

Using Miniapplications in a Mantevo Framework for Optimizing Sandia’s SPARC CFD Code on Multi-Core, Many-Core, and GPU-Accelerated Compute Platforms

Daniel W. Barnette, Richard F. Barrett, Simon D. Hammond, Jagan Jayaraj, James H. Laros III
{*dwbarne, rfbarre, sdhammo, jnjayar, jhlaros*}@sandia.gov
Sandia National Laboratories, Albuquerque, New Mexico 87185, USA

Advances toward exascale-class compute system architectures are pushing the limits of code team developers to stay current with correspondingly optimized large-scale applications. Large application codes, often unwieldy, are typically used to explore performance parameters such as processor architecture, grid structure and partition size, network topology and latency, programming models, data structures, power profiles, and various PDE discretization schemes and solver algorithms. Instead, we choose to develop a set of validated, portable, and easy-to-use miniapplications, or miniapps, to individually isolate and explore those parameters which most strongly influence performance metrics of the full application. We use as the target application a new lightweight compressible CFD code being developed at Sandia called SPARC. Our results with miniapps for SPARC will help drive other research efforts in the community to determine optimal platforms for given program needs such as efficient CFD solutions on many-core systems, new algorithmic development for accelerated platforms, and platform acquisition requirements. This paper discusses our efforts to-date on recent testbed and cluster architectures. We also discuss future directions.

I. Introduction

The complexity of exascale architectures requires application software developers to have a greater knowledge of system internals than ever before. Algorithms that ran efficiently in the past on serial, vector, or distributed-memory machines are expected to require a new paradigm from code developers if they are to make efficient use of the hybrid shared-distributed memory systems and compute power available in future platforms. Many performance improving approaches are available to developers that have yet to be studied in a systematic manner. Which approach or approaches will rise to the top of the heap, if any, are far from being decided.

The Mantevo project [1] represents an effort to systematically tackle these complexities. This project provides a framework under which much smaller applications called miniapps may be developed to test and report on focused performance parameters [2]. Such miniapps are emphatically not designed to capture convergence rates or accurate physics, nor do they represent reduced models of the full application. Rather, miniapps capture key performance issues that can be exposed in comparatively small, easy to execute, easy to modify code. Hence, a full application may have a suite of representative miniapps to test performance issues directly related to the full code. Once a miniapp shows a particular improvement on a newly introduced platform or architecture, for example, the more efficient algorithm represented in the miniapp may be more easily introduced into the main application. The goal is that similar improvement in the main application can then be realized.

We are in the process of developing a suite of miniapps to investigate various aspects of the full application to include I/O, memory use, architecture variations, grid construction, solution algorithms and more on a wide range of recently acquired multi-core, many-core, and GPU-accelerated architectures from a variety of vendors. We use our approach to investigate optimizing the Sandia Parallel Aerosciences Research Code (SPARC) compressible CFD code under development at Sandia. This ‘work-in-progress’ paper reports our progress to date as well as future directions.

II. Mantevo Project Overview

Our miniapps are developed under the aegis of the Mantevo project. Mantevo is a multi-faceted application performance project providing open-source software packages for the analysis, prediction, and improvement of high performance computing applications.

Mantevo provides application proxies of several types and sizes:

- **Miniapplications:** Small, self-contained programs that embody essential performance characteristics of key applications.

- **Minidrivers:** Small programs that act as drivers of performance-impacting components of production libraries such as the Trilinos solver library or Sandia’s engineering packages [3].
- **Application proxies:** Parametrizable applications that can be calibrated to mimic the performance of a large scale application, and then used as a proxy for the large scale application.

Goals for the project include

- Providing open source software to promote informed algorithm, application and architecture decisions within the HPC community
- Predicting performance of real applications in new architectures
- Aiding computer systems design decisions
- Fostering communication between applications, libraries and computer systems developers
- Guiding application and library developers in algorithm and software design choices for new systems

The primary focus of this paper is development of miniapps for SPARC. Unlike a benchmark, the result of which is to provide a ranking metric, the output of a miniapp is information which must be interpreted within some often subjective context. Unlike a compact application, which is designed to capture some sort of physics behavior, miniapps are designed to capture some key performance issue in the full application. Miniapps are typically developed and owned by application code teams. They are intended to be modified as needed and are inherently much shorter in number of source code lines than their target application.

Current Mantevo miniapps are listed in Table 1. Most can be downloaded at <http://mantevo.org/download.php>.

#	Miniapp	Description
	miniAero	Under development; overset and patched grid partitioning and flow solver study; currently 2D incompressible Navier-Stokes
1.	HPCCG	Conjugate gradient solver
2.	miniALE	Under development; remap step of ALE solvers
3.	miniFE	Implicit finite element method (FEM) assembly and solve
4.	miniGhost	Finite difference or volume method
5.	miniMD	A light-weight molecular dynamics application containing the performance impacting code from LAMMPS[4] simulator
6.	miniXyce	Circuit simulation
7.	miniETCFE	Under development; explicit dynamics for finite element models
8.	CloverLeaf	Langrangian-Eulerian hydro code; solves compressible Euler equations on a Cartesian grid; solver is explicit, second-order accurate
9.	CoMD	Molecular dynamics code used for exascale co-design studies

Table 1. Miniapps currently in the Mantevo framework

III. Testbeds and Compute Resources

Sandia is well-positioned to explore optimization characteristics as a function of platform variability. We intend to exercise our approach on a variety of available machines listed in the Table 2.

Each architecture meets our requirement of running existing MPI applications through execution on conventional commodity processor cores, requiring only recompilation. Additional programming languages and models include OpenMP, OpenACC, and OpenCL, allowing targeting of specialized compute hardware within each machine.

#	Resources	Description
1.	Cray XE6 (Cielo)	<ul style="list-style-type: none"> 8,894 compute nodes; 32 GB per node; uses Compute Node Linux (CNL) two AMD 2x8 Magny-Cours per node (16 cores per node) Cray 3D Torus Gemini interconnect
2.	Teller	<ul style="list-style-type: none"> 104 nodes of Trinity AMD APUs: K10 (4x2.9 GHz) + GPU (384 x 850 MHz) with a common virtual-address space Quad-core CPU and 384-core GPU InfiniBand interconnect
3.	Curie	<ul style="list-style-type: none"> 52 nodes of AMD Interlagos (8-“core-pairs” (16 cores total) @2.1 GHz) + Nvidia Kepler K20-X GPU (16x32) Cray XK7 Cray Gemini interconnect
4.	Compton	<ul style="list-style-type: none"> 42 nodes of dual-socket Intel Xeon Sandy Bridge CPUs (2 sockets x 8 cores at 260 GHz, 64 GB of system memory) + dual-card Knights Corner pre-production cards (57 cores & 1.1 GHz with 6GB of GDDR5 memory) Intel MIC – Knights Corner Mellanox InfiniBand interconnect
5.	Tilera TILE-Gx36 processors	<ul style="list-style-type: none"> 4 x 36 cores@1.2 GHz.
6.	Convey HC-1ex	<ul style="list-style-type: none"> Xeon Nehalem (4 @2.13 GHz), 4 FPGA Co-processor, 8 FPGA “personalities”
7.	Calxeda/ARM	<ul style="list-style-type: none"> 1.1 GHz 8 nodes, 4 cores per node
8.	Shannon	<ul style="list-style-type: none"> 42 nodes of dual-socket Intel Sandy Bridge (8 cores x 2 sockets = 16 cores/node) Dual Nvidia Kepler K20-X cards Infiniband interconnect
9.	Redsky	<ul style="list-style-type: none"> Sun Microsystems Vayu-X6275 blades; 22,528 cores unclassified 2.93 GHz dual socket/quad core Nehalem X5570 processors 12GB RAM per compute code (1.5GB/core); 64 TB RAM total 3D torus QDR InfiniBand interconnect
10.	Chama	<ul style="list-style-type: none"> Intel Sandy Bridge, dual socket, 8 cores/socket, 2.6 GHz, 32 GB of system memory 1,232 nodes; 19,712 cores Qlogic InfiniBand 4X QDR, fat tree interconnect

Table 2. Platforms available for optimization research.

IV. Sparc Description and Baseline Results

SPARC is a lightweight, compressible, viscous, finite-volume CFD code under development at Sandia. The governing equations representing mass, momentum, and energy are [5]

$$\int_{\Omega_f} \frac{\partial \mathbf{U}}{\partial t} d\Omega + \int_{\Gamma_f} \mathbf{F}_i(\mathbf{U}) n_i d\Gamma - \int_{\Gamma_f} \mathbf{G}_i(\mathbf{U}) n_i d\Gamma = \int_{\Omega_f} \mathbf{S} d\Omega \quad (1)$$

where

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho v_j \\ \rho E \end{pmatrix} \quad \mathbf{F}_i(\mathbf{U}) = \begin{pmatrix} \rho v_i \\ \rho v_i v_j + P \delta_{ij} \\ \rho E v_i + P v_i \end{pmatrix} \quad \mathbf{G}_i(\mathbf{U}) = \begin{pmatrix} 0 \\ \tau_{ij} \\ \tau_{ij} v_j - q_i \end{pmatrix} \quad (2)$$

$$\tau_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) + \lambda \delta_{ij} \left(\frac{\partial v_k}{\partial x_k} \right) \quad q_i = -\kappa \frac{\partial T}{\partial x_i} \quad (3)$$

and \mathbf{S} represents any internal source terms. The code uses either node-centered or cell-centered discretization schemes. Gradients are evaluated using typical Green-Gauss integrals. Identical gradients are used for limiter

calculations. Viscous fluxes are evaluated directly at the face using “L” and “R” primitives and gradients. Both explicit and implicit solvers are implemented in the code.

Though SPARC is in development phase, we still require baseline results at some point to measure improvements or regressions in the code as it evolves. The code version used in this report is MPI-everywhere approach and comes with the following test cases for measuring baseline results, complete with partitioned grids and input files for running at scale: sphere, blunt wedge, flat plate, bump, and vortex transport. The geometries are illustrated in Fig. 1. All test cases were run with identical number of iterations.

Fig. 2 illustrates results obtained by running SPARC, sphere, weak scaling, on our RedSky Nehalem-based cluster without Advanced Vector Extensions (AVX), and on Chama with and without AVX. Execution time is plotted versus increasing MPI tasks; hence, lower is better in the graph. Results indicate a significant speedup when run on Chama with the new Sandy Bridge architecture. However, little difference is gained by simply specifying the use of AVX on the compile line. This would indicate that more effort is needed to vectorize SPARC. Ideally, the curves should be fairly flat across the graph for weak scaling runs. In its current form, SPARC does not scale well. Improvement is expected in future releases, however, as we investigate ways using miniAero to better vectorize the code.

Also illustrated in Fig. 2 are effects of using more and more nodes, as the number of cores is fixed at 2 per node as MPI tasks increase. This means that more and more network traffic is generated as the tasks increase. At the lower end of the scale, the execution time for the three runs is almost identical. The difference is significant at the higher end. Hence, even though Chama’s Sandy Bridge processor runs slower than Redsky’s Nehalem (cf. Table 2), Chama’s fat tree interconnect appears to play a large role in decreasing execution time compared to Redsky’s 3D torus interconnect. We will attempt to duplicate these results with miniAero.

Fig. 3 shows the effects of running SPARC, sphere, weak scaling, on RedSky and Chama for 32 tasks but increasing the number of cores used per node. Redsky has a maximum of 8 cores per node, while Chama has 16. Again, the reduction in execution time is significant between the two machines, but the use of AVX on Chama appears to have little influence as currently implemented. It is believed the Chama runs are so close percentage-wise that the crossover shown may be due to noise. We intend to profile SPARC to examine more closely the AVX and MPI behavior.

Fig. 4 examines the use of either one or both sockets per node on RedSky while varying the cores per socket as 1, 2, or 4 and using either 1 or both sockets on the node. Results are for SPARC, blunt wedge, weak scaling. Intel’s 11.1 compiler was used with OpenMPI to generate the binary. Execution time is plotted versus the number of MPI tasks. Recall that RedSky has dual-sock quad-core nodes. Results indicate significant reduction in execution times occurs as each node is given more computational work. The most significant reduction for these cases occurs when both nodes and all cores on each node are utilized. This is as expected since less of the 3D torus interconnect is used, reducing latency and increasing bandwidth between cores and nodes. The less work on each socket, and therefore the more nodes used, the higher the network traffic between nodes, again significantly affecting execution time. Scaling appears to be good only for the cases in which the nodes are fully loaded. Obviously, significant variations in run time can occur by naïve core and socket allocation.

V. Development of MiniAero Miniapp

We are currently developing CFD miniapps for the SPARC code under the Mantevo paradigm. As listed in Table 1, our first SPARC-related miniapp is called miniAero. MiniAero solves the two-dimensional incompressible Navier-Stokes equations around arbitrary bodies using curvilinear grids. Point-to-point overlapped grids are used. The ability to model generically overset grids will be implemented in the near future. MiniAero also contains a grid partitioning scheme developed at Sandia that maximizes the volume-surface, or compute-communicate, ratio for the partitioned grids based on the number of cores used at runtime.

Although we are developing this miniapp as a prototype for later ones which will certainly include compressible flow, one might ask why we use an incompressible flow solver when SPARC is a compressible flow solver. The answer lies in keeping with the concepts of why we use miniapps at all: miniapps allow users to investigate performance trends, not necessarily model exact physics or obtain exact answers as in the case of reduced models, which once proven in one setting may be directly applied in another. As will be discussed, such a miniapp prototype is also an ideal vehicle for researching effects of using different compilers, briquettes for cache pipelining, task graphs for increasing parallelism, or vector data coalescing algorithms in one easy-to-use code, even if the modeled physics is not that of the targeted code.

Our miniAero prototype currently uses the MPI-everywhere programming model. We will be adding capabilities to run MPI simultaneously with threads via OpenMP, CUDA, OpenCL, or OpenACC in the near future. This will allow miniAero to run on the newest hybrid CPU/GPU and many-core test platforms listed in Table 1.

As of this writing, miniAero is being validated. Results will be included here once validations are complete.

VI. Data Management

Data files collected from SPARC and miniAero are persistently stored in MySQL databases for analysis and re-use. Previous storage and analysis methods, typically relegated to manual insertion of performance data into local spreadsheets, have proved to be inadequate for the extensive capture-store-analyze data methodologies and quick response time commensurate with the miniapp concept. Performance data are written to YAML-formatted [6] text files. A typical analysis involves tens or hundreds of these data files for each platform, each with a myriad of performance metrics, depending on the parameters being investigated.

Due to the volume of data to be analyzed, a suite of codes has been developed at Sandia, the aggregate of which is called PYLOTDB [7], to aid in reducing performance data determined from miniapp runs. This open-source software suite allows users to select YAML-formatted data files and transfer them to a pre-formatted MySQL database table, extract the information from the data file residing in the database, and insert the data into table fields generated on-the-fly from labels in the data file. The fields can then be selected for use in a wide range of graphical analyses. The suite also allows database creation, editing, formatting, and general management tasks. For convenience, a script is provided so that users may also email results to a database. The suite is written in Python. Graphs include X-Y, scatter, Kiviat diagrams, polynomial curve fits, pie charts, and bar charts, all of which can be saved in various document-friendly formats. Our data management approach allows for faster presentation of performance characteristics, investigations into effects of new schemes or algorithms, easier collaboration between researchers, and data re-use. The PYLOTDB suite is available on GitHub [8].

VII. Related Investigations

A. Other miniapp results

Besides obtaining and analyzing baseline results, other miniapps have been used to investigate areas related to SPARC running efficiently on newer architectures.

For example, miniGhost has been used to compare execution-time trends for optimizing a Sandia-developed hydrodynamics code [10]. By re-ordering how nodes were assigned, the number of hops in the z-direction of a 3D mesh topology was significantly reduced as shown in Fig. 5. Run times dropped significantly, as indicated in Fig. 6. The miniapp indicated the trends as well, proving that a carefully constructed miniapp can provide guidance for such tasks. The data show node assignment and locality play an important part in mapping miniGhost and its parent application CTH to specific architectures. Node layout and data locality must be carefully monitored, then, when obtaining optimal performance for our SPARC code.

Strong scaling studies were completed on our local platform Teller described in Table 2 using various compilers and the miniFE miniapp shown in Table 1. Results were obtained only on the CPUs, not the GPUs, for our first runs. Various grid sizes were used in the study. Combinations of compilers and programming models were studied: GNU with MPI, Open64 with MPI, Sun with MPI, and Open64 with MPI. Typical results for grid size effects on total run time are presented in Fig. 7 for the GNU compiler using OpenMPI. Results illustrate exponential decreases in total run time as is standard for strong scaling studies and are well balanced for this machine.

Kiviat diagrams, Fig. 8, illustrate the degree at which the compiler and programming model balance with machine architecture. Note that the GNU compiler with OpenMPI is well balanced for the given machine architecture and the variables under investigation. On the other hand, the Open64 compiler with OpenMP is not well balanced for this architecture. Hence, the application corresponding to the miniFE miniapp would likely not scale well using the Open64/OpenMP combination. Other architectures may cause these results to vary. At this point, it is not known whether the problem lies with the newest compilers on the newest architectures or whether underlying problems exist that are revealed through use of various compilers. Regardless, these variations indicate further study is needed, and miniapps such as miniFE provide the ideal vehicle for doing so.

B. Power/Energy Studies

Sandia has focused our research regarding power and energy on High Performance Computing (HPC) platforms at very large scale (thousands of nodes). Our research to date has been conducted using real production Department of Energy, National Nuclear Security Administration (DOE/NNSA) applications. We have conducted studies to determine the trade-offs between performance and energy use while tuning both the CPU frequency and network bandwidth. Our research has shown that the performance energy trade-off is application, architecture and scale dependent. In our experiments we have measured CPU energy savings of up to 40% with little to no impact on performance can be obtained for some applications [9]. We have also shown that while some applications are insensitive to network bandwidth reduction, some are immediately impacted by any reduction in available network bandwidth.

Obtaining empirical data for experiments such as these requires an in situ measurement capability. Our past work was conducted on the Cray XT architecture, which exposed component level measurement ability at a fairly high frequency along with a management infrastructure that accommodated single point data collection [11]. This initial work brought attention to the utility of this type of measurement and control capability and similar capabilities are being implemented on advanced architecture platforms from a number of computer vendors.

Power and energy have also been a focus of the Advanced Architecture Test Bed program at Sandia Labs. In partnership with Penguin computing a component level measurement capability has been developed that can be integrated into commodity clusters. The PowerInsight device [12] allows us to analyze the performance characteristics and energy profiles of applications running on future processors and architectures. The component level granularity and extremely high frequency of this new measurement capability will be used to advance our research in this area.

By leveraging the Mantevo suite of miniapplications, and other proxy applications, that represent important aspects of our production workload we will continue to investigate energy-performance trade-offs. Using miniapplications will both broaden the coverage of our application space and accelerate our research in this area. Our goals will be to determine which energy savings techniques have a positive effect on our miniapplications and which techniques translate to real increases in energy efficiency when applied to production applications, including future versions of SPARC.

VIII. Future Directions

Future directions for our investigations into using miniapps for SPARC include the following areas. Each area will first be implemented in some form in miniapps designed to run quickly and focused on the particular performance characteristic.

A. SPARC

Since our initial investigations, native vector processing capability has been added to SPARC. We will be re-running our test cases to compare with our baseline results.

Looking forward, our strategy for decomposing SPARC into multiple miniapps includes investigating

- Vector processing on newest architectures, along with data coalescing algorithms
- Setup/assembly phase (leverage miniFE)
- Various classes within SPARC
- Performance profiles (MPITune, VTune)
- Data movement across grid boundaries (leverage miniGhost)
- Use of various programming models (*e.g.*, MPI + OpenMP, etc.)
- Strong scaling

In addition, we will continue to investigate performance characteristics of SPARC when Trilinos libraries are used for matrix solves.

B. Briquettes

Woodward *et al.*, at the University of Minnesota have identified a data layout and code transformations to significantly boost the performance of CFD codes involving spatial derivatives and structured grids [13-15]. The key strategy is to reduce the required memory bandwidth off the chip and effectively utilize the short vector engines or the Single Instruction Multiple Data (SIMD) engines present in all the modern CPU/GPU architectures. Fundamental to these optimizations is a new data structure called the briquette. A briquette is a small contiguous record of data corresponding to a cubical region of the problem domain. It contains all the problem state information, which may be many variables, for that spatial region. Each sub-domain is made up of a 3-dimensional arrangement of briquettes.

Briquettes offer the following benefits:

- Coalesced reads and writes: The briquettes are constructed to be multiple cache lines long. They are read and written to the main memory as an atomic unit. This makes reading and writing them efficient and eliminates the need for strided memory accesses.
- Smaller working set: The size of the working set, a set of all the temporaries involved in the computation, is usually proportional to the size of the grid. The size of the working set to update a small briquette of size say 4^3 cells is now proportional to the size of the briquette, which is substantially smaller. The working set is small enough that they may become cache-resident. This reduces the required memory bandwidth significantly.
- Single Instruction Multiple Data (SIMD) vectorization: SIMD engines on most modern microprocessors are more efficient with shorter rather than longer vectors because short vectors allow for data reuse in registers and near-level caches. Long vectors usually spill to main memory. The smaller size of briquettes is more suitable for short vector operations. We vectorize along the planes of a briquette. The size of a plane becomes our vector length universally throughout the computation. We make it a multiple of the SIMD width. Short vectors are efficiently processed when individual planes and briquettes are aligned. The size of the planes and briquettes is suitable for alignment.
- Data alignment: The spatial locality offered by briquettes makes the data assembly to exploit the SIMD engines tremendously efficient. In a dimensionally split algorithm for solving a PDE, the three dimensional update of the problem state occurs in sweeps along each component dimension [18]. Transposing the problem state between sweeps is a data assembly technique to improve both the data alignment for computation by SIMD engines and reduce the number of strided accesses. It performs the gather-scatter operations once per sweep. With briquettes, we perform transposes within a briquette. It eliminates strided accesses to main memory for individual words, as the briquette is fully cache resident for the duration of the transpose operation.

Pipelining for re-use: Programming is difficult with the briquettes. However, most of the complexity can be avoided by performing redundant computations, and copies, at each briquette. Jayaraj and Lin have built a source-to-source pre-compilation tool called the CFD Builder to generate a more complex code expression without the redundant computations and copies from this simpler input expression [14]. CFD Builder eliminates the redundancies by aggressively pipelining the computation of a strip of briquettes. The partial results of computation in a briquette are reused by its adjacent briquettes. The main objective of the pipelining is data reuse and not parallelism. Therefore, it is called pipelining for reuse. The pipelining for reuse enables the CFD Builder to perform memory optimizations by reducing the size of the temporary computational arrays as explained in [13].

Preliminary results with the Piecewise Parabolic Method (PPM) advection module [16-18] as seen in Fig. 9 show the benefit of briquette and pipelining for reuse transformations. Four different combinations of the briquette and pipelining for reuse transformations were explored. The pipelined code with briquettes was generated by CFD Builder from an input expression with briquettes. The code without briquettes was pipelined manually but closely resembles the output from CFD Builder minus the briquettes. The four codes were benchmarked on two different Intel architectures, Nehalem and SandyBridge. For Nehalem, the codes were compiled with the Intel 9.0 Fortran compiler, and `-O3` and `-xT` (for SIMD vectorization) options, and run on a dual-socket dual-hyperthreaded quad-core Xeon 5570s. They have been threaded 16-way using OpenMP to utilize all the logical cores in the node. Similarly for Sandy Bridge, the codes were compiled with the latest Intel 13.0 Fortran compiler, and `-O3` and `-xAVX` flags, and run on a dual-socket dual-hyperthreaded 8-core Xeon E5-2670s. We set the number of OpenMP threads to 32 to utilize all the logical hardware threads in the node. We see 6.7x speedup on Nehalem and 6.28x

speed-up on SandyBridge by applying briquette and pipelining for reuse transformations. As a result of the indicated performance improvements seen with briquettes, we intend to apply the transformations to our miniAero miniapp and, if beneficial, later to the SPARC code.

C) Task Graphs

Task graphs describe data dependencies in a manner such that algorithms can take advantage of parallelizing non-dependent data. Both OpenCL and CUDA come close to this when run on a single node. The usefulness of task graphs applies across different programming models, enhancing portability. The primary goal is to make an application more parallel. However, the difficulty comes in the fact that task graphs are not easily automated. This will depend on both the hardware and the algorithms as well as how to express the functionality in various computer languages.

D) Vector Processing

With ever-widening vector processor units, the task of coalescing data for vector processing is becoming more and more difficult. For example, Intel's Sandy Bridge architectures provide 256-bit wide vectors enabling operating on 8 single-precision elements at a time. Preparing data for optimal vector processing operations is also being investigated.

IX. Conclusions

Work-in-progress results have been presented for SPARC, a lightweight CFD code under development at Sandia, and relevant miniapps. Benefits of node assignment and locality, possible effects of network noise, variations in the use of different compilers and programming models, and performance assessments for the current SPARC code version have been examined and quantified. We will continue to analyze various CFD constructs on multi-core, many-core, and GPU-accelerated compute platforms. We strive to discover which variables such as programming models, data structures, and solver algorithms, to name a few, run most efficiently on which platforms and to present these results as progress is made. Miniapps developed under the Mantevo project are being used to drive this discovery process, with SPARC as one of our primary target applications. Our results will help drive other research efforts in the community to determine optimal platforms for given program needs such as energy-efficient CFD solutions on many-core systems, new algorithmic development for accelerated platforms, and platform acquisition requirements.

References

1. Mantevo Project, <http://mantevo.org>
2. M. A. Heroux, D. W. Doerfler, Paul S. Crozier, James W. Willenbring, H. Carter Edwards, Alan Williams, M. Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich, "Improving Performance via Mini-applications," Technical Report SAND2009-5574, Sandia National Laboratories, September 2009.
3. An Overview of the Trilinos Project - ACM Transactions on Mathematical Software , Vol. 31, Issue 3, September 2005, Pages 397-423.
4. Large-scale Atomic/Molecular Massively Parallel Simulator, <http://lammmps.sandia.gov>.
5. J. H. Ferziger, M. Peric, *Computational Methods for Fluid Dynamics*, ISBN-10: 3540420746, [Springer](#).
6. "YAML Ain't Markup Language," <http://yaml.org>.
7. D. W. Barnette, M. A. Heroux, J. W. Shipman, "Supercomputer and Cluster Application Performance Analysis Using Python," presented at the US PyCon 2011 Python Users Conference, Atlanta, GA, March 2011.
8. Download PYLOTDB suite at <https://github.com/dwbarne>.
9. R. F. Barrett, S. D. Hammond, C. T. Vaughan, D. W. Doerfler, M. A. Heroux, J. P. Luitjens, D. Roweth, "Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS12)," Salt Lake City, UT 2012.
10. Energy Based Performance Tuning for Large Scale High Performance Computing Systems - James H. Laros III, Kevin T. Pedretti, Suzanne M. Kelly, Wei Shu, Courtenay T. Vaughan, 20th High Performance Computing Symposium, Orlando Florida, March 2012.

11. Topics on Measuring Real Power Usage on High Performance Computing Platforms - James H. Laros III, Kevin T. Pedretti, Suzanne M. Kelly, John P. Vandyke, Kurt B. Ferreira, Courtenay T. Vaughan, Mark Swan, IEEE International Conference on Cluster Computing, September 2009.
12. <http://www.penguincomputing.com/resources/press-releases/penguin-computing-releases-new-power-monitoring-device>
13. Woodward, P. R., J. Jayaraj, P.-H. Lin, and P.-C. Yew, "Moving Scientific Codes to Multicore Microprocessor CPUs," Computing in Science & Engineering, special issue on novel architectures, Nov., 2008, p. 16-25. Preprint available at www.lcse.umn.edu/CiSE.
14. Woodward, P. R., J. Jayaraj, P.-H. Lin, P.-C. Yew, M. Knox, J. Greensky, A. Nowatzki, and K. Stoffels, "Boosting the performance of computational fluid dynamics codes for interactive super-computing," Proc. Intl. Conf. on Comput. Sci., ICCS 2010, Amsterdam, Netherlands, May, 2010. Preprint available at www.lcse.umn.edu/ICCS2010.
15. Pei-Hung Lin, J. Jayaraj, P. R. Woodward, P. Yew, A Code Transformation Framework for Scientific Applications on Structured Grids, Technical Report 11-021, UMN Computer Science and Engineering Technical Report, Sep. 2011.
16. Woodward, P. R., and P. Colella, "The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks," *J. Comput. Phys.* 54, 115-173 (1984).
17. Colella, P., and P. R. Woodward, "The Piecewise-Parabolic Method (PPM) for Gas Dynamical Simulations," *J. Comput. Phys.* 54, 174-201 (1984).
18. Woodward, P. R., "The PPM Compressible Gas Dynamics Scheme," in *Implicit Large Eddy Simulation: Computing Turbulent Fluid Dynamics*, ed. F. Grinstein, L. Margolin, and W. Rider, Cambridge University Press (2006).

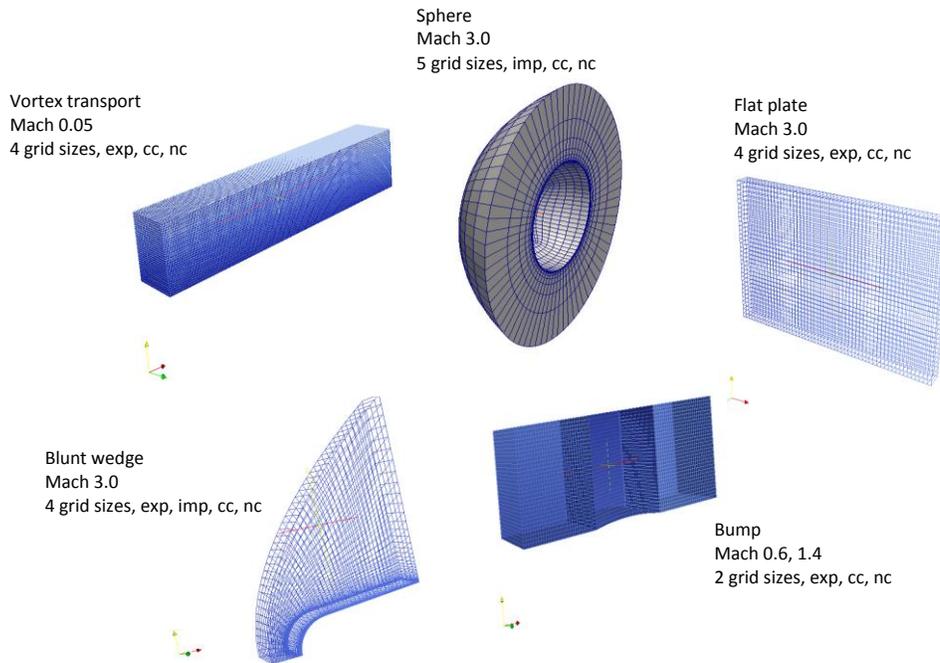


Figure 1. Test cases available for SPARC.

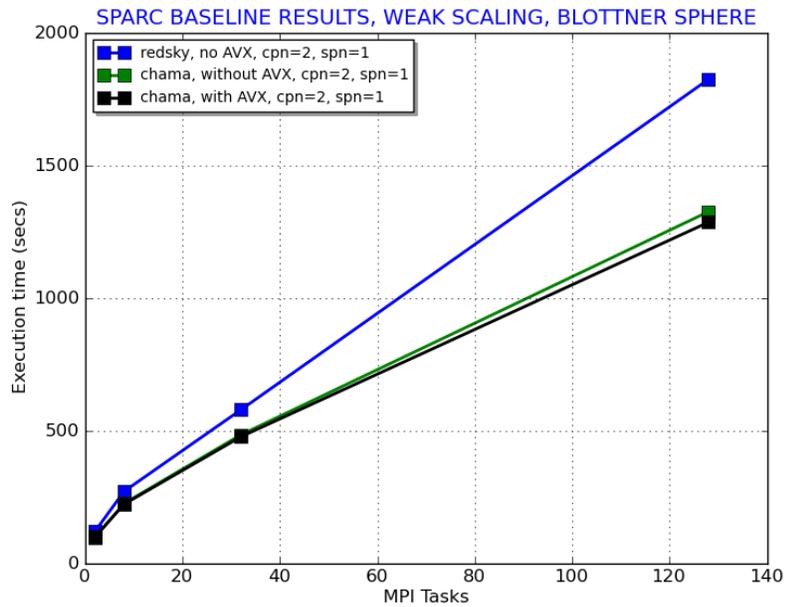


Figure 2. SPARC baseline results from REDsky and Chama (cpn: cores per node; spn: sockets per node).

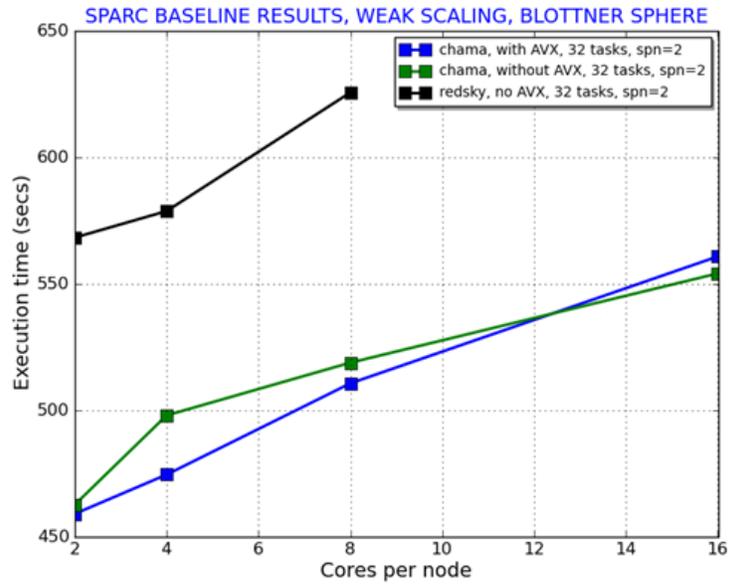


Figure 3. SPARC Blottner-sphere baseline results from RedSky and Chama showing execution time vs. cores per node.

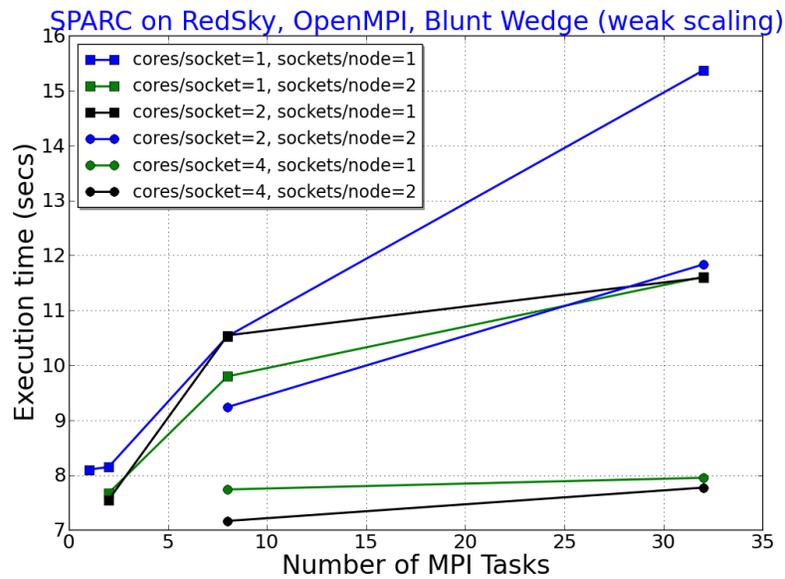


Figure 4. SPARC blunt wedge baseline results using OpenMPI.

Number of MPI ranks	Regular Order			Reordered		
	X	Y	Z	X	Y	Z
16	0.0	0.0	0.0	0.0	0.0	0.0
32	0.0	0.0	0.0	0.0	0.0	0.0
64	0.0	0.0	0.3	0.0	0.3	0.0
128	0.0	0.0	1.0	0.0	0.5	0.0
256	0.0	0.0	1.0	0.0	0.5	0.3
512	0.0	0.1	2.0	0.0	0.6	0.4
1024	0.0	0.3	2.1	0.2	1.0	0.7
2048	0.0	0.3	2.7	0.3	1.2	1.2
4096	0.0	0.3	3.7	0.3	1.2	1.2
8192	0.0	0.5	5.1	0.2	1.1	2.0
16384	0.0	0.5	4.9	0.2	1.1	2.2
32768	0.0	0.5	5.6	0.2	1.1	2.5
65536	0.0	1.1	10.2	0.2	1.6	2.8
131072	0.0	1.1	10.1	0.2	1.6	3.1

MINIGHOST AVERAGE HOP COUNTS ON CIELO

Figure 5. Z-hops reduction by re-ordering node assignment.

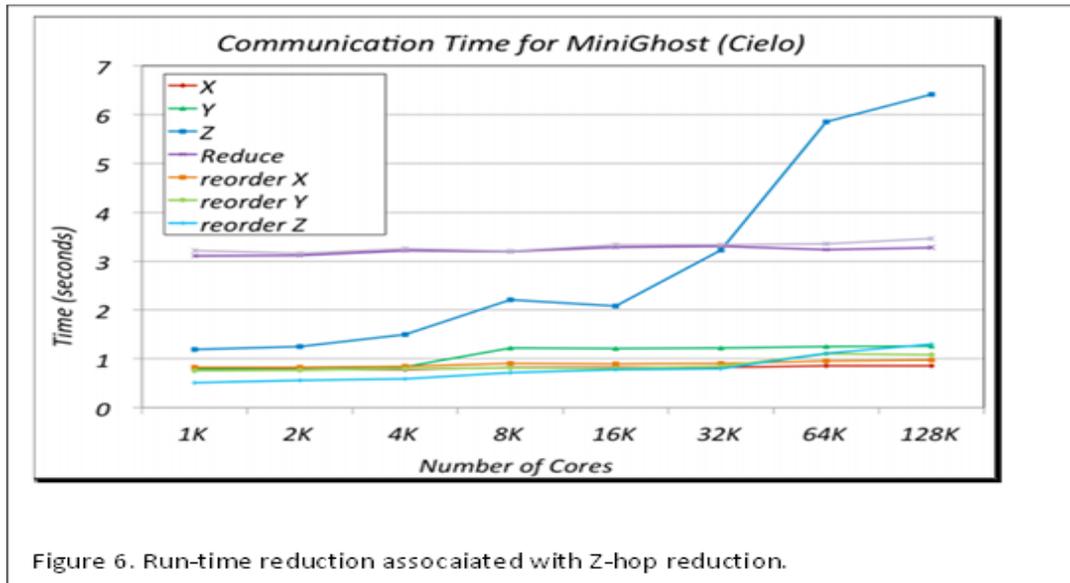


Figure 6. Run-time reduction associated with Z-hop reduction.

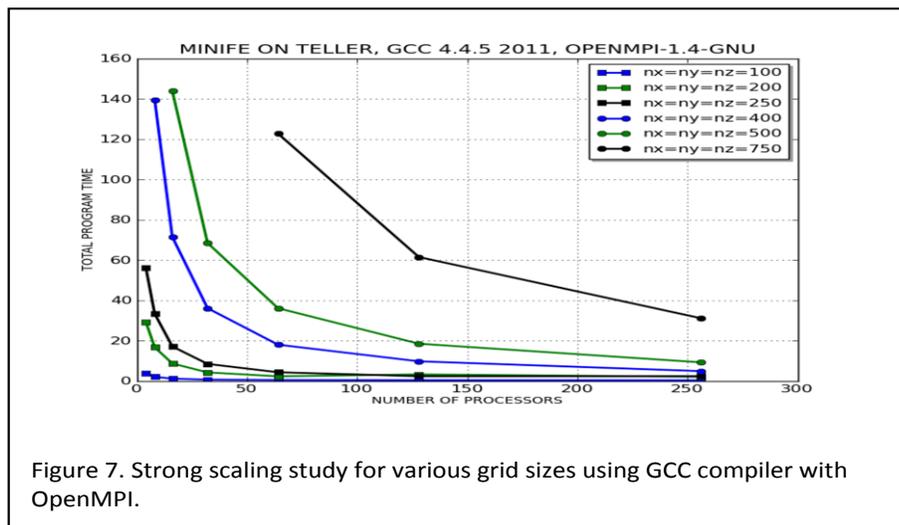


Figure 7. Strong scaling study for various grid sizes using GCC compiler with OpenMPI.

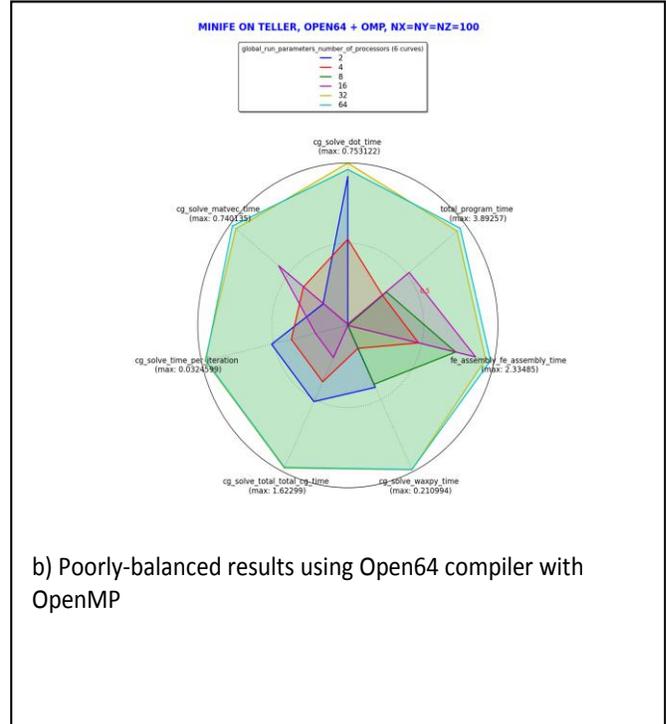
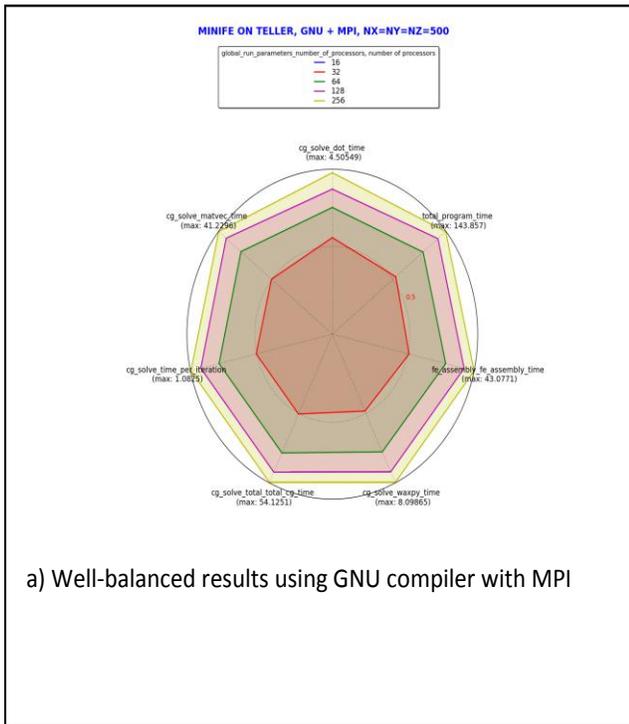


Figure 8. Kiviat diagrams illustrating degree at which the compiler and programming model balance with machine architecture during scaling.

