Eleventh International Conference on CFD in the Minerals and Process Industries
CSIRO, Melbourne, Australia
7-9 December 2015

GPU PROGRESS AND DIRECTIONS IN APPLIED CFD

Stan POSEY[1*], **Simon SEE**[2], **and Michael WANG**[2]

[1] NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050, USA
[2] NVIDIA Singapore Pte Ltd, Robinson 112 #05-0, 068902, SINGAPORE
*Corresponding author, E-mail address: sposey@nvidia.com

ABSTRACT

Current trends in high performance computing include the use of graphics processing units (GPUs) as massively-parallel co-processors to CPUs in order to accelerate numerical operations common to computational fluid dynamics (CFD) solvers. This paper examines GPU characteristics for various CFD methods and provides examples of current implementations for commercial CFD software. In order to increase adoption of GPUs for commercial CFD, a linear solver library called AmgX was developed by NVIDIA that offers an algebraic multigrid (AMG) solver and other features, with parallelization of all phases of AMG including hierarchy construction and ILU factorization and solve. Examples relevant to CFD practice are investigated in order to demonstrate the use of AmgX in ANSYS Fluent for industry-scale applications. Hardware system configuration is also discussed that examines directions on CFD solver development.

INTRODUCTION

The efficient use of hardware system resources and CFD simulation turn-around times continue to be important factors behind engineering decisions to expand CFD as a technology to support product design. While the progress in recent years of CFD simulation verses physical experimentation has been remarkable, the availability of inexpensive workstations and clusters with conventional multi-core CPU parallel solvers has not been enough to motivate broad industry deployment of high fidelity modelling and use of design optimization procedures.

Recent developments by commercial vendors of CFD software aim to achieve GPU speedups from fine-grain and second-level parallelism under an existing distributed memory, first-level CPU parallelism. For most vendors their GPU implementations have included implicit sparse iterative solvers that utilize a hybrid CPU-GPU computing scheme. In this way, matrix operations which are usually only sent to CPU cores for processing, are characterized before processing and appropriately overlapped across both CPU cores and GPU resulting in overall job acceleration. Figure 1 illustrates a schematic of such a hybrid scheme where a proper measure of speedup is made by comparison of the same simulation on multiple CPU cores both with and without GPU acceleration.

The continual increase in processor speeds is limited due to power and thermal constraints, and in order to achieve gains in performance without increasing clock speeds, application software parallelism must be implemented. This parallelism can come in the form of task parallelism, data parallelism, or a combination of both. Although parallel applications that use multiple cores are a well established practice in CFD, a recent trend towards the use of GPUs to accelerate multi-core CPU computations is increasingly common.

In this heterogeneous computing model the GPU serves as a co-processing accelerator to the CPU. The need for high performance and the parallel nature of CFD algorithms has led GPU developers to create designs with hundreds of cores. Today GPUs and software development tools are available for implementing more general algorithms that use the GPU for applications in a range of scientific and engineering domains where numerical computations are expected to complete as fast as possible.

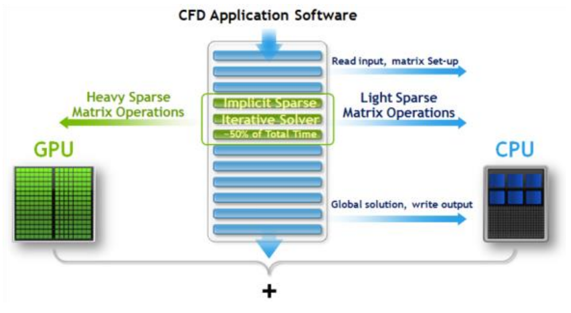
CFD Application Software
Read input, matrix Set-up
Heavy Sparse Matrix Operations
Implicit Sparse Iterative Solver ~50% of Total Time
Light Sparse Matrix Operations
GPU
CPU
Global solution, write output
+

Figure 1: Schematic of GPU acceleration for implicit sparse solvers typical of commercial CFD software.

During recent years, CFD has become increasingly reliant on clusters of multi-core CPUs to enable more detailed simulations within the limits of engineering project times. For this reason, the scalability of a solver across multiple servers has become equally important as single-processor performance. Most CFD developments during that period relied strictly on compilers to achieve a nominal level of single process performance optimization, and development investments went towards distributed memory parallel. GPU parallelism has brought back the focus on single-processor performance beyond what is possible with conventional CPU compilers, and with emphasis on multi-GPU scalability in an HPC cluster environment.

Another important processor trend is relative performance of floating point operations vs. memory bandwidth. Over the past few decades the gap has extended to more than three orders of magnitude. CPU designs have gone to great lengths to bridge the gap between processor and memory performance by introducing instruction and data caches, instruction level parallelism, and other technical advances. And although GPUs offer a different approach in terms of hiding memory latency because of their specialization to inherently parallel problems, the fact remains that processor performance will continue to advance at a much greater rate than memory performance. An extrapolation without any fundamental changes in memory designs, processors will become infinitely fast relative to memory, and performance optimization will become solely an exercise in optimizing data movement.

Algorithm Suitability for GPUs

At the highest level there are two considerations that can determine optimum suitability of a CFD algorithm for GPUs: the selected time integration method and spatial discretization scheme. CFD time integration methods are either explicit or implicit, and discretization schemes either structured or unstructured mesh. Successful GPU implementations exist for these high level algorithm characteristics, but the combination of some are more favourable for CFD software vs. others. For example structured grids offer uniform memory reference access during computations which can be a better fit for GPUs vs. unstructured, although breakthroughs in recent years with renumbering techniques have greatly improved the situation for unstructured grids which are more popular because of flexibility with meshing complex geometries.

The schematic in Figure 2 provides a profile description of select algorithm combinations, and both CFD and computational structural mechanics (CSM) are shown because of their close relationship in development and deployment. Explicit methods are used in CFD mostly for compressible flows and in CSM for high frequency impact loads where very small time steps may be necessary for proper resolution of a given simulation. Implicit methods are used for incompressible CFD but can also be used for compressible, and in CSM for low frequency loads such as structural vibration and static stress. The choice of discretization scheme is usually dependent upon the geometric complexity of the given simulation objectives. Practically speaking, nearly all commercial CFD uses finite volume unstructured or finite element, all CSM uses finite element, and CFD for structured grids is typically limited to non-commercial, compressible aerodynamics.

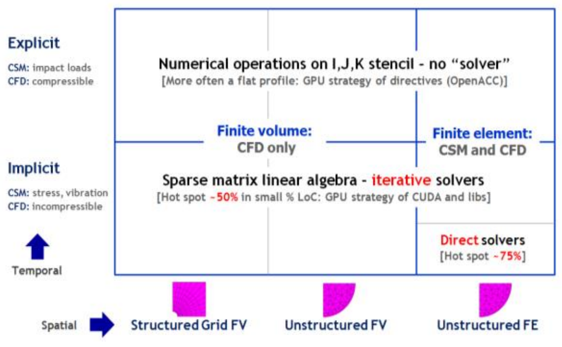

Figure 2: Profile description of suitability for CFD algorithm selections in a GPU computing environment.

Industry deployment of CFD and CSM for application use usually means legacy software that undergoes several years of verification and validation, and which comprises a large software code base with diverse teams of developers. These conditions significantly impact algorithm suitability and more importantly influence a given strategy for GPU development. For example, CFD and CSM explicit methods on structured and unstructured grids with ordered stencils are very well suited to GPUs, but only those based on recent GPU-architected developments have been successful. Meanwhile legacy explicit methods are typically order 1M lines of code (LoC) and exhibit very flat execution profiles, meaning that the entire code base would need to run on the GPU for any benefit to an industry-scale simulation. This situation is impractical from a developer view point and can only be successful with a GPU-based compiler approach such as OpenACC.

Implicit sparse solvers are the current GPU focus of most commercial CFD and CSM implementations owing to their favourable execution profile of a 'hotspot' and the simpler requirement that a small % of LoC must run on the GPU for potentially significant gains. Unlike explicit methods based on nearest neighbour computations of I,J,K stencils where each stencil computation might be a single thread on a GPU, implicit methods use matrix linear algebra computations and usually provide higher computation intensity, or more work for the GPU per memory movement of data. The solver hotspot of implicit methods are typically very good candidates for moving onto the GPU. Specifically these are the linear equation solvers which either use a direct or iterative method. Because they are usually a very small % of LoC, they are ported with CUDA and often make use of libraries such as CUBLAS and others, which provide tuned computational kernels.

Direct solvers which are used only by CSM software rely on double precision matrix-matrix (DGEMM) operations and might consume an average of 75% of the total profile time (depending on the model), which is a very good candidate for GPU acceleration. Effective acceleration of 75% of the total job time can mean a 3x speed-up for an overall simulation. These are the solvers found in implicit finite element CSM codes (Abaqus, ANSYS, etc.) and which were the first to demonstrate the best speed-ups on GPUs. These methods require extremely large system memory and are not used in CFD. Iterative methods rely on sparse matrix-vector operations (SpMV) and usually exhibit an average of 50% of the total profile time. These methods require much smaller memory capacity and are used by all CFD and some CSM software, and have become a recent focus for GPUs by commercial software vendors because of breakthroughs in matrix preconditioning, the ease in GPU deployment of parallel conjugate-gradient solvers, and the introduction of multigrid solvers on GPUs.

GPU-PARALLEL CFD

Parallel iterative sparse solvers are widely used in CFD for simulations that deploy implicit schemes. Iterative solvers are the standard for commercial CFD software, owing to their efficiency in computation and storage, and the need to mostly resolve incompressible flow fields. A GPU-parallel implementation of an iterative solver such as a

Krylov-based preconditioned conjugate gradient or multigrid method would rely, among other features, on kernel performance of sparse-matrix-vector-multiply (SpMV) and a few additional BLAS-1 kernels.

Because of the low arithmetic intensity of the SpMV kernel, it is highly memory-bound, and therefore optimization of the memory access pattern to achieve peak GPU memory-bandwidth is a primary consideration. Even though the data access pattern is irregular for general SpMV, with careful re-design of data structures for the sparse matrices, it is possible for the SpMV kernel to run very close to the peak of the GPU memory bandwidth.

With an efficient SpMV kernel, an iterative method can be implemented fully on a GPU, however preconditioners are usually required. All contemporary CFD iterative solvers use a preconditioner to speed-up the solution convergence rate, and this would also need to be implemented on the GPU for overall performance. The difficulty of this implementation varies significantly depending on the specific preconditioner used. For example, a GPU-efficient Jacobi preconditioner would be very simple to implement, however a highly sequential preconditioner such as an incomplete Cholesky scheme, would be difficult for good parallel efficiency. This trade-off may motivate a redesign of a particular preconditoner in order to run massively parallel on a GPU.

Linear equation solvers that use a Krylov method such as a preconditioned conjugate gradient (PCG) or generalized minimal residual method (GMRES) have been successfully implemented for GPUs. The use of multigrid methods, and in particular algebraic multigrid (AMG) for linear solvers has been a relatively new area of exploration for GPUs, and has become an emerging development for commercial CFD software.

In order to assist in the CFD community's range of GPU development interests, NVIDIA continues to invest in high performance iterative solvers in several areas:

- cuBLAS library of basic linear algebra subroutines (BLAS) for dense matrices
- cuSPARSE library of BLAS, kernels, and solver components for sparse matrices with variety of formats
- Contributions toThrust, and open source C++ template library of high performance parallel primitives
- AmgX: development of a complete linear solver library with emphasis on multigrid methods

The remainder of this paper will describe the implementations of AmgX for solving linear scalar and coupled systems of equations on GPUs. AMG is optimal for elliptic-type or coupled elliptic-dominant partial differential equations (PDEs) discretized over irregular grids. During the past decade AMG has evolved as the linear solver standard in the commercial CFD software community, and now benefits from GPU acceleration.

NVIDIA AMGX LIBRARY

The AmgX solver library developed by NVIDIA provides two kinds of AMG: aggregation-based and selection-based, also called classical aggregation. Each approach is superior for different types of problems, allowing an application to use the method that best matches its needs. These AMG developments were also optimized for overall time-to-solution, including both "setup" and "solve" phases, making it suitable for use inside an iterative non-linear solver such as Newton or Picard. In cases where the matrix structure is unchanged from one call to the next, the setup phase can run even faster by reusing this information.

The implementation of algebraic multigrid on GPUs has already been considered by several authors. Early attempts focused on the simple off-loading of sparse matrix-vector multiplications to the GPU during the solve phase of the algorithm. Later, others have shown how to expose fine-grained parallelism within a single node in the setup and solve phases of the aggregation-based algorithm using libraray primatives from Thrust.

The AmgX library can be seen as a super set of the listed algorithms. It implements both classical and aggregation based algebraic multigrid methods with different selector and interpolation strategies. It also contains many of the standard preconditioned Krylov subspace iterative methods with a variety of smoothers and preconditioners, including block-Jacobi, Gauss-Seidel, and incomplete-LU (ILU). A highlight of the library is the full congurability of a solver hierarchy with arbitrary depth, in which the outer solver uses inner solvers as preconditioners, which themselves can also be preconditioned by other methods. This allows the user to quickly experiment with variety of inner-outer schemes often discussed in the literature. Any of these methods can be used in combination with AMG to create a rich set of numerically powerful solvers.

In contrast to earlier approaches, the AmgX library also takes advantage of multiple-GPUs and allows handling of very large sparse linear systems that fit into the aggregate memory of all GPUs present in the system. Within a single node, in the setup phase we rely on existing parallel graph matching techniques such as Metis and Scotch, while the smoothers and preconditioners take advantage of parallel graph coloring algorithms. In a distributed environment, the AmgX library relies on graph partitioning algorithms and uses techniques based on rings of nearest neighbors to keep track of communication. In this setting, only the required halo elements are communicated across different nodes. The latency of these transfers is hidden by overlapping communication and computation. Moreover, if the global problem becomes too small to fill all active GPUs with work, consolidation onto fewer GPUs is performed, which again allows the library to minimize communication costs while fully taking advantage of computational resources at hand.

We will discuss this in the following section that describes the ANSYS Fluent implementation, where it is shown that the aggregation based algebraic multigrid algorithm implemented in AmgX achieves a greater than 5x speedup on a single GPU vs. the commercial proprietary ANSYS implementation on a single CPU scocket. The approach also scales well across multiple nodes sustaining this performance advantage.

The ability to mix and match different algorithms from the AmgX library to create a variety of solvers is possible because of the flexible plug-in architecture. For example, any smoother can be paired with any AmgX solution algorithm, and it is possible to implement these custom algorithms and solvers through a plug-in interface. This and other product-quality features were motivated by the requirements from commercial CFD software vendors for ease in linear solver integration.

ANSYS Fluent Implementation

An example of the GPU-accelerated AMG solver for commercial CFD software is ANSYS Fluent, provided by ANSYS Inc. with headquarters in Canonsburg, PA, USA. ANSYS Fluent is based on the finite volume method and is a multi-purpose fluids and thermal analysis software for a range of flow conditions and simulations. It is widely deployed by research organizations, and commercial industry that develop products for automotive, aerospace, power-generation, consumer products, and defense applications, among others.

At the core of ANSYS Fluent lies the solution of large sparse linear systems, which often take the largest percent of the total simulation time. For example, we are usually interested in solving a set of linear systems where these systems are often constructed as part of the non-linear process for a number of steps, where either the coefficient matrix changes at every iteration, or the matrix does not change, but the right-hand-side and the solution do change from one step to the next.

In realistic cases, these systems are often so large that they do not fit into memory of a single node and large clusters must be used to find the solution. In this setting direct solution of these linear systems becomes prohibitively expensive from the memory stand point and also does not scale well with the increased number of nodes. Therefore, multigrid and iterative methods that can attain a lower accuracy solution much faster are often preferred by commercial CFD software vendors including ANSYS.

Development of a GPU-accelerated ANSYS Fluent was implemented through a technical collaboration between NVIDIA and ANSYS. Engineers from each organization first sought to characterize the execution profile of ANSYS Fluent on a conventional CPU before deciding which tasks in a simulation would be best suited for GPU-acceleration. A study was conducted for different models with a range of cell types and flow conditions, and for either the segregated or coupled solution scheme for the pressure-based Navier-Stokes (PBNS) equations.

Both the segregated and coupled schemes require a substantial amount of time to assemble the linear system of equations before a solution of that system, and these procedures are required at each iteration, often requiring 1000’s for a fully converged solution. On average the models using the coupled scheme exhibited a profile that led to about 65% of time spent in the PBNS solution as shown in Figure 3 whereas the models using the segregated scheme only averaged about 30% for the same solution. The coupled approach is recommended by ANSYS over segregated owing to its superior convergence behavior, and therefore became the initial focus for the NVIDIA and ANSYS GPU-acceleration project.

As a result of the technical collaboration, the GPU-based AMG solver was developed as a plug-in library, and ANSYS developed additional custom functionality in the linear solution process. These ANSYS custom functions can override a NVIDIA-developed feature such as a smoother, an aggregation scheme, or even an inter-node communication protocol. This “override” approach allows NVIDIA to provide plug-in updates to their contributions of the AMG solver, while preserving the ANSYS customer functionality in a seamless way.

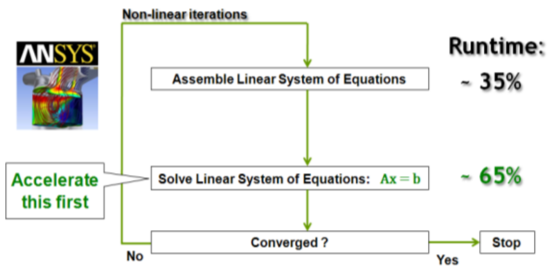

Figure 3: Execution profile of ANSYS Fluent for a single iteration of coupled pressure-based Navier-Stokes solver.

Performance and parallel efficiency for ANSYS Fluent is dependent upon many factors including hardware system architecture, and model geometry and flow and boundary conditions of a simulation. Fluid simulations in ANSYS Fluent often contain a mix of finite volume cell types and fluid properties that can exhibit substantial variations in computational expense, but ANSYS Fluent has a proven and well documented ability to efficiently scale to a very large number of CPU cores.

The two models used for the GPU performance studies were based on the same geometry and case for external aerodynamics of a truck body. This model is a standard benchmark case provided by ANSYS that is considered relevant to industry-scale practice. The model consists of steady RANS conditions using a detached-eddy (DES) turbulence model, and the coupled PBNS solver is applied. The model has mixed cell types but is tetrahedral cell dominant, and was configured with (i) a case of 14M cells, and (ii) a case of 111M cells. The first study of 14M cells was conducted with the latest ANSYS Fluent 16.0 on a small cluster system consisting of 2 server nodes, each configured with:

- 2 x Intel Xeon E5-2697v2 @2.7GHz with 12 cores each, for a total 24 cores (48 cores total system)
- 2 x NVIDIA Tesla K80 GPUs (4 GPUs total system)

The current ANSYS Fluent 16.0 release supports GPU acceleration for the coupled PBNS solver, and similar to the CPU-only AMG solver, there are solver settings that can be custom configured in order to achieve the optimal performance for a particular model and/or flow condition. For these studies the optimal solver settings were used for each, and it should be noted for this case that such settings for CPU-only and CPU+GPU are most often different.

The 14M cell results shown in Figure 4 give performance comparisons of the CPU time both stand-alone and with GPU acceleration. The metric used in the performance is number of seconds for a fully converged solution for both the AMG solver only, and total solution time. The timings show more than a 5x speedup for the AMG solver on a cluster of 2 nodes with 4 CPUs + 4 GPUs vs. 4 CPUs alone, and with a 3x speedup for the total solution time.

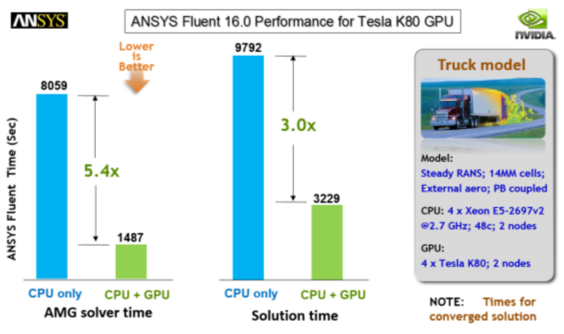

Figure 4: Speedups for ANSYS Fluent 16.0 on 2 nodes with a total of 4 x NVIDIA Tesla K80 GPUs.

The second study of 111M cells investigates performance of ANSYS Fluent on a much larger and scalable GPU cluster. The motivation for ANSYS Fluent deployment on clusters of GPUs is identical to that for CPU clusters – larger problems can be solved with overall performance increases. By distributing a model over a cluster with multiple GPUs in each node, single GPU memory size limitations can be overcome such that inexpensive GPUs become practical for solving large CFD simulations. There are several OEM manufacturers who can accommodate as many as 8 GPUs in a single node, enabling large-scale compute power in even small to medium size clusters. However, the resulting heterogeneous architecture poses an extended memory hierarchy that creates challenges in developing scalable and efficient application software.

Multiple programming APIs along with a domain decomposition strategy for geometry and data-parallelism is required to achieve high throughput and scalable results from a CFD application on a multi-GPU platform. When more than one GPU is used, cells at the edges of each GPU's computational domain must be communicated to the GPUs that share the domain boundary so that each GPU can have the current data necessary for their computations. Data transfers across neighboring GPUs adds latency into an implementation, which is a source that limits scalability if not properly managed.

Parallel simulations in ANSYS Fluent usually begin with a geometry domain decomposition step. This partitions the model geometry and distributes the partitions among the number cores on each available CPU – one partition per CPU core. ANSYS uses the Message Passing Interface (MPI) API for parallel programming on clusters and to manage communication between partitions – one MPI rank per partition per CPU core.

Typically, having one MPI process managing one GPU was the most straightforward approach for a distributed memory application on a multi-GPU cluster. In the case of ANSYS Fluent and most CFD, this would impose a configuration of one GPU per one CPU core, which is impractical owing to limits on the number of GPUs available for a single node, in addition to the overkill of available GPU memory capacity. For this reason, a 'consolidation' capability was developed for AmgX that efficiently manages multiple CPU-core MPI processes per GPU and is used in ANSYS Fluent. The number of CPU-core MPI processes that can be mapped to a single GPU is arbitrary, which allows flexibility in the configuration of multi-GPU clusters.

The second study of 111M cells was conducted with ANSYS Fluent 15.0 on a moderate-sized cluster system consisting of 12 nodes, each configured with:

- 2 x Intel Xeon E5-2667 @2.9GHz with 6 cores each, for a total of 12 total cores (144 cores total system)

- 4 x NVIDIA Tesla K40 GPUs (48 GPUs total system)

The 111M cell results shown in Figure 5 give performance comparisons of the CPU time both stand-alone and with GPU acceleration. The metric used in the performance is number of seconds per iteration for both the AMG solver only, and total solution time. The timings show a 2.7x speedup for the AMG solver on a cluster of 12 nodes with 144 CPUs + 48 GPUs vs. 144 CPUs alone, and with a 2x speedup for the total solution. These 12 node results were possible with the same solver settings as the 2 node case which simplifies use from a practitioner's perspective.

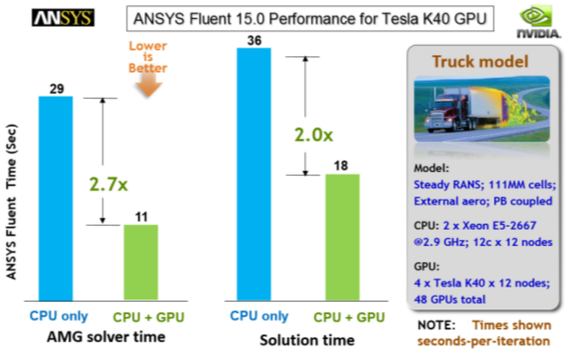

Figure 5: Speedups for ANSYS Fluent 15.0 on 12 nodes with a total of 48 x NVIDIA Tesla K40 GPUs.

Increased levels of parallel processing by utilizing GPUs for industry-relevant CFD applications in an HPC environment can enable much larger and complex simulations to be addressed with shorter turnaround times. As CFD simulation requirements continue to grow, such as the need for transients, high-resolution, and multi-scale simulations, heterogeneous parallel application software and systems of CPUs and GPUs will become an essential HPC technology.

For industry-leading commercial CFD software ANSYS Fluent, it was demonstrated that substantial performance gains can be achieved by using the latest NVIDIA GPU technology based on the Kepler architecture, in a GPU acceleration deployment of x86-based CPUs. Results from GPU acceleration achieved substantial factors of speedup, and results from a multi-GPU cluster demonstrate the potential for GPU parallel scalability. Based on these trends, we can expect that GPU acceleration will be a meaningful HPC trend in the future of commercial CFD software and in engineering modeling and simulation.

CONCLUSION

Computational challenges associated with high fidelity and advanced CFD simulations that impact requirements of HPC systems can now deploy hybrid parallel models of distributed and shared memory utilizing GPU acceleration for overall performance gains. This development also means deployment of heterogeneous clusters of CPU-GPU configured server nodes. Industry-leading commercial and non-commercial CFD software is undergoing development on such platforms that will provide a practical HPC environment for growing requirements of advanced CFD including, high-resolution and multidiscipline simulations.

As CFD simulation requirements grow and motivate the need for more transients, multi-scale simulations, and improved turbulence treatment, hybrid parallel application software and systems of CPUs + GPUs have become a relevant HPC technology. For ANSYS Fluent, it has been demonstrated that gains are achieved with each new GPU release. In addition, alternatives to x86 will be available for GPUs during 2015 including the Power and ARM64 CPU architectures, and with a new interconnect between CPU and GPU that is 4x faster than the current PCI-express interface. Based on current trends, GPU-based parallel CFD will be grow as a meaningful HPC benefit in the practice of engineering simulation for industry.

REFERENCES

CORRIGAN, A., CAMELLI, F., LOHNER R., and MUT, F, 2010. Porting of an edge-based CFD solver to GPUs. 48th AIAA Aerospace Sciences Meeting, number AIAA-2010-522. Orlando, FL, January 2010.

COHEN, J. and CASTONGUAY, P., Efficient Graph Matching and Coloring on the GPU, GPU Technology Conference, GTC on-demand S0332 (2012).

NVIDIA Corporation, AmgX REFERENCE MANUAL, V2.0, 2014.

KRAUS, J. and FORSTER, M., Efficient AMG on Heterogeneous Systems, LNCS, 7174 (2012), pp.
133- 146.

DEMOUTH, J., Optimization of Sparse Matrix-Matrix Multiplication on GPU, GPU Technology Conference, GTC-on-demand S0285 (2012).

KARYPIS, G. and KUMAR, V., A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM J. Sci. Comput., 20 (1999), pp. 359-392.

NAUMOV, M., Preconditioned Block-Iterative Methods on GPUs, Proc. Appl. Math Mech., 12
(2012), pp 11-14.

ANSYS, Fluent, http://www.ansys.com