Figure 2: Diagram of a typical ScIFEN simulation.

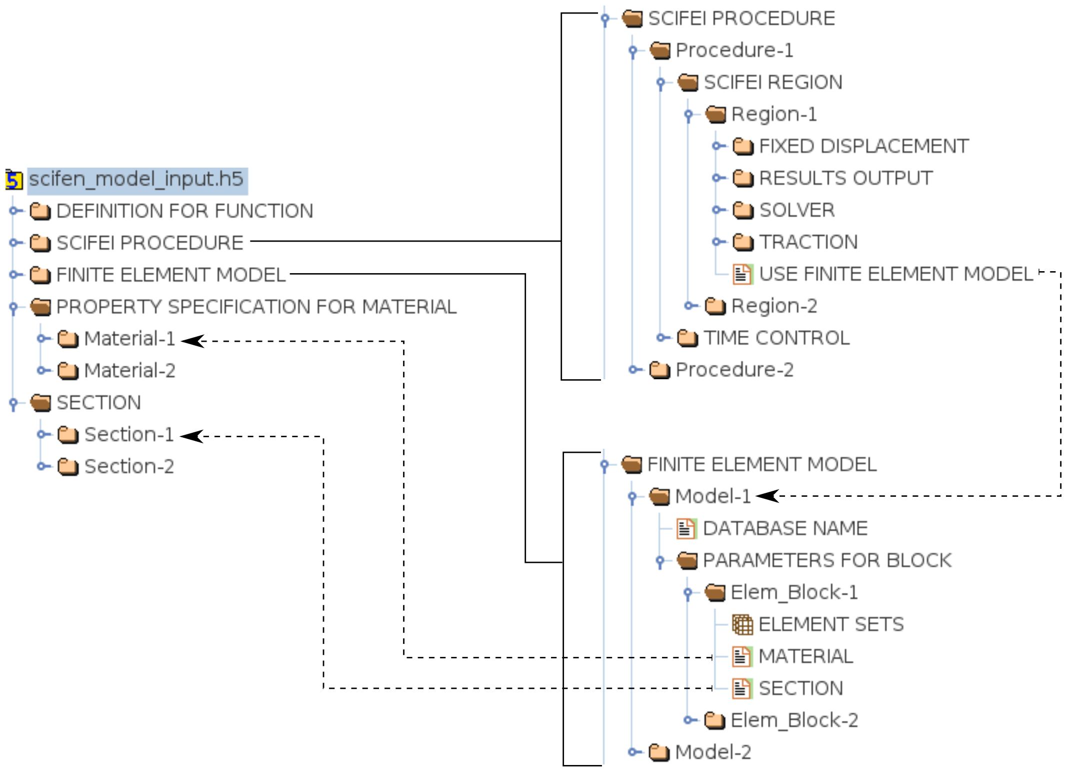

Figure 3: ScIFEi model input file example. Solid lines indicate group expansion. Dashed lines indicate groups referenced by a dataset.

5 Output and Visualization

ScIFEN output files are formatted using MOAB, as was discussed in Section 4. Computed results that are specified for output are added as tags to the output MOAB file. Since the finite element mesh is included in the output MOAB file it can also be used as an input mesh file. Also, there are mechanisms for visualization of MOAB files within the VisIt [13] and ParaView [14] open source visualization and analysis tools.

For additional flexibility, the MOAB output file can be used within the eXtensible Data Model and Format (XDMF) concept [15]. XDMF is a library which distinguishes the metadata (light data) and the values themselves (heavy data). An XML file serves as a mechanism to explicitly define light data or to reference a HDF5 file storing the heavy data [15]. ParaView, VisIt and EnSight [16] visualization programs are able to read XDMF. The XDMF format provides the ability to customize the data being used for visualization by manipulating the XML file. The `scifenpy` module includes a method for automatically generating the XML file from an existing MOAB file.

6 Comparison and Scalability

This section discusses the results of testing parallel performance of ScIFEi in two separate studies. First, the amount of time required to complete simulations of various sizes on a workstation with shared memory is compared between ScIFEi and Abaqus. The motivation of the comparison with Abaqus is to serve as a point of reference for parallel performance of an existing commercial finite element code; no comparisons are implied regarding breadth of capability. Next, the ability of ScIFEi to take advantage of a distributed memory supercomputer to solve an otherwise computationally-intractable problem is illustrated using the NASA supercomputer, Pleiades [17]. In the second study, no reasonable comparisons can be made to Abaqus (or another commercial finite element code) since it does not currently take full advantage of modern computing platforms.

6.1 Workstation Parallel Performance

The first benchmarking test was performed in order to investigate the performance and scalability of ScIFEi in a workstation environment. The study was performed on a 32 cpu core workstation (eight Quad-Core AMD Opteron 8378 processors) with shared memory. In the study, a 0.01 [m.] displacement was applied to a face of a unit cube while fixed on the opposite face. The boundary conditions are summarized by:

$$\begin{aligned} u_x(x=0) &= 0.0 \\ u_y(x=0, y=0, z=0) &= 0.0 \\ u_y(x=0, y=0, z=1) &= 0.0 \\ u_z(x=0, y=0, z=0) &= 0.0 \\ u_x(x=1) &= 0.01, \end{aligned} \tag{7}$$

where $u_i(j=k)$ is the applied displacement, in meters, in the i direction applied to all nodes where the j coordinate is equal to k [m.]. The cube was modeled using a linear elastic material with Young's modulus and Poisson coefficient of 100 [Pa.] and 0.3, respectively. One time step was taken, such that the governing system of equations was solved only once.

The test was performed using 10 finite element discretizations (meshes) of the cube where the resulting number of degrees of freedom (dof) were approximately 50,000; 100,000; ... 500,000. Each mesh was constructed using the same finite element type: linear tetrahedra. The 10 meshes combined with the above boundary conditions result in 10 finite element models, which will henceforth be referred to by their approximate number of dof (*e.g.* 50K dof). To illustrate the parallel properties of ScIFEi and compare this to Abaqus, each finite element model is solved using both ScIFEi and Abaqus with a range of cpu power, from 1 to 16 cores. The result is 160 unique finite element simulations for both codes. Finally, to minimize any effects of machine load and to maximize accuracy in simulation timing, each reported timing value is the average of 10 runs of the same simulation. The resulting test array provides insight into the performance and scaling properties of ScIFEi as it pertains to parallelization and problem size. To the best possible degree, the Abaqus simulations are replicas of the ScIFEi simulations, including iterative solver type, solver tolerances, and output variables.

For verification of the ScIFEi code and to ensure that the iterative solvers of ScIFEi and Abaqus were solving the system of equations to a similar degree of accuracy, the relative differences between nodal displacements ($\mathbf{u}$) and element stresses ($\boldsymbol{\sigma}$) computed by each code were calculated. Table 1 shows these relative differences, defined by

$$\Delta\mathbf{u} = \frac{\|\mathbf{u}_{\text{ScIFEi}} - \mathbf{u}_{\text{Abaqus}}\|_2}{\|\mathbf{u}_{\text{Abaqus}}\|_2} \tag{8}$$

$$\Delta\boldsymbol{\sigma} = \frac{\|\boldsymbol{\sigma}_{\text{ScIFEi}} - \boldsymbol{\sigma}_{\text{Abaqus}}\|_2}{\|\boldsymbol{\sigma}_{\text{Abaqus}}\|_2}, \tag{9}$$

for each of the 10 finite element discretizations. Good agreement is observed between the displacements and stresses computed by ScIFEi and Abaqus, with $\Delta\mathbf{u} < 5.0\text{e}{-5}$ and $\Delta\boldsymbol{\sigma} < 8.2\text{e}{-6}$ for all discretizations considered.

Figure 4a illustrates the wall clock time taken for ScIFEi and Abaqus simulations of two finite element models (differing numbers of dof) as a function of the number of cpu cores used. A typical scaling curve is seen in which more processing power corresponds to faster computation. By comparison, the ScIFEi simulations are notably faster than the Abaqus simulations. As shown in Figure 4b, the ScIFEi simulations are approximately 1.5 to 3 times faster than the Abaqus simulations, depending on the finite element model size and number of cpu cores used. This trend is, in part, due to the fact that ScIFEi is a specialized code which allows it to have less overhead, but also demonstrates the efficacy of the PETSc software package. It should also be noted that the Abaqus simulation times do not include the license checkout time but rather only the time taken from input parsing through final output.

A strong scaling plot is shown in Figure 5a which illustrates how many times faster (speed up) a finite element model runs on multiple cpu cores than the same finite element model on a single core. Notably, ScIFEi scales better than Abaqus in general, especially when using more cpu cores. Figure 5b illustrates the improved strong scaling of the PETSc solver relative to the Abaqus solver. This indicates that, in models dominated by solver time (*e.g.*, more dof and/or time increments), the advantage of using ScIFEi will become more pronounced as the number of cpu

	50k	100k	150k	200k	250k
$\Delta\mathbf{u}$	1.33e-6	3.45e-6	4.94e-6	8.88e-6	1.97e-5
$\Delta\boldsymbol{\sigma}$	2.42e-6	2.55e-6	3.44e-6	3.76e-6	4.14e-6
	300k	350k	400k	450k	500k
$\Delta\mathbf{u}$	1.91e-5	3.23e-5	3.90e-5	1.75e-5	4.96e-5
$\Delta\boldsymbol{\sigma}$	5.10e-6	8.06e-6	4.80e-6	7.76e-6	8.15e-6

Table 1: Relative differences between displacements and stresses (Equations 8 and 9) calculated by ScIFEi and Abaqus.

cores increases. Here, even for the largest model, Abaqus speedup plateaus at 12 cpu cores.

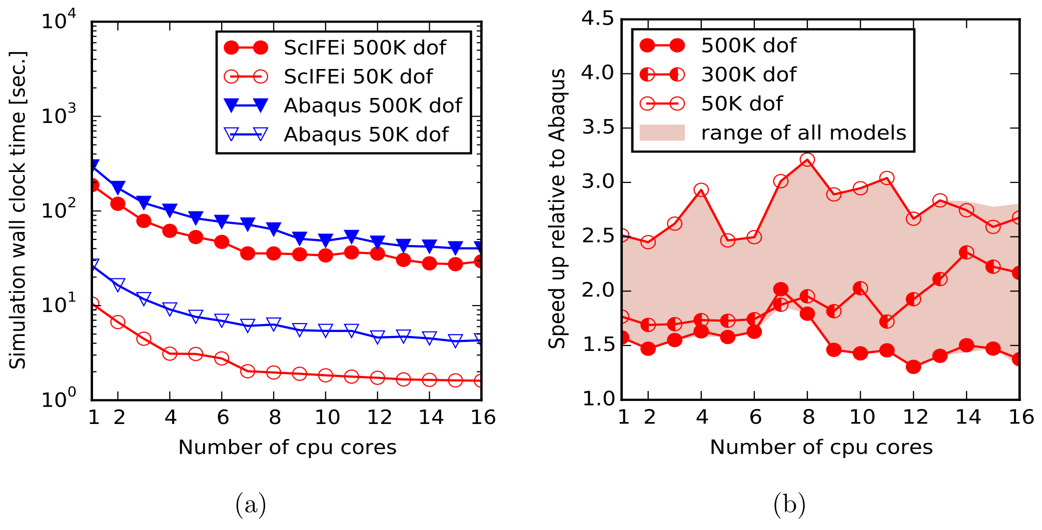

Figure 4: Comparison of ScIFEi and Abaqus performance. (a) Wall clock time for two example finite element models and (b) the speed up of a ScIFEi simulation when compared to the same simulation performed with Abaqus.

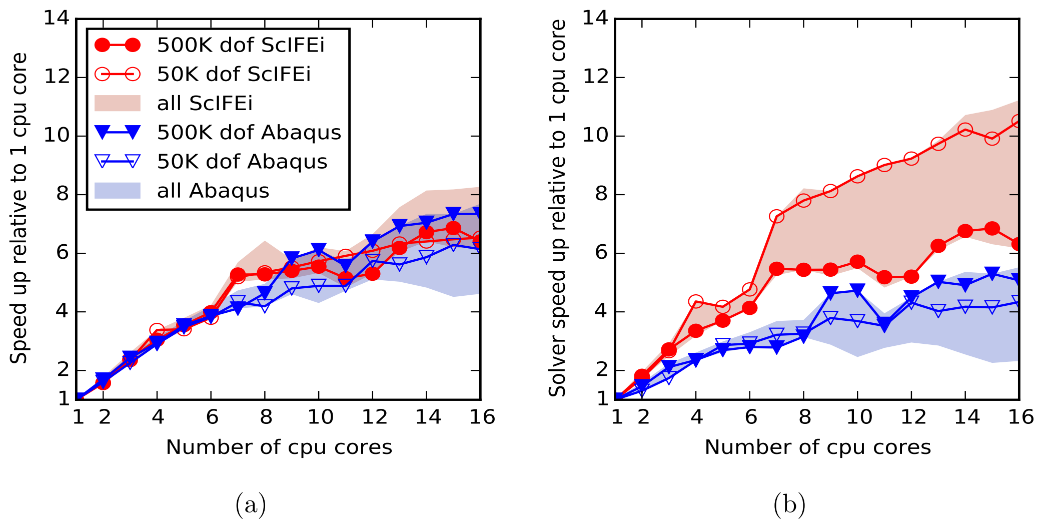

Figure 5: Strong scaling for ScIFEi and Abaqus. The scaling is calculated on (a) the total wall clock time and (b) solver wall clock time.

6.2 Supercomputer Parallel Performance

The motivation in developing a parallel finite element code, like ScIFEN, is to take advantage of modern supercomputer architectures to solve otherwise computationally-intractable problems. The NASA supercomputer, Pleiades [17], was used here to demonstrate this capability. Pleiades compute nodes contain two ten-core Intel Xeon E5-2680v2 processors which communicate over an InfiniBand® network. The parallel file input/output is enabled by a Lustre® filesystem.

This section serves to demonstrate the capability of ScIFEi in computing solutions for necessarily-large models and to quantify the current level of scalability for such models. Spear *et al.* recently generated one such model of a 3-D volumetric finite element mesh obtained by using near-field high-energy X-ray diffraction microscopy (nf-HEDM) data [18]. The nf-HEDM results provided a high-resolution (approx. 2 [μm.]) spatial quantification of the microstructural features, grain morphology, immediately surrounding a crack in an Al-Mg-Si alloy. From a mesh-convergence study, it was determined that the characteristic element sizes could be coarsened from the nf-HEDM provided 2 [μm.] size to 6 [μm.], while maintaining converged solutions in the areas of interest.

Figure 6 is an illustration of the concurrent multiscale mesh used for these parallel performance trials. The regions representing the macroscale specimen and the microscopic grains were represented with a linear elastic material model, where for each region a Young's modulus was randomly sampled from a normal distribution ranging from 65-75 [GPa.]. A global strain of 1% was applied in uniaxial tension to the top face of the macroscale model, while the bottom face was held fixed, see Figure 6. The microscale finite element mesh was embedded at the notch root of the macroscale model, see Figure 6. The microstructure and macroscale model had conforming boundary meshes such that no multiple point constraints were required. The entire resulting finite element mesh had 15,931,139 nodes and 11,856,757 quadratic tetrahedra, the simulation of which required a supercomputing platform. The solution was obtained using the PETSc conjugate residuals implementation with the block-Jacobi preconditioner.

To test the parallel performance of ScIFEi, the finite element computations were run on 200, 500, 1000, and 2000 cpu cores, which resulted in approximately 240K, 100K, 50K, and 25K dof per cpu core, respectively. In each of the trials, all 20 of the cpu cores were utilized on each node. Figure 7 is an illustration of the obtained results of the parallel performance study on Pleiades. The wall clock time represents the entire time for ScIFEi to complete the computations. Furthermore, Figure 7 illustrates the most significant contributions to the wall clock time: MOAB reading the mesh file in parallel, and PETSc iteratively solving the system of equations. Between 200 and 500 cpu cores, the simulation speed up was greater than ideal, *i.e.*, 2.5 times the number of cores resulted in a 2.7 times speedup. This rate slowed significantly when fewer than 100K dof were on each cpu core. The implementation of more sophisticated solution methods such as multigrid solvers are planned for ScIFEi to improve scalability.

Also illustrated in Figure 7 is the monetary cost to complete each of the simulations. The cost is computed from the number of nodes used, the wall clock time, the

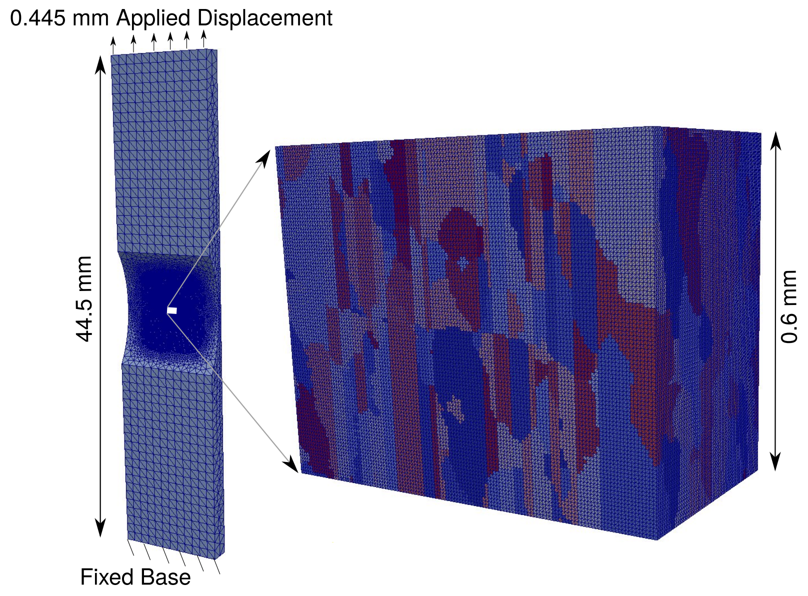

Figure 6: Concurrent multiscale finite element model as obtained from the study presented in [18].

cost per node hour ($0.28), and a multiplication factor (2.52) unique to the compute nodes chosen. From this combined illustration of the monetary cost along with the total wall clock time it is clear that the most efficient use of resources was between 500 and 1000 cpu cores, *i.e.*, 50K-100K dof per cpu core.

7 Summary

This report is a brief introduction to basic ScIFEi concepts, features, vocabulary, and depedencies. To help potential users understand the advantages of using ScIFEi, results of several parallel performance tests are provided. On a single workstation, the scalability of ScIFEi was evaluated and compared against the commercial finite element analysis code, Abaqus, as a point of reference. It is noted that no comparisons are implied regarding breadth of capability between the two codes. Most notably, the results of a single test illustrate that ScIFEi produced a speed up of up to 3.25x when compared to Abaqus performance. Furthermore, relative differences between the outputs of each code were provided as verification of ScIFEi and to ensure that each code's iterative solver was solving the system of equations to a similar degree of accuracy for the timing comparisons.

Commonplace today in computational mechanics research is the requirement to analyze the large 3-D finite element models that are produced using state-of-the-art experimental capabilities, *e.g.*, those obtained via X-ray computed tomography or near-field high-energy X-ray diffraction microscopy. Largely because of its dependence on PETSc and MOAB, ScIFEi is able to take advantage of high-performance computing architectures to analyze these models. Consequently, parallel performance tests were run on Pleiades, a NASA distributed memory super-

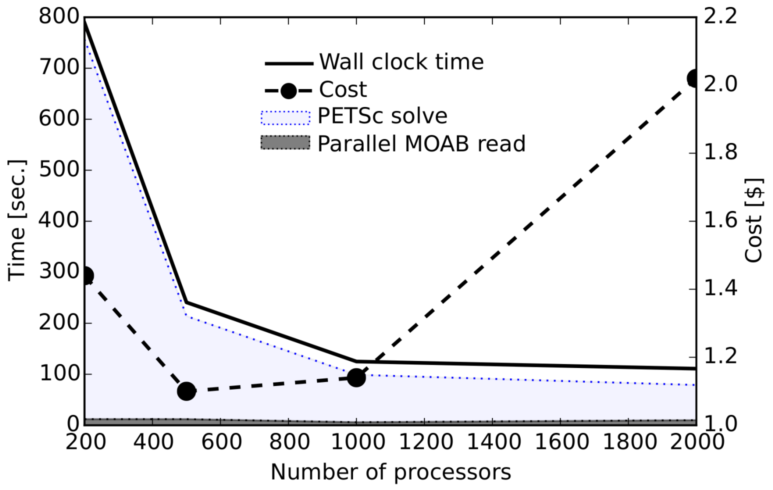

Figure 7: Parallel performance test results from the Pleiades supercomputer.

computer. These preliminary results indicate that 50K-100K dof (approximately 15K-30K nodes) per cpu cores is most cost effective. Future attempts for realizing scalability beyond this should consider the inclusion of multi-grid solvers and graphics processing unit cores.

References

1. Tautges, T. J.; Meyers, R.; Merkley, K.; Stimpson, C.; and Ernst, C.: MOAB: A Mesh-Oriented Database. Sand2004-1592, Sandia National Laboratories, April 2004.

2. The HDF Group: Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/, Jan. 2016.

3. Balay, S.; Abhyankar, S.; Adams, M. F.; Brown, J.; Brune, P.; Buschelman, K.; Dalcin, L.; Eijkhout, V.; Gropp, W. D.; Kaushik, D.; Knepley, M. G.; McInnes, L. C.; Rupp, K.; Smith, B. F.; Zampini, S.; and Zhang, H.: PETSc Web page. http://www.mcs.anl.gov/petsc, 2015.

4. Balay, S.; Abhyankar, S.; Adams, M. F.; Brown, J.; Brune, P.; Buschelman, K.; Dalcin, L.; Eijkhout, V.; Gropp, W. D.; Kaushik, D.; Knepley, M. G.; McInnes, L. C.; Rupp, K.; Smith, B. F.; Zampini, S.; and Zhang, H.: PETSc Users Manual. ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

5. Balay, S.; Gropp, W. D.; McInnes, L. C.; and Smith, B. F.: Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

6. European Centre for Research and Advanced Training in Scientific Computation: MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. http://mumps.enseeiht.fr/, Jan. 2016.

7. Li, X. S.: An Overview of SuperLU: Algorithms, Implementation, and User Interface. *ACM Trans. Math. Softw.*, vol. 31, no. 3, September 2005, pp. 302–325.

8. Sandia National Laboratories: Integrated Codes. http://www.sandia.gov/asc/integrated_codes.html, Jan. 2016.

9. SIMULIA: *Abaqus Documentation*. 6.14 ed., 2014.

10. Belytschko, T.; Liu, W.; and Moran, B.: *Nonlinear Finite Elements for Continua and Structures*. Wiley, 2000.

11. Kitware Inc. : CMake Cross-Platform, Open-Source Build System. http://www.cmake.org/, Jan. 2016.

12. The HDF Group: HDFVIEW. https://www.hdfgroup.org/products/java/hdfview/, Jan. 2016.

13. Lawrence Livermore National Laboratory: VisIt. https://visit.llnl.gov/, Jan. 2016.

14. Kitware Inc. : Paraview Parallel Visualization. http://www.paraview.org/, Jan. 2016.

15. XDMF: eXtensible Data Model and Format. http://www.xdmf.org/, Jan. 2016.

16. CEI Software: EnSight. https://www.ceisoftware.com/, Jan. 2016.

17. NASA High-End Computing Capability: Pleiades. http://www.nas.nasa.gov/hecc/resources/pleiades.html, Jan. 2016.

18. Spear, A.; Hochhalter, J.; Cerrone, A.; Li, S.; Lind, S.; Suter, R.; and Ingraffea, A.: A Method to Generate Conformal Finite-Element Meshes from 3-D Measurements of Microstructurally Small Fatigue-Crack Propagation. *Fat. Fract. Eng. Mater. Struct.*, vol. Accepted, no. 00, 2016, pp. 00 – 00.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 04 - 2016	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Scalable Implementation of Finite Elements by NASA - Implicit (ScIFEi)	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Warner, James E.; Bomarito, Geoffrey F.; Heber, Gerd; Hochhalter, Jacob D.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 736466.07.10.07.01.01
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20685
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA-TM-2016-219180

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 61
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Scalable Implementation of Finite Elements by NASA (ScIFEN) is a parallel finite element analysis code written in C++. ScIFEN is designed to provide scalable solutions to computational mechanics problems. It supports a variety of finite element types, nonlinear material models, and boundary conditions. This report provides an overview of ScIFEi ("Sci-Fi"), the implicit solid mechanics driver within ScIFEN. A description of ScIFEi's capabilities is provided, including an overview of the tools and features that accompany the software as well as a description of the input and output file formats. Results from several problems are included, demonstrating the efficiency and scalability of ScIFEi by comparing to commercial finite element analysis.

15. SUBJECT TERMS

Finite element method; Implicit tme integration; Parallel computing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	20	(757) 864-9658