Preparation of Codes for Trinity

Courtenay T. Vaughan, Mahesh Rajan, Dennis C. Dinge, Clark R. Dohrmann, Micheal W. Glass,
Kenneth J. Franko, Kendall H. Pierson, and Michael R. Tupek
Sandia National Laboratories
Albuquerque, NM 87185
ctvaugh, mrajan, dcdinge, crdohrm, mwglass, kjfrank, khpiers, mrtupek @sandia.gov

***Abstract*—Sandia and Los Alamos National Laboratories are acquiring Trinity, a Cray XC40, with half of the nodes having Haswell processors and the other half having Knight's Landing processors. As part of our Center of Excellence with Cray, we are working on porting three codes, a Solid Mechanics code, a Structural Dynamics code, and an Aerodynamics code, to effectively use this machine. In this paper, we will detail the work that we have done in porting the codes in preparation of getting the machine. We have started by profiling the codes using tools including CrayPat, which showed that a large portion of the time is being spent in the solvers. We will describe the work we are doing on the solvers such as ongoing work on Haswell processors and Knight's Corner machines.**

Keywords; mechanics, sparse solver, Trinity, Haswell performance analysis and tuning

I. Introduction

Los Alamos National Laboratories (LANL) and Sandia National Laboratories (SNL) have formed a partnership, the New Mexico Alliance for Computing at Extreme Scales (ACES), to acquire and deploy a production capability system for the Department of Energy NNSA ASC program to support the national Stockpile stewardships Program. Trinity, a Cray XC40, with approximately 9500 nodes with the Intel Haswell processors and approximately 9500 nodes with the Intel Knights Landing processors is first of the NNSA's new Advanced Technology Systems, ATS-1, anticipated to be installed in 2016. As part of the Trinity Center of Excellence with Cray and Intel, SNL code developers and support staff are working on porting three SIERRA mechanics codes to Trinity. These complex multi-physics applications are: SIERRA/Solid Mechanics (SM), SIERRA/Structural Dynamics (SD) and SIERRA/Aerodynamics. In this paper, we will detail the preliminary work that we have undertaken in preparation for efficient use of these production applications on Trinity. Trinity's architecture challenges these codes to achieve high node level and thread level parallelism, high vectorization efficiency and efficient use of the 576 Burst Buffer I/O nodes with NVRAM. We will present profiles of the codes with tools including CrayPat, vTune, TAU, and HPCtoolkit on our current large Cray XE6 called Cielo, on our large Sandy Bridge/Infiniband cluster called Chama, and on our Knights Corner test beds called Compton and Morgan. All the three SIERRA applications (for SIERRA/SM the implicit case) for the target simulations of interest show a large fraction of the simulation time in matrix equation solves, albeit with different characteristics. These performance profiles and scaling studies will focus on leveraging Trillinos and MKL/Pardiso sparse solver kernels. In SIERRA/SM the preconditioning step dominates consuming more than 90% of the solve time, as it is computed at each time step. The iterative linear solve done with the FETI (Finite Element Tearing and Interconnecting) algorithm requires a local solve, a coarse solve and a preconditioner solve. In contrast, computations for SIERRA/SD are dominated by forward/backward triangular solves associated with already factored matrices for local and global (coarse) problems. That is, much more time is spent applying the preconditioner than in its initialization. We also will present our efforts with vectorization of key compute kernels using tools like Intel's Vector Advisor. For code kernels for which the current generation of compilers are unable to auto-vectorize, we will discuss data structure layout modifications and the direct use of Intel vector intrinsics to systematically improve vector performance.

II. Trinity Architecture

The new machine, Trinity, will be a Cray XC40. The first half of the machine will have approximately 9500 nodes with Intel Haswell processors and is scheduled to be delivered later this fiscal year. Each Haswell node will have two 16 core processors running at 2.3 GHz with 128 GB of memory per node. The second half of the machine will have approximately 9500 nodes with Intel Knights Landing processors, each with 60+ cores, and will be delivered later in fiscal year 2016. The peak speed for the machine will be about 42 Petaflops. The nodes will be connected with Cray's Aries interconnect, which is a Dragonfly interconnect with four nodes connected to each Aries node.

III. SIERRA Mechanics

Sierra is an engineering mechanics simulation code suite supporting the Nation's Nuclear Weapons mission as well as other customers. It has explicit ties to Sandia National Labs' workflow, including geometry and meshing, design and optimization, and visualization. Distinguishing strengths include "application aware" development, scalability, SQA and V&V, multiple scales, and multi-physics coupling [1]. The Sierra code suite is a large C++ framework, which the applications are built upon. It includes several Third Party Libraries as well as common C++ classes and methods for the codes. It also includes the common infrastructure for running the codes on parallel machines.

A. *SIEERA/SM*

SIERRA/SM is a general purpose massively parallel nonlinear solid mechanics finite element code for explicit transient dynamics, implicit transient dynamics and quasi-statics analysis of structures [2]. It is built upon extensive material, element, contact and solver libraries for analyzing challenging nonlinear mechanics problems for normal, abnormal, and hostile environments. It is similar to commercial codes LSDyna and Abaqus. Two models were studied with a view to understanding performance factors that would ensure that analyst applications targeted for Trinity perform well. The first model studied was a weapon component impact analysis that stressed the contact mechanics computations and structural material models. The second model was a quasistatic I-Beam model that uses an implicit solver. With the explicit dynamic computations with contact, SIERRA/SM spends a large fraction of the compute time on proximity search and enforcing contact constraints. With the implicit I-Beam model the focus is on the performance of the FETI [3] pre-conditioner and sparse direct solve at each time step which invoke matrix factorization function calls and forward/backward solves with a few right-hand-sides.

In this paper the focus for SIERRA/SM is only on the implicit model as it shares with the other two SIERRA applications described in this paper, the need for efficient sparse matrix solution capability. The Quasi static solution algorithm consists of four steps:

1. Initialize Time Step, t = 0, dt = dt0
2. Compute Residual Force:
   $R(x,t) = F_{external}(x,t) - F_{internal}(x,t) + F_{contact}(x,t)$
3. Iterate until: R(x,t) = 0
4. If Converged, t = t + dt

The non-linear conjugate gradient algorithm psudo code is as follows:

1. k= 0
2. Loop, until converged
   $R(x,t) = F_{external}(x,t) - F_{internal}(x,t) + F_{contact}(x,t)$
   $G = M^{-1} R(x,t)$ // **preconditioning**
   $S = G + beta * S^{k-1}$ // **axby**
   alpha = LineSearch(S) // **extra residual call**
   x = x + alpha * S // **axby**
   Compute ||R||, check convergence
   Beta = compute_beta() // dot products, all reduce

The preconditioning step dominates the solve time and occurs once per time step. The precondioning is accomplished with a Jacobian matrix which requires an iterative linear solver algorithm to provide M^{-1}. The iterative linear solve is done with the FETI (Finite Element Tearing & Interconnecting solver) domain decomposition algorithm. FETI requires a local solve, coarse solve, and a preconditioner solve (similar to most domain decomposition algorithms) and extensively uses sparse direct solvers.

The quasi static I-Beam model studied is shown in Figure 1.

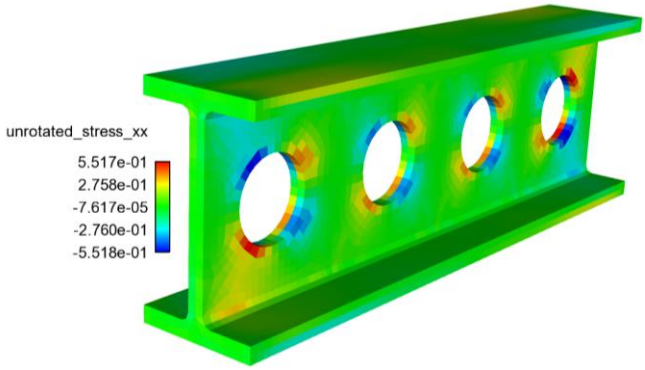

Figure 1. SIERRA/SM quasi-static analysis I-Beam Model with 548,864 hex elements

The model has about one half million elements and meshed so to investigate performance on our moderate size clusters, and not necessarily representative of the mesh sizes targeted for Trinity. When we investigate performance of this or similar models on Trinity, it is anticipated that the mesh sizes would be considerably larger. However investigations of weak scaling with different mesh refinements of this I-Beam model confirms that what we learn from analysis of this one half million element model will be applicable to larger models.

Figure 2 shows the strong scaling characteristics measured on our Sandy-Bridge, FDR inifiniband cluster called Chama.

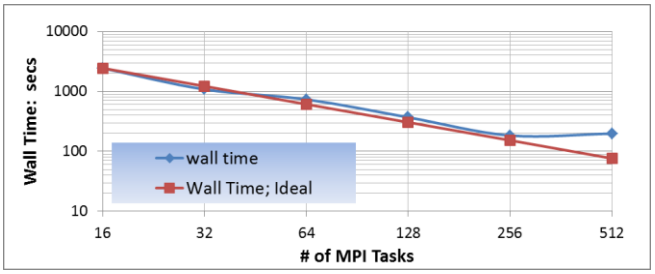

Figure 2. SIERRA/SM I-Beam model strong scaling on Chama

The performance of this model was analyzed on Chama using vTune, mpiP and HPCTololkit. The percentage of run time in MPI is an important component to understand scaling. Figure 3 shows the increase in MPI time with scale for the strong scaling runs shown in Figure 2.

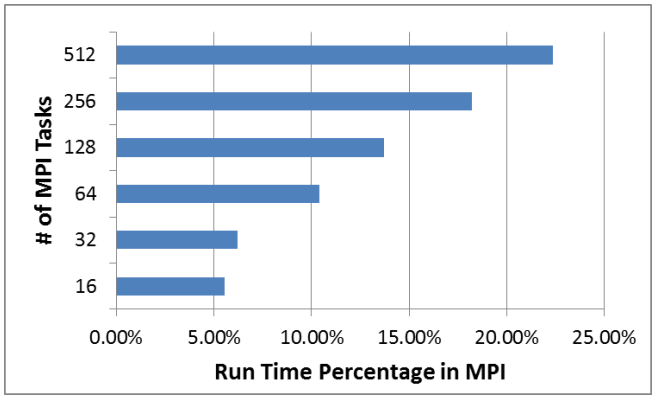

Figure 3. SIERRA/SM fraction of time in MPI with scale on Chama

Allinea MAP and HPCToolkit and vTune were used to identify the hot spots. As anticipated 36% of run time out of which 52% of the time was in MPI calls in the FETI solver. I(~/FETI-DP/src/FETI_DP_FiniteElementData.C (line 919) feti::FetiDriver (FetiDriver.C line 228)). This function called FetiAlgorithm.C (FetiAlgorithm line 275, DirichletPreconditioner line 243, OrthoSet::orthoganlize line 263 & and line 245).

The vTune profile showed that the most compute intensive finctions called by the FETI solver were "blkl2" and "blkslv". These functions invoke basic BLAS operations. As of writing this paper we have not investigated opportunities for threading and vectorization of these compute kernels.

B. SIERRA/Aero

SIERRA/Aerodynamics is a node-centered, edge-based finite volume code that approximates the compressible Navier-Stokes equations on unstructured meshes. It can be run in two or three dimensions. It is applicable to high Reynolds number laminar and turbulent flows. Currently, two classes of turbulence models are provided: Reynolds Averaged Navier-Stokes (RANS) and hybrid methods such as Detached Eddy Simulation (DES). The gas may be modeled either as ideal, or as a non-equilibrium, chemically reacting mixture of ideal gases [4].

Our test case was the isentropic vortex problem. This problem has an exact solution for the compressible Navier-Stokes equations sans the viscous terms (the Euler equations). This test problem involves the convection of an isentropic vortex in inviscid flow. It is used to test the ability of the code to accurately model vortical flows.

We used CrayPat under perftools version 6.0.1 to examine the performance of SIERRA/Aero on the isentropic vortex problem. For this test we used a cluster called Muzia. Muzia is a smaller version of our current large Cray XE6 system Cielo. The results of this profile are given in Table 1 and Figure .

Aero Profile (Major Functions)

Samp %	Samp	Imb. Samp	Imb. Samp%	Function
100	125816.8	-	-	Total
88.7	111577.4	-	-	User
28.4	35715.2	545.8	1.5	sumInto[1]
19.9	25054.0	391.0	1.5	localGaussSeidel[2]
14.5	18261.9	4939.1	21.5	ElementFlux::operator()[3]
13.7	17243.7	232.3	1.3	localApplyBlockNoTrans[4]

1. tftk::linsys::TpetraBaseBlockLinearSystem::sumInto
2. Tpetra::Experimental::BlockCrsMatrix<double, int, long, KokkosClassic::SerialNode>::localGaussSeidel
3. sierra::conchas::ElementFlux::operator()
4. Tpetra::Experimental::BlockCrsMatrix<double, int, long,KokkosClassic::SerialNode>::localApplyBlockNoTrans

Table 1. Aero Profile

The most time consuming function in Table 1, tftk::linsys::TpetraBaseBlockLinearSystem::sumInto, fills the actual linear system with values from the application code. The next highest time consuming function, Tpetra::Experimental::BlockCrsMatrix<double, int, long, KokkosClassic::SerialNode>::localGaussSeidel, is the main work routine of the pre-conditioner (local on each process) that computes a smoothed solution for symmetric Gauss-Seidel. It is called twice for each linear iteration of the code. The third most time consuming function is sierra::conchas::ElementFlux::operator() which does the main computation of the residual and sensitivities for the linear system. And Tpetra::Experimental::BlockCrsMatrix<double, int, long,KokkosClassic::SerialNode>::localApplyBlockNoTrans is a sparse matrix-vector multiply. All other functions were found to use less than 3% of user compute time.

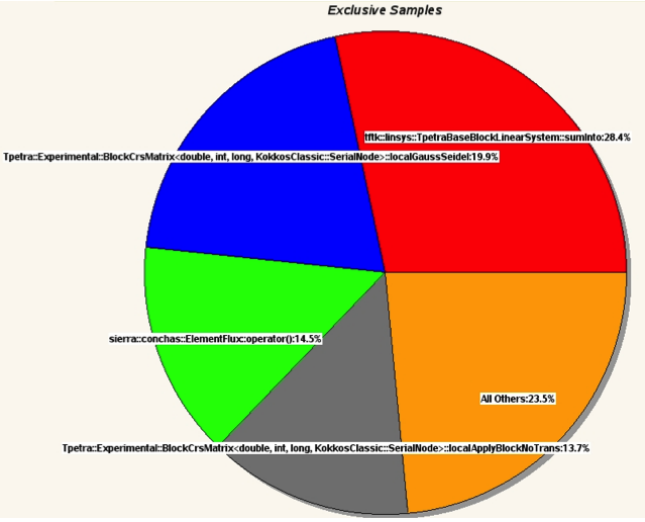

Figure 4. Pie Chart of Most Compute Intensive Aero Functions

Figure 5 shows the strong scaling characteristics measured on Chama.

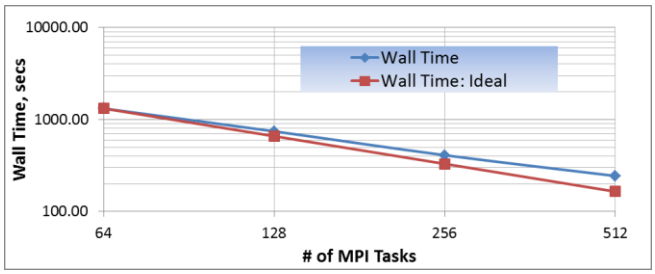

Figure 5. SIERRA/AERO IsentropicEulerVortex model strong scaling on Chama

The percentage of run time in MPI is an important component to understand scaling of SIERRA/AERO and is shown in Figure 6.

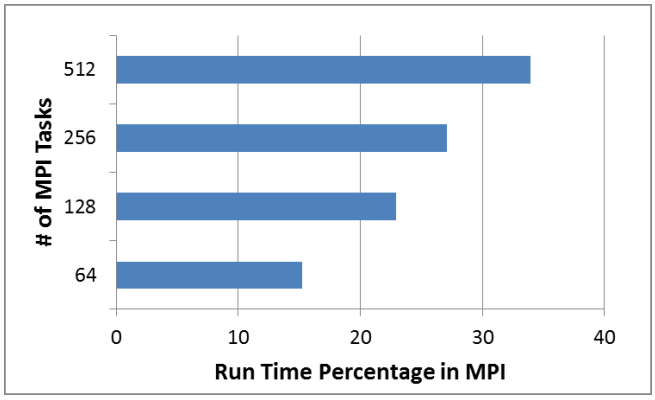

Figure 6. SIERRA/AERO fraction of time in MPI with scale on Chama

C. SIERRA/SD

SIERRA/SD is a general purpose parallel structural dynamic finite element analysis code and is used for large scale structural analysis including problems in modal, vibration, static and shock analysis [5]. The problem that has been chosen for the Center of Excellence work is a modal analysis of a mesh of a complex part with 1044913 nodes and is called "nfn9". For this type of analysis, there are multiple solves for each eigenmode. The result is that the majority of the time (98% or better) is spent in the solve portion of the code and the initialization of the solver is a very small portion of that time. Even though SIERRA/SD has several solver options for several varieties of problems, this behavior is typical of problems run with the code.

This test problem that we ran for SIERRA/SD uses the GDSW (Generalized Dryja, Smith, Widlund) domain decomposition solver [6]. The GDSW solver relies on domain decomposition, where the domain is divided into a number of subdomains (generally, one per processor). Two local solves are done for each of these subdomains, and a global (coarse) solve is also done on a reduced system. These solutions are then combined to produce the preconditioned residual for the Krylov method. In contrast to the other codes that we are studying in this paper, SIERRA/SD avoids the repeated matrix assembly operations, which is why the solution portion of the code dominates the runtime. We ran SIERRA/SD on 120 cores of a Cray XE6 using CrayPat and Figure 7 shows a simplified call tree resulting from the run. There is communication in places other than the "Epetra communication" location in the call tree. As mentioned above, there are multiple backsolves for the solution of each linear system and these backsolves call a routine called "blkslvn" which in turn calls the BLAS routine "dgemm". For this test, this part of the code takes about 84% of the computation time and dominates the run. This is typical for problems that use this solver. Most of the calls to "blkslvn" only have one right hand side and could be replaced by a call to "blkslv", which calls dgemv, which can be more efficient on some machines.

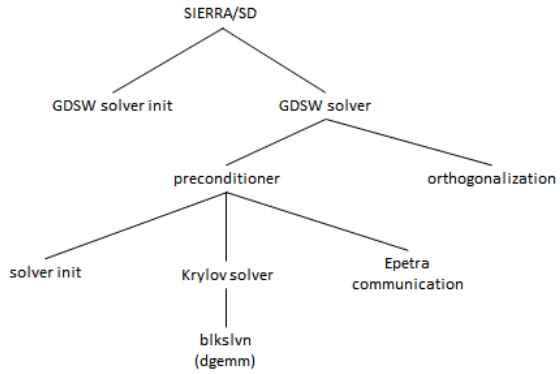

Figure 7. Simplified call tree for SIERRA/SD

We also found that the time MPI takes is a little less than 1% of the total runtime. However, the MPI_SYNC time is about 47% of the runtime. Most of this time is waiting for MPI_Barrier (34%) and MPI_Allreduce (12%). This indicates load imbalance in the calculation. Part of this is due to the fact that the global solve on a reduced system is being done by only one processor. The communication matrix is shown in Figure 8. It shows all of the other cores communicating with rank 105 to do the global portion of the solve. It also shows the usual nearest neighbor communication that results from domain decomposition of the mesh.

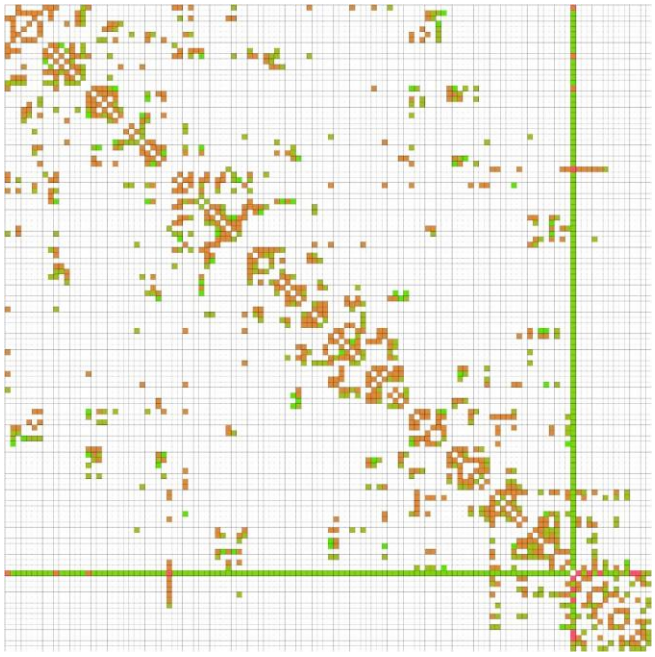

Figure 8. Communication matrix for GDSW solver

Figure 9 shows the strong scaling characteristics measured on Chama. Due to the nature of the problem and solver, the strong scaling curve is a straight line, but deviates from the ideal. As more processors are used, the size of the domains becomes smaller and more time is spent in communication.

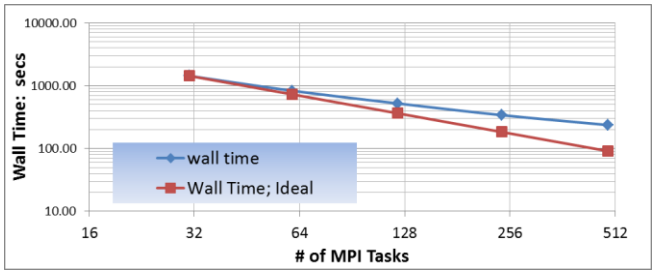

Figure 9. SIERRA/SD nfn9 model strong scaling on Chama

The percentage of run time in MPI is an important component to understand scaling of SIERRA/SD and is shown in Figure 10. The time spent in MPI in this chart includes the MPI_SYNC time discussed before and the total percentage of time in MPI agrees with the results from the runs on the Cray XE6.

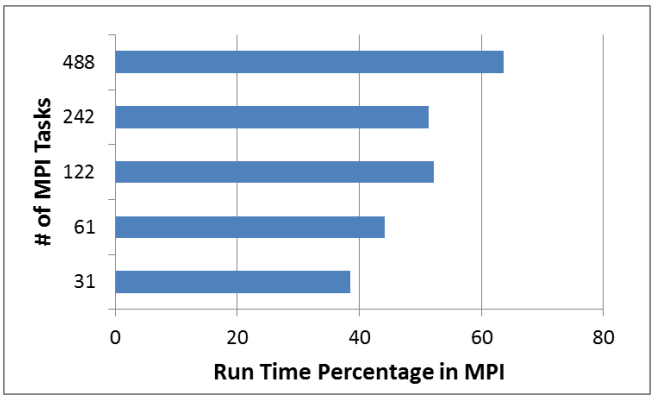

Figure 10. SIERRA/SD fraction of time in MPI with scale on Chama

IV. Summary and Future Work

In anticipation of acquiring Trinity, we have looked at three codes in the SIERRA Framework to characterize their performance and strategize what work needs to be done such that those codes will run well on both halves of the machine. We have looked at the codes and profiled them with various tools such as CrayPat. In doing so, we encountered some problems with compiling with CrayPat due to mismatch of modules since SIERRA has a set of modules that it uses and those clashed with some of the modules that CrayPat needed. We also saw some problems with some versions of CrayPat giving us answers that were not right.

In moving forward, there are several things that we are going to work on to get good performance on Trinity. As we have seen, there are two places in the codes that take up a large portion of the time. Those are the solvers for all of the codes and matrix assembly for SIERRA/SM and SIERRA/Aero. We have been experimenting with Pardiso, the Intel math library that provides threading to help with the solvers. We have also been looking at various methods of threading and vectorization of the codes.

On Trinity, effective vectorization taking full advantage of the AVX2 vector units on each core of Haswell and the two 512-bit vector SIMD units on each core of KNL that support AVX-512F (AVX3.1) instructions, is an important goal for the SIERRA code development team. In [7] the gain in performance possible through auto-vectorization is presented studying TSVC and LCALS benchmarks for Intel, Cray and GNU compilers. They also show that for a few SIERRA/SM compute kernels a range of approaches from changes in data structures to use of pragmas to the

development of specially coded SimdLIB can yield up to 50% performance gain.

ACKNOWLEDGMENT

This work was supported in part by the U.S. Department of Energy. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] Ryan P. Shaw, Anthony M. Agelastos, and Joel D. Miller, "Guide to Using Sierra," SAND2015-1642, Sandia National Laboratories, Albuquerque, NM, 2015.

[2] Sierra/SM Team, "Sierra/SM Theory Manual," SAND2013-4615, Sandia National Laboratories, Albuquerque, NM, 2013.

[3] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen, "FETI-DP: A Dual-Primal Unified FETI Method – Part I: A Faster Alternative to the Two-Level FETI Mathod," International Journal for Numerical Methods in Engineering, 2001, Vol 50, pp. 1523 – 1544.

[4] SIERRA/Aero Theory Manual – Version 4.34 (Internal Sandia Document)

[5] Garth M. Reese, Timothy F. Walsh, and Manoj K. Bhardwaj, "Salinas – Theory Manual Version 4.22," SAND2011-8272, Sandia National Laboratories, Albuquerque, NM, 2011.

[6] Clark R. Dohrmann and Olof B. Widlund, "Hybrid domain decomposition algorithms for compressible and almost incompressible elasticity," International Journal for Numerical Methods in Engineering, 2010, Vol 82. pp. 157-183.

[7] Rajan, M., Doerfler, D.W., Tupek, M.R., and Hammond, S.D.," An investigation of compiler vectorization on current and next-generation Intel processors using benchmarks and Sandia's Sierra Applications," to be presented at CUG 2015, Chicago, Il, 2015.