2010 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
MODELING & SIMULATION, TESTING AND VALIDATION (MSTV) MINI-SYMPOSIUM
AUGUST 17-19 DEARBORN, MICHIGAN

NOVEL HPC CONSIDERATIONS FOR ADVANCED CFD

Stan Posey
NVIDIA Corporation
Santa Clara, CA

ABSTRACT

This paper examines the current state of scalable CFD for high-performance computing (HPC) clusters at industry-scale, and provides a review of novel technologies that can enable additional levels of CFD parallelism beyond today's conventional approach. Recent trends in HPC offer opportunities for CFD solution performance increases from the use of parallel file systems for parallel I/O, and a second level of solver parallelism through hybrid CPU-GPU co-processing.

INTRODUCTION

Parallel efficiency and overall simulation turn-around times continue to be important factors behind scientific and engineering decisions to develop CFD models at higher fidelity. While the return-on-investment for several years of CFD verses physical experiments has been remarkable, the maturity of parallel CFD solvers and availability of inexpensive scalable HPC clusters has not been enough to advance most CFD practice beyond steady state modeling. Fluids are inherently time dependent, yet to model such complexity at a practical HPC scale for industry, requires parallel efficiency for all levels of the CFD solution.

A capability of rapid simulation turn-around and multi-job throughput has potential to transform current practices in engineering analysis and design optimization procedures. This presentation will examine the current state of scalable CFD for HPC clusters at industry-scale, and provide a review of novel HPC technologies that can enable additional levels of CFD solution parallelism beyond a conventional approach.

Recent trends in HPC offer opportunities for substantial CFD solution performance increases from (i) the use of parallel file systems for parallel I/O, and (ii) a second level of solver parallelism through hybrid CPU-GPU co-processing. GPUs or graphics processing units, are now developed to share computational tasks with the CPU, in particular tasks that benefit from massively-parallel numerical operations. As more numerical operations are processed by GPUs, which lower the time spent in solvers, data I/O becomes a larger percentage of the overall solution processing time. This requires the need for parallel data I/O in order to benefit from an overall CFD time-to-solution.

HPC REQUIREMENTS FOR CFD

Parallel processing in HPC environments has enabled increasingly complex CFD problems to be addressed. From a processor viewpoint, CFD requires a balance of memory bandwidth and floating point performance, but benefits most from parallel scalability. More and more affordable parallel computing has enabled: higher resolution for multi-scale phenomena of a flow field induced by complex physical and geometrical features; transient simulation, unsteady effects; advanced turbulence modeling, such as the combination of RANS and LES; multiphase and non-equilibrium chemical reactions; multidiscipline analysis and design optimization accounting for the interaction of fluid flow with other disciplines including structures, thermal, and controls.

It is noteworthy that while HPC has indeed facilitated advanced CFD simulations and the growth of CFD, direct numerical simulations (DNS) of the Navier-Stokes equations for industry standard vehicle configurations with typical operating conditions are still beyond today's computing capability and therefore, still intractable. However, more practical intermediate approaches like RANS (Reynolds Averaged Navier Stokes), unsteady RANS, LES (Large-eddy Simulation) and other sophisticated turbulence modeling have received much attention in recent years.

Despite recent gains in such models, most industry CFD applications remain at a limiting level of steady state RANS owing to the relative cost-performance benefits of advanced CFD. A schematic of CFD complexity hierarchy is provided in figure 1 with requirements of computational capability. As industry addresses much larger and more complex CFD applications it naturally leads to a requirement of even higher levels of parallelism that in turn brings the important elements that make up HPC systems to immediate attention.

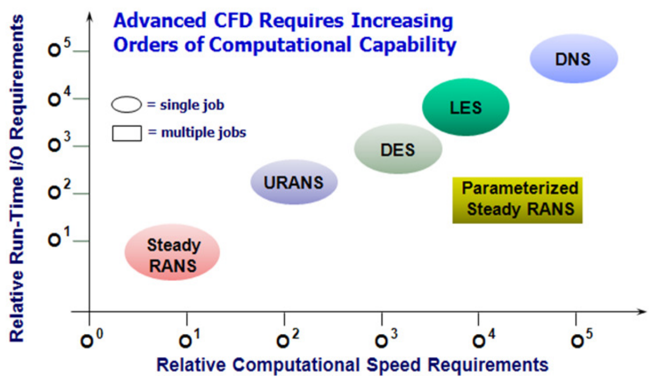
Advanced CFD Requires Increasing Orders of Computational Capability
= single job
= multiple jobs
Relative Run-Time I/O Requirements
Relative Computational Speed Requirements
Steady RANS
URANS
DES
LES
DNS
Parameterized Steady RANS
10^0
10^1
10^2
10^3
10^4
10^5

Figure 1: Schematic of relative HPC requirements for CFD

Commodity clusters are limited in their ability to provide scalable systems with very low latency and high bandwidth between potentially thousands of processing cores, as well as a limited in the support of tools for code profiling and optimization, parallel debugging, and cluster management. In addition the maintenance costs associated with such commodity clusters grows exponentially owing to:

• the physical space required to house and operate a large scale cluster of commodity servers is prohibitively high

• the power consumption and cooling challenges that data centers face housing a large scale cluster of commodity servers involving many power panels full of circuits, many, many power cables behind and below the racks potentially impacting airflow, performance, are indeed significant.

Some of the computational challenges associated with high fidelity advanced CFD simulations that impact requirements of HPC systems include parallel I/O for collective operations related to transient CFD, and hybrid parallel models of distributed and shared memory utilizing GPU acceleration of CPUs for improved scalability.

PARALLEL FILE SYSTEMS

As the HPC community continues an aggressive platform migration from proprietary supercomputers and Unix servers to Linux-based clusters, expectations grow for clusters to meet the I/O demands of increasing fidelity in CFD modeling. Cluster deployment has increased as organizations seek ways to cost-effectively grow compute resources for CAE applications. During this migration, many of these same organizations also implemented network attached storage (NAS) architectures to simplify administration and further reduce costs.

While NAS implementations offer several advantages of shared file systems, many are too limited in the scalability required to effectively manage the I/O demands of parallel CAE applications. As such, a new storage migration is underway to replace legacy serial NAS with parallel NAS architectures and parallel file systems. For example, the use of legacy file systems such as NFS on serial NAS for CFD I/O requirements can actually increase overall job time as more compute cores are added, rather than provide the desired effect of faster job turn-around through parallelism.

A new class of parallel file system and shared storage technology has developed that scales I/O in order to extend overall scalability of CFD simulations on clusters. For most implementations, entirely new storage architectures were introduced that combine key advantages of legacy shared storage systems, yet eliminate the drawbacks that have made them unsuitable for large distributed cluster deployments.

Parallel NAS can achieve both the high-performance benefits of direct access to disk, as well as the data-sharing benefits of files and metadata that HPC clusters require for CFD scalability. One implementation from Panasas [2] provides an object-based storage architecture that can eliminate serial I/O bottlenecks. Object-based storage enables two primary technological breakthroughs vs. conventional block-based storage.

First, since an object contains a combination of user data and metadata attributes, the object architecture is able to offload I/O directly to the storage device instead of going through a central file server to deliver parallel I/O capability. That is, just as a cluster spreads the work evenly across compute nodes, the object-based storage architecture allows data to be spread across objects for parallel access directly from disk. Secondly, since each object has metadata attributes in addition to user-data, the object can be managed intelligently within large shared volumes under a single namespace. This eliminates the need for administrators to focus on LUN management as those operations are automatically handled by intelligent storage blades.

Object-based storage architectures provide virtually unlimited growth in capacity and bandwidth, making them well-suited for handling CFD run-time I/O operations and large files for post-processing and data management. With object-based storage, the cluster has parallel and direct access to all data spread across the shared storage system. This means a large volume of data can be accessed in one simple step by the cluster for computation and visualization to improve speed in the movement of data between storage and other tasks in the CFD workflow.

Panasas provides this architecture by offering tuned hardware components that optimize the parallel file system software architecture capabilities.

Transient CFD with Parallel I/O

The benefits of parallel I/O for transient CFD were investigated with a production case of an ANSYS FLUENT Release 12 [3] aerodynamics model of 111M cells, provided by an industrial truck vehicle manufacturer. Figure 2 illustrates the I/O schematic of the performance tests that were conducted, which comprised a case file read, a compute solve of 5 time steps with 100 iterations, and a write of the data file. In a full transient simulation the solve and write tasks would be repeated to a much larger number of time steps and iterations, and with roughly the same amount of computational work for each of these repeatable tasks.

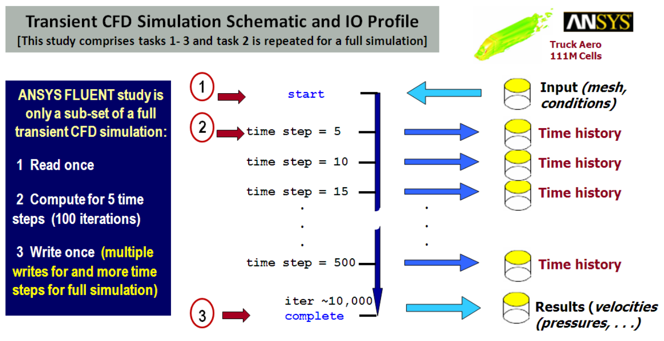
Transient CFD Simulation Schematic and IO Profile
[This study comprises tasks 1- 3 and task 2 is repeated for a full simulation]
ANSYS
Truck Aero
111M Cells
ANSYS FLUENT study is only a sub-set of a full transient CFD simulation:
1 Read once
2 Compute for 5 time steps (100 iterations)
3 Write once (multiple writes for and more time steps for full simulation)
1
start
2
time step = 5
time step = 10
time step = 15
time step = 500
3
iter ~10,000
complete
Input (mesh, conditions)
Time history
Time history
Time history
Time history
Results (velocities (pressures, . . .)

Figure 2: Schematic of truck aerodynamic transient CFD simulation and I/O scheme

Details of the CFD model, the cluster and storage configuration and the results of solver plus write times are provided in figure 3. (Note: the case-read are one-time tasks in full simulations and were therefore omitted in the results).

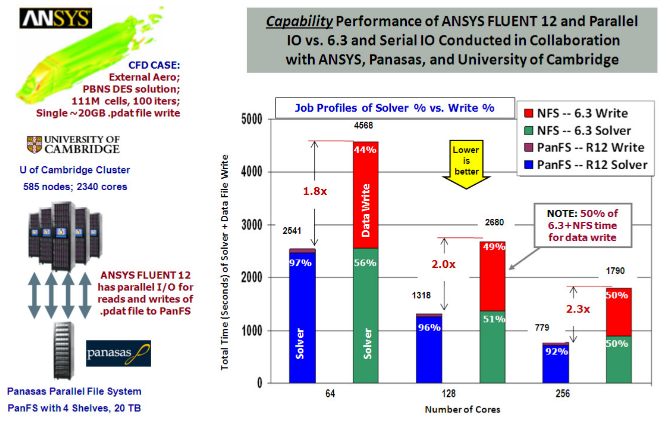

Figure 3: Comparison of ANSYS FLUENT R12 using a PanFS parallel storage file system vs. v6.3 using NFS

In this case study, ANSYS FLUENT R12 with parallel I/O conducted concurrent writes of the local (partition) solutions directly to the global solution data file on PanFS, unlike the case for v6.3 that writes the local solutions one-by-one to a global solution data file. The results demonstrate that R12 on the Panasas parallel file system demonstrated a more than 2 times reduction in total time vs. v6.3 on NFS and the NAS system. The total time improvement due to the write time advantage for R12 over v6.3, demonstrated up to 39 times higher data transfer rates measured in MB/s.

In the case of 64 cores, the solver advantage of R12 over v6.3 was only 4% with the total time benefit of 1.8 fold shown in Figure 3 attributed to the parallel I/O speed-up. The R12 solver advantage grows to 9% at 128 cores, and 24% at 256 cores, which contribute to the growing benefits in total time improvements of 2.0 fold on 128 and 2.3 fold on 256 cores for R12 on PanFS vs. v6.3 on NFS.

It is important to note that the performance of CFD solvers and numerical operations are not affected by the choice of file system, which only improves I/O operations. That is, a CFD solver will perform the same on a given cluster regardless of whether a parallel or serial NFS file system is used. The advantage of parallel I/O is best illustrated in a comparison of the computational profiles of each scheme. R12 on PanFS keeps the I/O percent of the total job time in the range of 3% at 64 cores to 8% at 256 cores, whereas v6.3 and NFS spend as much as 50% of the total job time in I/O.

An ANSYS FLUENT license is too valuable to spend a high percentage of operations in I/O relative to numerical operations. The high efficiency of a parallel I/O solution equates to ~2 times more ANSYS FLUENT 12 utilization that can be achieved within the same license cost structure as the v6.3 serial I/O solution under similar conditions.

GPU-PARALLEL CFD

The continual increase in CPU speeds has limits due to power and thermal constraints with processors now having multiple cores. To achieve boosts in performance without increasing clock speeds parallelism must be developed. This parallelism can come in the form of task parallelism, data parallelism, or perhaps a combination of the two. Common methods for implementing parallelism are explicit message-passing using an MPI library for either distributed or shared memory systems, and OpenMP for shared memory systems. A hybrid method is also possible with OpenMP used on multi-core and multiprocessor nodes and MPI used among the nodes.

Although parallel applications that use multiple cores are a well established technology in computational science and engineering (CSE), a recent trend towards the use of Graphics Processing Units (GPUs) to accelerate CPU computations is emerging. In this heterogeneous computing model the GPU serves as a co-processor to the CPU. The need for high performance and the parallel nature of CSE problems has led GPU designers to create current designs with hundreds of cores. Today GPUs and software development tools are available for implementing more general applications that use the GPU not for graphics but for applications such as CFD and others where computations are needed to be completed as fast as possible.

Much work has recently been focused on GPUs as devices that can be used in general-purpose computing. A GPU can produce a very high FLOPS (floating-point operations per second) rate if an algorithm is well-suited for the device. There have been several studies illustrating the acceleration of scientific computing codes that is possible by using GPUs [4 - 6]. Despite the tremendous performance gains possible with GPUs, relatively few commercial CFD software has yet to make use of them, and those that have demonstrate nominal overall gains of 2x over a multi-core CPU. This is mostly due to current GPU focus on iterative linear equation solvers rather than complete CFD code implementations.

GPU Considerations for CFD

The GPU was originally designed for graphics and the majority of this computation involves computing the individual color for each pixel on a screen. If we think of each pixel as being like a quadrilateral CFD cell, computing the pixel colors will be similar to the computations on a structured mesh. There is a very regular, orderly data access pattern with neighbors easily computed by simple offsets of indices. However, the majority of commercial CFD use some form of an unstructured mesh often with triangles in 2D or tetrahedral in 3D. These meshes lead to an irregular and somewhat disorderly data access pattern which is not particularly well suited to the memory system of a GPU.

Through a careful process of analyzing the relation between cells or elements and vertices, and taking advantage of the large amount of available processor performance on a GPU, techniques can be applied that partitions and sorts the mesh in such a way that the data access pattern becomes much more regular and orderly. The preprocessing of the mesh connectivity is a one-time step performed just before the main computation begins and can require a negligible amount of compute time while significantly increasing the performance of the equation solver.

Shared memory is an important feature of the GPU and is used to avoid redundant global memory access among threads within a block. The GPU does not automatically make use of shared memory, and it is up to the software to explicitly specify how shared memory should be used. Thus, information must be made available to specify which global memory access can be shared by multiple threads within a block. For structured grid based solvers, this information is known up-front due to the fixed memory access pattern of such solvers, whereas the memory access pattern of unstructured grid based solvers is data-dependent.

Algorithm design for optimizing memory access is further complicated by the number of different memory spaces the developer must take into consideration. Unlike a CPU the memory accesses are under the full and manual control of the developer. There are several memory spaces on the GPU which in turn is connected to the CPU memory. Different memory spaces have different scope and access characteristics: some are read-only, some are optimized for particular access patterns. Significant gains (or losses) in performance are possible depending on the choice of memory usage.

Another issue to be considered for GPU implementation is that of data transfers across the PCI-Express bus which bridges the CPU and GPU memory spaces. The PCI-Express bus has a theoretical maximum bandwidth of 4 or 8 GB/s depending on whether it is of generation 1 or 2. When this number is compared to the bandwidth between the GPU's on-board GDDR3 memory and the GPU multi-processors (up to 141.7 GB/s), it becomes clear that an algorithm that requires a large amount of continuous data transfer between the CPU and GPU will unlikely achieve good performance.

For a CFD solver, the obvious solution is to limit the size of the domain that can be calculated so that all of the necessary data can be stored in the GPU's main memory. Using this approach, it is only necessary to perform large transfers across the PCI-Express bus at the start of the computation (geometry) and at the end (final flow solution). High-end NVIDIA GPUs will offer up to 6 GB of main memory by the end of 2010, sufficient to store all the data needed by most commercial parallel CFD software, so this restriction is not a significant limitation.

Over the past 10 years, CFD has become increasingly reliant on clusters of multiprocessors to enable more detailed simulations within design time frames. For this reason, the scalability of a solver across multiple processors can be equally important as its single-processor performance. A potential problem with increasing the single-processor performance by an order of magnitude is then that the multi-processor performance suffers since the time required to exchange boundary information remains roughly constant. When operating in parallel across multiple GPUs, some boundary information must be transferred across the PCI-Express bus at the end of each time step. However, with implementation of a low surface-to-volume ratio in mesh partitioning, this data transfer need not be a bottle-neck.

A well suited memory access pattern and GPU-parallel considerations are not sufficient to achieve compelling performance gains. Many commercial CFD codes use an iterative method for the equation solver that typically includes linearization because they are relatively fast and easy to develop and maintain. Most often these are conjugate-gradient based methods with pre-conditioning schemes, and operate on sparse matrices. While these methods are simpler for development they are not as well suited to the GPU due to the fact that they access memory often and conduct a rather low amount of actual computation with each data access. This metric is typically referred to as arithmetic intensity and GPUs are particularly well suited to algorithms with high arithmetic intensity.

The relative performance of processors vs. memory over the past few decades has extended to more than 3 orders of magnitude. CPUs have gone to great lengths to bridge the gap between processor and memory performance by introducing instruction and data caches, instruction level parallelism, and so forth. 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. If we extrapolate out, without any fundamental changes in memory, processors will become infinitely fast relative to memory, and performance optimization will become solely an exercise in optimizing data movement.

Example: AcuSolve Commercial CFD

An example is provided with AcuSolve, a commercial CFD software based on the finite element method and developed by ACUSIM [9], with headquarters in Mountain View, CA. AcuSolve is based on the Galerkin/Least-Squares (GLS) finite element method. GLS is a higher-order accurate, yet stable formulation that uses equal order nodal interpolation for all variables, including pressure. The method is specifically designed to maintain local and global conservation of relevant quantities under all operating conditions and for all unstructured mesh types. AcuSolve utilizes an efficient iterative solver for fully coupled pressure/velocity equation systems which like most commercial CFD solvers has sparse matrix-vector multiply as its primary operations.

Sparse matrix-vector multiplication (SpMV) is of singular importance in sparse linear algebra. In contrast to the uniform regularity of dense linear algebra, sparse operations encounter a broad spectrum of matrices ranging from the regular to the highly irregular. Exploiting the tremendous potential of throughput-oriented GPU processors for sparse operations requires that a solver expose substantial fine-grained parallelism and impose sufficient regularity on execution paths and memory access patterns. Optimizing SpMV for GPUs is qualitatively different than SpMV on latency-oriented multi-cores. Whereas a multi-core SpMV kernel needs to develop 4 or more threads of execution, a many-core implementation must distribute work among thousands or tens of thousands of threads. Many-core GPUs will often demand a high degree of fine-grained parallelism because, instead of using large sophisticated caches to avoid memory latency, they use hardware multithreading to hide the latency of memory accesses.

The linear equation solver in AcuSolve was examined for GPU acceleration for a relatively simple case of 80,000 elements for an S-Duct geometry. AcuSolve utilizes a hybrid parallel MPI/OpenMP approach with an iterative GMRES solver. An execution profile of AcuSolve for the S-Duct case appears in figure 4 and demonstrates the dominant feature of SpMV operations as 57% of the total execution time. As described in a previous section, even a complete port of SpMV to the GPU will still leave 43% of execution time on the CPU and limit the overall effective speed-up to ~2x.

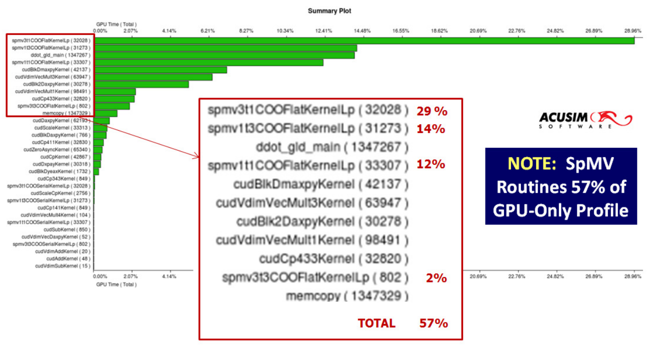

Figure 4: Execution profile of AcuSolve and S-Duct case

Results for the S-Duct case are shown in figure 5 and feature a comparison between a quad-core Xeon 5500 series CPU (Nehalem) and a single NVIDIA Tesla C2050 (Fermi). The speed-up of the 1 core + GPU is ~2x and this improves to 3.3x with the use of 2 GPUs owing to the distributed memory parallel implementation in AcuSolve. ACUSIM has additional GPU development projects underway to improve the performance of future releases.

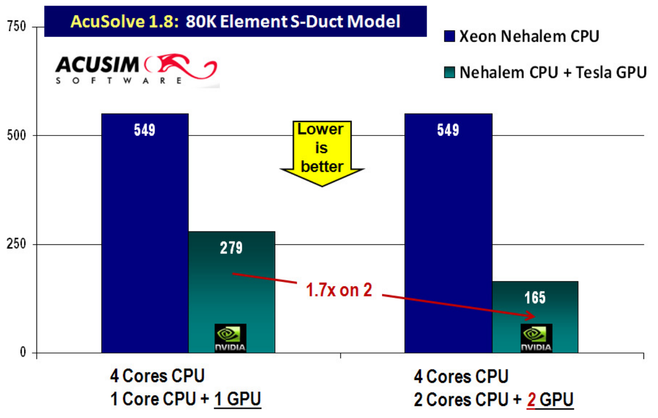

Figure 5: AcuSolve 1.8 results for multi-GPU NVIDIA Tesla C2050 results vs. 4 Core Xeon Nehalem CPU

SUMMARY

Increased levels of CFD parallel processing by utilizing parallel file systems and GPUs in an HPC environment has enabled much larger and complex CFD simulations to be addressed in product development workflows. As CFD simulation requirements continue to grow such as the need for transients, high-resolution LES, and multidiscipline simulation that are heavy in I/O operations relative to numerical operations, parallel file systems and parallel NAS will be essential technologies. The heterogeneous nature of such high fidelity simulations and their HPC resource usage will continue to grow the requirements for balanced and GPU co-processing HPC environments involving large GPU-based servers within distributed memory clusters. It was demonstrated that substantial performance gains can be achieved by using the latest novel HPC technologies. Based on these trends, we can expect that HPC will be a powerful tool in the future of scientific computing and advanced CFD modeling and practice.

REFERENCES

[1] Kodiyalam, S., Kremenetsky M., Posey S., "Balanced HPC Infrastructure for CFD and Associated Multidiscipline Simulations of Engineering Systems," Proceedings, 7th Asia CFD Conference 2007, Bangalore, India, November 26 – 30, 2007.

[2] Gibson, G.A., Van Meter, R., "Network Attached Storage Architecture," Communications of the ACM, Vol. 43, No. 11, November 2000.

[3] ANSYS FLUENT: www.fluent.com/software/fluent

[4] Andrew C., Fernando C., Rainald L., John W., 19th AIAA Computational Fluid Dynamics, June 22-25,San Antonio, Texas.

[5] Brandvik, T., Pullan, G., "An Accelerated 3D Navier-Stokes Solver for Flows in Turbomachines," Proceedings of GT2009 ASME Turbo Expo 2009: Power for Land, Sea and Air June 8-12, 2009, Orlando, USA.

[6] Michalakes, J. and Vachharajani, M., "GPU Acceleration of Numerical Weather Prediction," Parallel Processing Letters, 18(4):531-548. 2008.

[7] Palix Technologies, LLC. http://www.palixtech.com. Advanced Numerical Design Solver ANDSolver White Paper.

[8] NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture 2.0 Programming Guide, 2008.

[9] ACUSIM Corporation and AcuSolve: www.acusim.com