50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
09 - 12 January 2012, Nashville, Tennessee

AIAA 2012-0564

A Hybrid Grid Compressible Flow Solver for Large-Scale Supersonic Jet Noise Simulations on Multi-GPU Clusters

Andrew Corrigan* K. Kailasanath† Junhui Liu‡ Ravi Ramamurti§

Douglas Schwer¶

Laboratory for Computational Physics and Fluid Dynamics

Naval Research Laboratory, Washington, DC 20375-5344, USA

Johann P.S. Dahm ‖

University of Michigan, Ann Arbor, MI 48109-2140, USA

A compressible flow solver for multi-GPU clusters has been developed for performing large-scale supersonic jet noise and other high-speed compressible flow simulations. Supersonic jet noise simulations require the accurate representation of complex nozzle geometry and thus the use of unstructured grids. While such a grid representation is crucial for the accurate representation of the nozzle geometry, it has the disadvantage of introducing a highly irregular memory access pattern, which violates GPU coalescing requirements resulting in an inefficient use of the otherwise high memory bandwidth of GPUs and therefore a bottleneck in computational performance. In order to mitigate this performance bottleneck, a hybrid grid representation is implemented which allows for augmenting the unstructured grid representation in the vicinity of complex nozzle geometry with an efficient structured grid representation in other regions of the flow domain, which is able to fulfill GPU coalescing requirements, and thus achieve a significant improvement in computational performance, leading to a reduction in simulation turnaround time.

I. Introduction

As more and more realistic geometrical features and operating conditions are simulated, the numerical resolution and corresponding grid size and time requirements for supersonic jet noise simulations are rapidly increasing. In order to reduce simulation turnaround time, and thus provide valuable results within an acceptable timeframe, it is necessary to ensure that current developments in computational architectures are exploited by production CFD codes. The purpose of this paper, therefore, is to investigate the application of GPUs for achieving faster turnaround times when performing large-scale supersonic jet noise and other high-speed compressible flow simulations, while developing new techniques to ensure that full performance is achieved. The code under consideration is the recently developed JENRE (Jet Engine Noise Reduction) code, which has been developed to run on multi-GPU clusters and other platforms, within a unified codebase.

Memory bandwidth is often observed to be a significant performance issue for unstructured grid solvers with low arithmetic intensity for which relatively few arithmetic operations are performed per memory transaction. Graphics processing units (GPUs) may help alleviate this issue and improve overall performance due to their high peak memory bandwidth in comparison to CPUs. For example, the NVIDIA Tesla C2070

*Research Mathematician, Laboratory for Propulsion, Energetics, and Dynamic Systems, Code 6041, AIAA Member.
†Head, Laboratories for Computational Physics and Fluid Dynamics, Code 6040, AIAA Fellow.
‡Mechanical Engineer, Laboratory for Propulsion, Energetics, and Dynamic Systems, Code 6041, AIAA Member.
§Aerospace Engineer, Laboratory for Propulsion, Energetics, and Dynamic Systems, Code 6041, AIAA Associate Fellow.
¶Mechanical Engineer, Laboratory for Propulsion, Energetics, and Dynamic Systems, Code 6041, AIAA Senior Member.
‖Graduate Research Assistant, Department of Aerospace Engineering, AIAA Student Member.

achieves a peak memory bandwidth of 144 GB/s[a], while the Intel Xeon 5670 achieves a peak memory bandwidth of 32 GB/s [b]. Such peak performance numbers do not automatically translate into achieved performance, and care is required in developing algorithms which can fully exploit available computational resources; otherwise significant losses in performance are to be expected.

There has been a substantial amount of work reported in the literature, which indicate that GPUs can achieve high performance for computational fluid dynamics applications, and that they are increasingly employed in mature, production codes. In previous work, an unstructured, cell-centered finite volume Euler solver was implemented, resulting in a substantial speed-up on previous-generation hardware.[1] Similar results, for a two-dimensional edge-based solver, were also presented by Dahm and Fidkowski.[2] Asouti, Kampolis, et al.[3,4] have also implemented a vertex-centered finite volume code for unstructured grids on GPUs. Finally, the general-purpose CFD code FEFLO was semi-automatically ported to run on GPUs[5] and GPU clusters,[6] resulting in a modest performance gain of up to 2x when running on GPUs in comparison to equivalent multi-core CPUs, for which the lack of an appropriate numbering scheme for unstructured grids was observed to be the primary impediment to achieving even higher gains in performance.

A great deal of research has been dedicated to the subject of unstructured grid numbering schemes for non-GPU architectures, such as cache-based, shared-memory parallel, and vector architectures, c.f., references.[7–14] These grid numbering schemes have already played a decisive role in reducing memory bandwidth consumption, and ultimately improving solver performance. However, while analogies can be made between previous architectures, especially vector processors, and modern GPUs, there are significant differences in the memory access requirements. Previously considered numbering schemes are often designed to maximize temporal reuse of data stored in cache. Although the current NVIDIA Fermi GPU architecture has L1 and L2 cache memory spaces, these GPU caches are not intended for temporal reuse in the way that CPU caches are, as reflected by their smaller size. In fact, performance guidelines advise against employing techniques like cache blocking and even suggest ignoring the presence of the cache when optimizing GPU code [15, Slide 11]. Therefore, techniques employed for minimizing cache misses, such as adjacency matrix bandwidth minimization, are not directly relevant to the memory access requirements of GPUs. In fact, such schemes may actually be harmful to achieving a memory access pattern suitable for GPUs, as it essentially leads to what has been called, "one of the worst access patterns for GPUs" [16, Slide 28]. Instead, achieving coalesced memory access should be used as the criterion when constructing unstructured grid numbering schemes, as it has been designated the "single most important performance consideration in programming for CUDA" [17, Section 3.2.1].

In order to reduce overall memory bandwidth consumption this work investigates transitioning from fully unstructured grids to hybrid grids, with the use of unstructured grids restricted to regions of the computational domain exhibiting complex geometry while structured grids are used in the remainder of the domain. This technique is effective for the meshes used for jet noise simulation: while unstructured grids are required in the vicinity of the jet nozzle, structured grids can easily represent the wake region of the domain. This results in a reduction of storage, and most importantly, memory bandwidth consumption, since the grid numbering is specified directly to fulfill GPU coalescing requirements.

II. Flow Solver

The Jet Engine Noise Reduction (JENRE) code implements a compressible flow solver based on the MILES approach (Monotonically Integrated Large-Eddy Simulation), which is under development for the simulation of supersonic jet flow and its acoustic properties. An example of such a flow field is shown in Figure 1, which shows a cross-section through the XY-plane of the three-dimensional temperature field computed by the JENRE code for a nozzle geometry consisting of 193 million cells. Practical supersonic jet noise simulations require increasingly detailed physics, complex geometry, large mesh sizes, and fast turn-around times, and therefore a major emphasis of this code's development is on ensuring that the code is capable of fully exploiting emerging massively parallel, high-performance computing (HPC) architectures.

The JENRE code implements a finite volume discretization of the Euler equations together with the Flux-Corrected Transport (FCT) convection scheme,[18] over cell-centered unstructured grids using arbitrary cell shapes. The multi-dimensional FCT flux limiter of Zalesak[19] provides an implicit sub-grid scale model, which ensures monotonicity at shocks and sharp features, with minimal artificial dissipation. As illustrated

[a] http://www.nvidia.com/object/personal-supercomputing.html

[b] http://ark.intel.com/Product.aspx?id=52576&processor=X5670

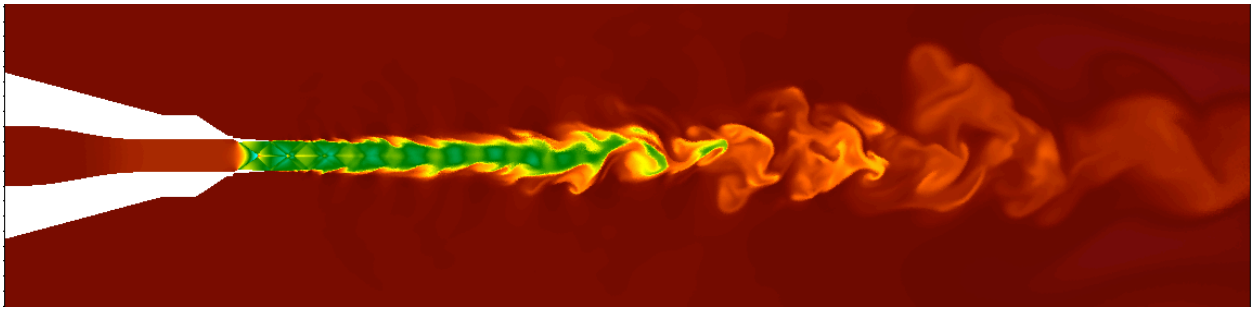

Figure 1. A cross-section through the XY-plane of the three-dimensional temperature field computed by the JENRE code for a nozzle geometry over an unstructured grid consisting of 193 million tetrahedral cells.

in Figure 2, face-based loops are used exclusively to exchange information across cells while performing flow calculations. Such face-based loops are used extensively in CFD, as they lead to a natural implementation of many flow calculations, such as limiting and flux integration while reflecting the symmetry inherent in such operations, and as a result both avoids redundant computation and enforces conservation laws at the discrete level. In the context of nodal finite element or finite volume schemes, such loops may be termed edge-based, but they exhibit a similar memory access pattern, with the exception of edge-based connectivity leading to greater variation in the degree of connectivity between nodes in comparison to that of the face-based connectivity between cells. These loops follow a gather-scatter memory access pattern and require a face-coloring scheme when executed in parallel. Such a coloring scheme organizes faces sharing an adjacent cell into separate color groups, so that all faces of the same color may be safely executed in parallel. Consistently using face-based loops in the flow solver, increases the code's amenability to optimization, since there is only a single non-trivial memory access pattern to target when constructing the face and cell numbering. One of the overarching goals of the JENRE code, in addition to numerical and physical accuracy and usability, has been to achieve the highest computational performance possible. To this end, JENRE is designed to exploit the parallelism inherent in modern computer architectures at multiple levels.

The first form of parallelism achieved by JENRE is coarse-grained, distributed-memory parallelism. Such parallelism is implemented using the common approach of peforming domain decomposition and using MPI to communicate between sub-domains.[20] The particular approach used to implement this in JENRE is by using processor boundary conditions, which are implemented in the same way as physical boundary conditions, but instead make MPI calls to exchange field data across inter-processor cell faces. In addition, JENRE provides parallel IO support using MPI-IO and on-the-fly parallel partitioning using the open-source ParMETIS library.[21]

The second form of parallelism achieved by JENRE is fine-grained, shared-memory parallelism. Because of the gather-scatter memory access pattern present in the face-base loops of JENRE, a face-coloring scheme is employed which serializes execution over the color groups, while exploiting shared-memory parallelism over the faces contained within each color group. Such parallelism is implemented via the open-source Thrust library,[22,23] which provides generic algorithms (e.g., copy, transform, reduce, compress, prefix-sum, sort), function objects, and iterators, which are used as a foundation upon which the mesh and flow solver operations in JENRE are implemented. Since the Thrust library is optimized for either a serial, OpenMP, or CUDA backend, JENRE can be compiled to produce either an optimized serial CPU code, multi-core CPU code, or many-core GPU code, and could easily be extended to future computational architectures that might emerge, while also remaining flexible enough to accommodate any custom, architecture-specific implementations required for high performance. Finally, these two forms of parallelism are completely orthogonal, and as a result JENRE can be compiled to run on both clusters of multi-core CPUs and multi-core GPUs.

III. Memory Bandwidth

III.A. Coalescing requirements

Achieving coalesced memory access is considered to be the "single most important performance consideration in programming for the CUDA architecture".[17] Coalescing shares the purpose of maximizing cache hits on

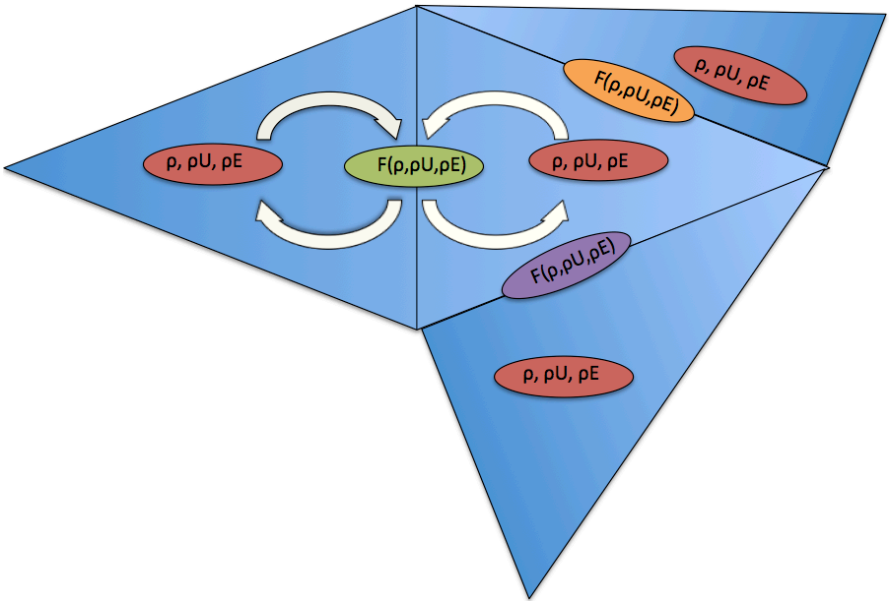

Figure 2. An illustration of the gather-scatter pattern of the face-based loops, with a face coloring scheme applied, which is implemented in the JENRE code and used for operations such as exchanging fluxes and performing limiting.

CPUs, i.e., reading memory in low-latency cache in order to reduce slower off-chip memory access. Since GPUs service memory transactions in 32, 64, or 128 byte segments, coalesced memory access minimizes the number and size of segments required to fully service a particular global memory transaction for a given warp (32 consecutive threads). Given a particular array access in a CUDA kernel, coalesced memory access means that the threads of a warp access a contiguous, aligned segment of memory (the CUDA Programming Guide[24] provides full technical details). If the threads of a warp were to access memory in completely disparate locations, then each thread would require a separate segment to be read from or written to in global memory leading to an enormous degradation in performance [17, Figure 3.9]. Furthermore, this additional memory bandwidth consumption will mostly go to waste. Hardware up until the latest Fermi architecture discarded unused portions of these segments. While the latest generation of GPUs introduces a cache which can retain previously read segments on-chip, coalescing remains of the utmost importance for achieving high performance.

In the field of computational fluid dynamics, GPU coalescing requirements are primarily a concern in the context of face or edge based loops, which often dominate the overall run-time of flow solvers as they are used to implement fundamental operations such as flux integration and limiting. As illustrated in Figure 2, the memory access pattern is that of gathering flow quantities from the two cells adjacent to each face, performing computation such as interpolation or flux integration and then scattering the result back to each adjacent cell. The JENRE code, like many other CFD codes, parallelizes such loops over the faces. We denote the first cell accessed by each face as the "left" cell, while the second cell is denoted the "right" cell. While an unstructured grid will always involve some amount of irregularity in the memory access pattern, using a structured grid, the cells can be numbered such that they increase with a unit stride between the left cell of each face, as well as between the right cell along consecutive faces. Often CPU numbering schemes attempt to achieve good cache behavior by minimizing the jump between adjacent cells in order to maintain temporal locality within face loops as flow quantities stored at the cell centers are accessed. Instead the GPU numbering scheme ignores the jump in index across faces, and focuses on minimizing the jump in index between the left cells accessed by each consecutive warp of faces, and then the right cells accessed by each consecutive group of faces, in order to ensure that coalesced memory access is achieved.

III.B. Structured Grids

The technique considered in this work to reduce overall memory bandwidth consumption is to transition from fully unstructured grids to hybrid grids, with the use of unstructured grids restricted to regions of

the computational domain exhibiting complex geometry, with structured grids used in the remainder of the domain. This results in a reduction of storage, and most importantly, memory bandwidth consumption, since structured grid connectivity is computed on-the-fly, rather than being read from memory. An additional benefit is that the optimal grid numbering dictated by GPU coalescing requirements can be specified directly, which further reduces memory bandwidth consumption. In the case of the jet nozzle run illustrated in Figure 1, a tetrahedral unstructured grid encloses the nozzle geometry, while the wake region of the mesh is represented using a hexahedral structured grid. The JENRE code shares a unified flow solver for each case, with the specialization to each mesh representation performed at compile-time. By using structured grids in the wake region, not only is memory bandwidth consumption reduced and better coalescing achieved, the amount of flow data storage is reduced, which is important for GPUs since currently available hardware only provides up to 3-6 GB of memory per GPU, and therefore increases the problem sizes accessible to currently available GPU-based systems.

IV. Results

Benchmark runs have been performed on the Mayhem cluster at the Laboratory for Computational Physics and Fluid Dynamics at the Naval Research Laboratory. Each node on the Mayhem cluster consists of dual six-core Intel X5650 Xeon CPUs, together with dual NVIDIA Fermi GPUs, with some nodes containing Tesla C2050 GPUs and others containing Geforce GTX 480 GPUs.[25] JENRE was used to calculate the Tanna nozzle flow on a mesh consisting of 13.5 million cells, which serves as an appropriate example for benchmarking performance. Running this benchmark calculation, JENRE achieves on average a speed of 15.76 million tetrahedral cells per second per time-step per GPU. In comparison, the Intel Xeon CPU achieves on average a speed of 7.37 million tetrahedral cells per second per time-step per six-core-CPU. Therefore, the current performance advantage of using NVIDIA GPUs instead of a comparable six-core Intel CPU is a factor of 2.14x in the case of unstructured grids. This indicates that GPUs already provide a modest performance advantage over CPUs, despite the fact that the CPU-tailored reverse Cuthill-McKee numbering scheme was employed for both architectures.

To benchmark the performance of structured/hybrid grids using a coalesced grid numbering scheme, an additional benchmark was run for a thirty-eight million hexahedral cell mesh corresponding to the wake region of the flow field beginning from a distance of three nozzle diameters, and extending to a distance of fifty nozzle diameters downstream. In this case, JENRE achieves on average a speed of 24 million hexahedral cells per second per time-step per GPU. In comparison, JENRE achieves on average a speed of 4 million hexahedral cells per second per time-step per six-core-CPU. Therefore, a six-fold increase in computational performance is achieved when using NVIDIA GPUs instead of comparable six-core Intel CPUs. It is interesting to note that the performance advantage of running on GPUs is roughly proportional to the difference in memory bandwidth, which is to be expected for memory bandwidth bound applications. On a per-cell basis, hexahedral cells are slightly more expensive to compute than tetrahedral cells since they contain more faces per cell, but have the advantage of being able to resolve a given spatial resolution using a lower number of cells.

V. Conclusions and Outlook

The Jet Engine Noise Reduction code is able to achieve high performance across a variety of computational platforms, while being developed under a unified codebase. When using unstructured grids, JENRE achieves a modest two-fold increase in computational performance running on GPUs in comparison to running on comparable six-core CPUs, and more than a six-fold speed-up when using a structured grids. Such a substantial performance gain was made possible by using a specialized grid numbering scheme which meets GPU coalescing requirements, and therefore makes optimal use of the high memory bandwidth offered by GPUs in comparison to CPUs. JENRE's ability to combine both unstructured and structured grids enables it to achive an appropriate balance of geometric flexibility using unstructured grids, while achieving the highest possible performance using structured grids. This performance advantage will serve to significantly reduce simulation turnaround time for solving large-scale supersonic jet noise simulations moving forward.

Acknowledgements

This work sponsored by ONR through the Jet Noise Reduction Project under the NIHL Program and The NRL 6.1 Computational Physics Task Area.

References

[1]Corrigan, A., Camelli, F., Löhner, R., and Wallin, J., "Running unstructured grid-based CFD solvers on modern graphics hardware," *Int. J. Numer. Meth. Fluids*, Vol. 66, May 2011, pp. 221–229, Published online February 2010.

[2]Dahm, J. and Fidkowski, K., "Employing Coprocessors to Accelerate Numerical Solutions to the Euler Equations," 2009, Retrieved from http://www.johanndahm.com/research.php.

[3]Asouti, V., Trompoukis, X., Kampolis, I., and Giannakoglou, K., "Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units," *Int. J. Numer. Meth. Fluids*, 2010.

[4]Kampolis, I., Trompoukis, X., Asouti, V., and Giannakoglou, K., "CFD-based analysis and two-level aerodynamic optimization on graphics processing units," *Computer Methods in Applied Mechanics and Engineering*, Vol. 199, No. 9–12, January 2010, pp. 712–722.

[5]Corrigan, A., Camelli, F., Löhner, R., and Mut, F., "Semi-Automatic Porting of a Large-Scale Fortran CFD Code to GPUs," *Int. J. Numer. Meth. Fluids*, May 2011, Published online.

[6]Corrigan, A. and Löhner, R., "Semi-automatic porting of a large-scale CFD code to multi-graphics processing unit clusters," *Int. J. Numer. Meth. Fluids*, August 2011, Published online.

[7]Löhner, R., "Some useful renumbering strategies for unstructured grids." *International Journal for Numerical Methods in Engineering*, Vol. 36, 1993, pp. 3259–3270.

[8]Löhner, R., "Renumbering Strategies for Unstructured-Grid Solvers Operating on Shared-Memory, Cache-Based Parallel Machines," *13th AIAA Computational Fluid Dynamics Conference*, June 1997, AIAA-1997-2045.

[9]Löhner, R., "Renumbering Strategies for Unstructured-Grid Solvers Operating on Shared-Memory, Cache-Based Parallel Machines," *Comp. Meth. Appl. Mech. Eng.*, Vol. 163, 1998, pp. 95–109.

[10]Anderson, W., Gropp, W., Kaushik, D., Keyes, D., and Smith, B. F., "Achieving High Sustained Performance in an Unstructured Mesh CFD Application," *Supercomputing 1999, IEEE Computer Society*, November 1999.

[11]Gropp, W., Kaushik, D., Keyes, D., and Smith, B., "Achieving high sustained performance in an unstructured mesh CFD application," *Supercomputing 00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, November 2000.

[12]Fouladi, N., Darbandi, M., and Schneider, G., "A Directional Renumbering Strategy for Improving Unstructured Grid Data Structure," *48th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, January 2010, AIAA-2010-1266.

[13]Aubry, R., Houzeaux, G., and Vàzquez, M., "Some Useful Strategies for Unstructured Edge-Based Solvers on Shared Memory Machines," *International Journal for Numerical Methods in Engineering*, Vol. 85, 2010, pp. 3259–3270.

[14]Aubry, R., Houzeaux, G., and Vàzquez, M., "Some Useful Strategies for Unstructured Edge-Based Solvers on Shared Memory Machines," *49th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, January 2011, AIAA 2011-614.

[15]Micikevicius, P., "Fundamental Optimizations," *Supercomputing 2010, Tutorial S03, High Performance Computing with CUDA*, November 2010.

[16]Micikevicius, P., "Analysis-Driven Optimization," *Supercomputing 2010, Tutorial S03, High Performance Computing with CUDA*, November 2010.

[17]NVIDIA Corporation, "NVIDIA CUDA 4.0 Best Practices Guide," 2011.

[18]Boris, J. P. and Book, D. L., "Flux-corrected transport. I. SHASTA, a fluid transport algorithm that works," *Journal of Computational Physics*, Vol. 11, No. 1, 1973, pp. 38 – 69.

[19]Zalesak, S. T., "Fully multidimensional flux-corrected transport algorithms for fluids," *Journal of Computational Physics*, Vol. 31, No. 3, 1979, pp. 335 – 362.

[20]The MPI Forum, "MPI: A Message-Passing Interface Standard," July 2011, retrieved from `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf`.

[21]Karypis, G. and Schloegel, K., "ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 4.0," August 2011, retrieved from `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`.

[22]Hoberock, J. and Bell, N., "Thrust: A Parallel Template Library," 2011, Version 1.4.0.

[23]Bell, N. and Hoberock, J., "Thrust: A Productivity-Oriented Library for CUDA," *GPU Computing Gems: Jade Edition*, Morgan Kaufmann, 2011, pp. 359–372.

[24]NVIDIA Corporation, "NVIDIA CUDA 4.0 Programming Guide," 2011.

[25]Obenschain, K., Corrigan, A., and Patnaik, G., "Performance of Unstructured Finite Volume Code on a Cluster with Multiple GPUs per Node," *49th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, January 2011, AIAA-2011-0945.