Unstructured-Grid CFD Algorithms on Many-Core Architectures

Aaron Walden*
Eric J. Nielsen
{aaron.walden,eric.j.nielsen}@nasa.gov
NASA Langley Research Center
Hampton, VA 23681, USA

Mohammad Zubair
Jason Orender
{zubair,jorender}@cs.odu.edu
Old Dominion University
Norfolk, VA 23529, USA

John C. Linford
Izaak Beekman
Samuel Khuvis
Sameer S. Shende
{jlinford,ibeekman,skhuvis,sameer}@paratools.com
ParaTools, Inc.
Eugene, OR 97405, USA

Justin P. Luitjens
jluitjens@nvidia.com
NVIDIA Corp.
Salt Lake City, UT 84101, USA

John G. Wohlbier
john.wohlbier@engilitycorp.com
HPCMP PETTT, Engility Corp.
West Bethesda, MD 20817, USA

ABSTRACT

In the field of computational fluid dynamics (CFD), the Navier-Stokes equations are often solved using an unstructured-grid approach to accommodate geometric complexity. Furthermore, turbulent flows encountered in aerospace applications generally require highly anisotropic meshes, driving the need for implicit solution methodologies to efficiently solve the discrete equations. These approaches require frequent construction and solution of large, tightly-coupled systems of block-sparse linear equations.

We explore the transition of two representative CFD kernels from a coarse-grained MPI-based model originally developed for multi-core systems to a shared-memory model suitable for many-core platforms. Results for the Intel Xeon Phi Knights Landing, NVIDIA Pascal P100, and NVIDIA Volta V100 architectures are compared with the aforementioned MPI-based implementation for the multi-core Intel Xeon Broadwell (BWL) processor. We observe substantial speedups over BWL as well as higher performance per dollar MSRP and performance per watt for the many-core architectures.

ACM Reference format:
Aaron Walden, Eric J. Nielsen, Mohammad Zubair, Jason Orender, John C. Linford, Izaak Beekman, Samuel Khuvis, Sameer S. Shende, Justin P. Luitjens, and John G. Wohlbier. 2017. Unstructured-Grid CFD Algorithms on Many-Core Architectures. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, CO, November 12–17, 2017 (SC '17)*, 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

*Also with Old Dominion University.

SC '17, Denver, CO

DOI: 10.1145/nnnnnnn.nnnnnnn

1 EXTENDED ABSTRACT

NASA Langley Research Center's FUN3D is an unstructured-grid computational fluid dynamics software suite used to tackle complex aerodynamics problems [2]. The toolset enables multidisciplinary capabilities through coupling to variable fidelity models encompassing structural effects, multibody dynamics, acoustics, radiation, optics, propulsion, and ablation. FUN3D provides advanced adjoint-based design capability, enabling formal optimization of time-dependent moving-body simulations involving turbulent flows. The adjoint formulation is also used to perform rigorous mesh adaptation and error estimation. FUN3D simulations annually consume more computational resources on NASA's Pleiades supercomputer than any other application [3]. The software is widely used to support major national research and engineering efforts at NASA and among groups across U.S. industry, other government agencies, and academia.

FUN3D currently relies on explicit domain decomposition and a coarse-grained MPI-based communication paradigm optimized for multi-core systems. Practical considerations guiding the development of future exascale-class systems impose severe power constraints on their design [7]. This next generation of high-performance computing systems is likely to rely on many-core architectures offering substantially improved energy efficiency through higher degrees of concurrency and lower clock rates. These systems are also expected to lower the amount of memory available per core. Ultimately, these and other factors are likely to render legacy MPI-based paradigms woefully inefficient, and it is widely recognized that such applications will require a transition to some form of hybrid parallelism to effectively leverage future exascale-class computing platforms. The current study examines two representative FUN3D kernels and their optimization for three many-core architectures, the Intel Xeon Phi Knights Landing (KNL) and the NVIDIA Pascal (P100) and Volta (V100) GPUs.

The two kernels chosen for the current study account for approximately 70% of wall time for the most common FUN3D simulations. We obtain node-level performance results for optimized versions of the kernels running on Knights Landing 7230 (in flat quadrant mode) using OpenMP parallelism and on Pascal P100 PCIE and

Volta V100 SXM using CUDA. These results are compared with those of optimized kernels running on a dual-socket 14-core Intel Xeon Broadwell (BWL) E5-2680v4 system (28 physical cores) using MPI, a benchmark representative of current practice. The input grid is specifically chosen so that all data fits into the high-bandwidth MCDRAM and HBM2 memories of KNL and P100/V100, respectively. Comparisons are established for both performance, performance per dollar MSRP, and performance per watt.

1.0.1 Kernel 1: Compressible Viscous Flux Jacobians. The first kernel computes the linearizations of the element-based compressible viscous fluxes on three-dimensional mixed-element grids [1]. This computation requires no intra-node communication; however, traversal of the unstructured grid, complexities related to the underlying physics, and irregular memory accesses associated with storing results present a number of challenges for many-core architectures. Threaded implementations of this kernel require a mechanism to avoid race conditions when updating shared variables. Atomics perform extremely well in this role on GPUs, but are prohibitively expensive using OpenMP on KNL. Instead, a coloring approach is used for KNL, which further exacerbates data locality issues and introduces a performance penalty of 30–60%.

Optimization of the flux Jacobian kernel for KNL revealed several strategies which proved beneficial for all three architectures. An extensive refactoring has been performed to reduce redundant computations of intermediate quantities. Loop extents previously dependent on element type are now held constant and prefetching has been introduced. To further improve memory performance, indirect addressing present in the baseline kernel has been reformulated as a lookup table. Combined, these modifications provided more than a 2× speedup over the baseline kernel on BWL/KNL. Optimizations improved the KNL colored kernel nearly 3×, but it still falls short of the MPI BWL benchmark with a speedup of 0.7.

Beginning with a CUDA C++ port of the optimized Fortran kernel, we observe speedups of 1.5 and 3.9 over BWL for P100 and V100, respectively, after extensive optimization. We auto-tune the number of threads per cell for each element type and use atomic updates to flatten a critical nested loop. This enhanced parallelization coalesces memory access patterns and reduces register and shared memory pressure, increasing occupancy, and driving shared memory bandwidth to approximately 12 TB/s on V100.

1.0.2 Kernel 2: Multicolor Point-Implicit Linear Solver. The second kernel is a linear solver based on a multicolor point-implicit scheme. In this approach, unknowns are colored using a greedy algorithm such that no two adjacent grid points map to the same color group of matrix rows. The kernel is dominated by a block-sparse matrix-vector product within each color. The matrix is stored in a modified compressed sparse row (CSR) format [6] in which the diagonal blocks are omitted. When using explicit domain decomposition with MPI, updated halo values must be exchanged prior to the start of each color. If the data is ordered appropriately, this communication may be effectively overlapped with the computation of interior values. Shared memory programming models avoid the need for such data movement [8].

Optimization of this kernel for KNL is straightforward, as the linear algebra is amenable to an implementation using AVX-512 vector intrinsics. We achieve 94% vectorization efficiency for all but a small remainder loop. Intrinsic prefetching is also used. A speedup of 2.5× over BWL is observed.

For GPUs, an implementation based on the existing *cuSPARSE* [5] and *cuBLAS* [4] libraries was originally developed. However, experiments have shown that the resulting performance is suboptimal for matrices representative of those encountered in actual simulations. Accordingly, custom optimized CUDA C++ implementations have been developed [8]. To perform a block-sparse matrix-vector product, a number of warps are allocated to process a subset of the blocks in a single row of the block-sparse matrix. Forward and backward substitutions using the diagonal block are performed using a single warp. Speedups of 2.8 and 6.1× over BWL are observed for P100 and V100, respectively.

1.0.3 Kernels 1 and 2 Combined. Adding the kernel run times, we observe speedups of 1.3, 2.1, and 5.2× over BWL for KNL, P100, and V100, respectively. With BWL as a baseline of 1.0, the relative performance per dollar MSRP we calculate for KNL, P100, and V100 is 2.3, 1.4, and 2.2, respectively. Relative performance per watt using manufacturer-provided thermal design power is 1.5, 2.1, and 4.1 for KNL, P100, and V100, respectively.

ACKNOWLEDGMENTS

This work was developed in part by the User Productivity Enhancement, Technology Transfer and Training (PETTT) Project No. PETA-KY07-001 and by DoE SBIR Grants DE-SC0009593 and DE-SC0017183. The authors would like to acknowledge the computational resources and PETTT software support from the DoD High Performance Computing Modernization office under Contract No. GS04T09DBC0017. Support from the NASA Langley Research Center Comprehensive Digital Transformation initiative and the Revolutionary Computational Aerosciences sub-project within the NASA Aeronautics Research Mission Directorate is also acknowledged. This work was in part developed at the 2017 ORNL Hackathon at NASA, which was a collaboration between and used resources of both the National Aeronautics and Space Administration and the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory. Oak Ridge National Laboratory is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The authors also wish to acknowledge Dr. Larry Davis of the DoD HPCMP Office for hardware resources used in this study.

REFERENCES

[1] Robert Biedron, Veer Vatsa, and Harold Atkins. 2005. Simulation of Unsteady Flows Using an Unstructured Navier-Stokes Solver on Moving and Stationary Grids. *23rd AIAA Applied Aerodynamics Conference* 5093 (06 2005). DOI: https://doi.org/10.2514/6.2005-5093

[2] Robert T. Biedron, Jan-Reneé Carlson, Joseph M. Derlaga, Peter A. Gnoffo, Dana P. Hammond, William T. Jones, Bil Kleb, Elizabeth M. Lee-Rausch, Eric J. Nielsen, Michael A. Park, Christopher L. Rumsey, James L. Thomas, and William A. Wood. 2017. *FUN3D Manual 13.1.* NASA/TM-2016-219580.

[3] Robert Hood. 2017. private communication. (2017). NASA Advanced Supercomputing Division.

[4] NVIDIA. 2015. cuBLAS User Guide. http://docs.nvidia.com/cuda/cublas/. (2015). Accessed 2016-08-01.

[5] NVIDIA. 2015. cuSPARSE User Guide. http://docs.nvidia.com/cuda/cusparse/index.html. (2015). Accessed 2016-08-01.

[6] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[7] Horst D. Simon. 2013. *Barriers to Exascale Computing.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–3. DOI: https://doi.org/10.1007/978-3-642-38718-0_1

[8] Mohammad Zubair, Eric Nielsen, Justin Luitjens, and Dana Hammond. 2016. An Optimized Multicolor Point-implicit Solver for Unstructured Grid Applications on Graphics Processing Units. In *Proceedings of the Sixth Workshop on Irregular Applications: Architectures and Algorithms (IAAA 2016).* IEEE Press, Piscataway, NJ, USA, 18–25. DOI: https://doi.org/10.1109/IA3.2016.9