

# Implementing a GPU-Enhanced Cluster for Large-Scale Simulations

**Robert F. Lucas, Gene Wagenbreth & Dan M. Davis**  
Information Sciences Institute, Univ. of So. Calif.  
Marina del Rey, California  
[rflucas](mailto:rflucas@isi.edu), [genew](mailto:genew@isi.edu) & [ddavis](mailto:ddavis@isi.edu) @isi.edu

## ABSTRACT

The simulation community has often been hampered by constraints in computing: not enough resolution, not enough entities, not enough behavioral variants. Higher performance computers can ameliorate those constraints. The use of Linux Clusters is one path to higher performance; the use of Graphics Processing Units (GPU) as accelerators is another. Merging the two paths holds even more promise. The authors were the principal architects of a successful proposal to the High Performance Computing Modernization Program (HPCMP) for a new 512 CPU (1024 core), GPU-enhanced Linux Cluster for the Joint Forces Command's Joint Experimentation Directorate (J9). In this paper, the basic theories underlying the use of GPUs as accelerators for intelligent agent, entity-level simulations are laid out, the previous research is surveyed and the ongoing efforts are outlined. The simulation needs of J9, the direction from HPCMP and the careful analysis of the intersection of these are explicitly discussed. The configuration of the cluster and the assumptions that led to the conclusion that GPUs might increase performance by a factor of two are carefully documented. The processes that led to that configuration, as delivered to JFCOM, will be specified and alternatives that were considered will be analyzed. Planning and implementation strategies are reviewed and justified. The presentation will then report in detail about the execution of the actual installation and implementation of the JSAF simulation on the cluster in August 2007. Issues, problems and solutions will all be reported objectively, as guides to the simulation community and as confirmation or rejection of early assumptions. Lessons learned and recommendations will be set out. Original performance projections will be compared to actual benchmarking results using LINPACK and simulation performance. Early observed operational capabilities of interest are proffered in detail herein.

## ABOUT THE AUTHORS

**Robert F. Lucas** is the Director of the Computational Sciences Division of the University of Southern California's Information Sciences Institute (ISI). There he manages research in computer architecture, VLSI, compilers and other software tools. He has been the principal investigator on the JESPP project since its inception in 2002. Prior to joining ISI, he was the Head of the High Performance Computing Research Department for the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory, the Deputy Director of DARPA's Information Technology Office, and a member of the research staff of the Institute for Defense Analysis's Center for Computing Sciences. From 1979 to 1984 he was a member of the Technical Staff of the Hughes Aircraft Company. Dr. Lucas received his BS, MS, and PhD degrees in Electrical Engineering from Stanford University in 1980, 1983, and 1988 respectively.

**Gene Wagenbreth** is a Systems Analyst for Parallel Processing at the Information Sciences Institute at the University of Southern California, doing research in the Computational Sciences Division. Prior positions have included Vice President and Chief Architect of Applied Parallel Research and Lead Programmer of Pacific Sierra Research, where he specialized in tools for distributed and shared memory parallelization of Fortran programs. He has also been active in benchmarking, optimization and porting of software for private industry and government labs. He has programmed on CRAY, SGI, Hitachi, Fujitsu, NEC, networked PCs, networked workstations, IBM SP2, as well as conventional machines. He received a BS in Math/Computer Science from the University of Illinois in 1971

**Dan M. Davis** is the Director, JESPP Project, Information Sciences Institute (ISI), University of Southern California, and has been active in large-scale distributed simulations for the DoD. While he was the Assistant Director of the Center for Advanced Computing Research at Caltech, he managed Synthetic Forces Express, a major simulation project. He was a lead in the proposal to take over the Maui High Performance Computing Center, where he subsequently served as the Director of Finance and Contracts. Prior to that, he was a Software Engineer on the All Source Analysis System project at the Jet Propulsion Laboratory and worked on a classified project at Martin Marietta, Denver. An active duty Marine Cryptologist, he recently retired as a Commander, USNR, Cryptologic Specialty. He has served as the Chairman of the Coalition of Academic Supercomputing Centers and the Coalition for Academic Scientific Computation. He received a B.A. and a J.D., both from the University of Colorado in Boulder.

# Implementing a GPU-Enhanced Cluster for Large-Scale Simulations

Robert F. Lucas, Gene Wagenbreth & Dan M. Davis  
Information Sciences Institute, Univ. of So. Calif.  
Marina del Rey, California  
[rflucas](mailto:rflucas@isi.edu), [genew](mailto:genew@isi.edu) & [ddavis](mailto:ddavis@isi.edu) @isi.edu

## INTRODUCTION

This paper addresses the background for, approach to and the experience of the authors with the new GPU accelerator-enhanced Linux Cluster at JFCOM. Requirements, design considerations, configuration decisions, and early experimental results are reported.

### Joint Forces Command Mission and Requirements

Live, virtual and constructive simulations play a vital role in DoD analysis, evaluation and training. The Joint Forces Command (JFCOM) has the mission to lead the transformation of the U.S. Armed Forces and to enable broad-spectrum dominance as per Joint Vision 2010 (CJCS, 1996) and 2020 (CJCS, 2000). JFCOM's research arm is the Joint Experimentation Directorate, J9. This leads to the nearly unique situation of having a research activity lodged within an operation command, calling for experiments in which warfighters in uniform are staffing the consoles during interactive, HPC-supported simulations.

The complexities of urban warfare are modeled by J9 in a series of experiments using well-validated entity-level simulations, *e.g.* Joint Semi-Automated Forces (JSAF) and the Simulation of the Location and Attack of Mobile Enemy Missiles (SLAMEM). These need to be run at a scale and resolution adequate for modeling the complexities of urban combat.

The J9 code came from a long lineage of entity-level battlefield codes. Terrain representations are populated with intelligent-agent friendly forces, enemy personnel and civilian groups. These have compute requirements in order to generate their behaviors. In addition, a major computational load is imposed in the performance of line-of-sight calculations for the entities and route-finding algorithms for the movers. This is a problem of some moment, especially in the light of its inherently onerous "n-squared" growth characteristics of such code (Brunett, 1998).

Consider a case of several thousand entities needing to interact with each other in urban settings with vegetation and buildings obscuring the lines of sight. This situation has been successfully met by the use of

innovative interest-managed communications (Barrett, 2004).

JFCOM requires an enhanced Linux cluster of adequate size, power, and configuration to support simulations of more than 2,000,000 entities operating within high-resolution insets on a global-scale terrain database. This facility will be used occasionally to interact with live exercises, but more often will be engaged interactively with users and experimenters while presenting virtual or constructive simulations. (Ceranowicz, 2005) It must be robust, to reliably support hundreds of personnel, and it must be scalable, to easily handle both small activities and large, global-scale experiments with the participants distributed trans-continentially, as shown in below.

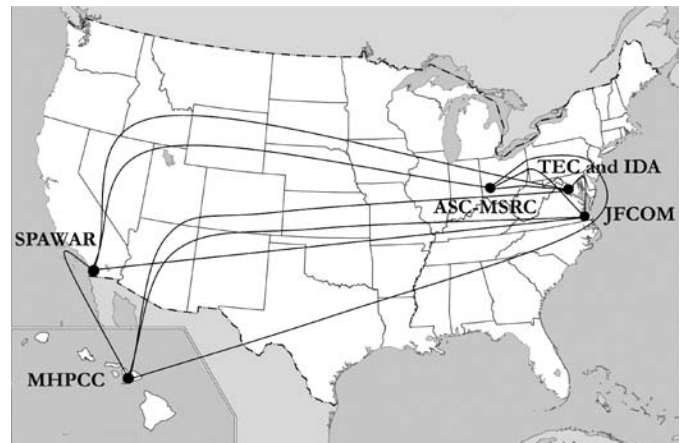


Figure 1 - JFCOM's HPC Simulation Net

### Joint Futures Lab (JFL)

The creation of a standing experimentation environment that can respond immediately to DoD time-critical needs for analysis is the goal of the JFL. It operates in a distributed fashion over the Defense Research and Engineering Network (DREN), at a scale and level of resolution that allows JFCOM and its partners to conduct experimentation on issues of concern to combat commanders, who often participate in the experiments themselves.

The Joint Futures Lab consists of extensive simulation federations, software, and networks, joined into one common infrastructure that supports experiments. This capability includes quantitative and qua-

litative analysis, flexible plug-and-play standards, and the opportunity for diverse organizations to participate in experiments.

### **Joint Advanced Training and Tactics Laboratory (JATTL)**

Supporting mission rehearsal, training, operational testing, and analysis is the JATTL's *raison d'etre*. The principal thrusts of the JATTL are developing technologies that support the pre-computed products required for joint training and mission rehearsal. This is being explored under the Joint Rapid Distributed Database Development Capability and support programs. The latter include phenomenology such as environment, cultural assets, civilian populations, and other effects necessary to represent real operations. The JATTL is connected nationally via both DREN and the National Lambda Rail (NLR) to over thirty Joint National Training Capability sites.

### **JFCOM's JESPP**

A scalable simulation code that has been shown capable of modeling more than 1,000,000 entities has been designed and developed by the J9 team. This effort is known as the Joint Experimentation on Scalable Parallel Processors (JESPP) project (Lucas, 2003.) This work builds on an earlier DARPA/HPCMP project named SF Express. (Messina, 1997) The early JESPP experiments on the University of Southern California Linux cluster showed that the code was scalable, well beyond the 1,000,000 entities actually simulated, given the availability of additional nodes (Wagenbreth, 2005).

The current code has been successfully fielded and reliably operated using JFCOM's HPCMP-provided compute assets hosted at ASC-MSRC, Wright Patterson AFB, and at the Maui High Performance Computing Center (MHPCC) in Hawai'i. The J9 team has been able to make the system suitable and robust for day-to-day use, both unclassified and classified.

This HPC platform is needed in order to deliver a state-of-the-art capability to military experimenters so they can use it to easily initiate, control, modify, and comprehend any size of a battlefield experiment. It now additionally allows for the easy identification, collection, and analysis of the voluminous data from these experiments, all of which have been enabled by the work of Dr. Ke-Thia Yao's team (Yao, 2005).

A typical experiment would find the JFCOM personnel in Suffolk Virginia interfacing with a "Red Team" in Fort Belvoir Virginia, a civilian control group at SPAWAR San Diego California, and partic-

ipants at Fort Knox Kentucky and Fort Leavenworth Kansas, all supported by the clusters on Maui and in Ohio. The use of interest-managed routers on the network has been successful in reducing inter-site traffic to low levels.

Even using these powerful computers, the JFCOM experimenters were constrained in a number of dimensions, *e.g.* number of entities, sophistication of behaviors, realism of various environmental phenomenology, *etc.* While the scalability of the code would have made the use of larger clusters feasible, a more effective, efficient, economical and elegant solution was sought.

### **Broader Impacts for the HPCMP Community**

The discipline of Forces Modeling and Simulation (FMS) is unique to the DoD, compared to many of the other standard science disciplines, *e.g.* CFD (Computational Fluid Dynamics) and Weather. In a similar way, interactive computing is a new frontier being explored by the JESPP team for FMS, coordinating with a few other user groups. Along these lines, the newly enhanced Linux Cluster capability will provide significant synergistic possibilities with other computational areas such as visualization, advanced numerical analysis techniques, weather modeling and other disciplines or computational sciences such as SIP, CFD and CSM (Signals/Image Processing, Computational Fluid Dynamics, and Computational Structural Mechanics).

The specific DoD goal is to enhance global-scale, computer-generated support for experimentation by sustaining more than 2,000,000 entities on appropriate terrain, along with valid phenomenology. To accomplish this, the authors proposed a configuration of a 512 CPU (1024 core), GPU-enhanced Linux Cluster to be located at the JFCOM site in Suffolk Virginia, with one NVIDIA 7950 GPU on each of the dual CPU (Quad-core) nodes. GPUs should especially be of consequence in such algorithms as those for the line-of-sight and route-planning calculations, mentioned above. Early experiments have already suggested that they are amenable to exploitation on GPUs (Salmon *et al.* 2004) While the optimal mix of GPUs to CPUs is as yet unknown, the authors thought that space, heat dissipation, and other engineering constraints mitigated in favor of one GPU per node.

The quest to explore broader use of GPUs is often called GPGPU, which stands for General Purpose computation on GPUs (Lastra 2004). While the programming of GPUs has been pursued for some time, the newly released Compute Unified Device Archi-

texture (CUDA) programming language (Buck, 2007) has made that effort more accessible to journeyman C programmers. For that reason, the HPCMP accepted the desirability of upgrading the original cluster configuration, which called for NVIDIA 7950s, to NVIDIA 8800, specifically to enable the use of CUDA. This met with HPCMP's long-standing goal of providing operationally sound platforms rather than experimental configurations that could not be utilized easily by the wider DoD HPC community.

## RESEARCH APPROACH

The full conversion of the JSAF code to make use of the GPU is considered infeasible. Instead, only computational bottlenecks such as LOS can plausibly be considered for the GPUs. To gain familiarity with GPU programming the authors opted to implement a code segment from another simulation community. The code chosen was extracted from one of the well known "crash codes." A brief description of the computational kernel and the methods employed will assist the readers in analyzing applicability to their own codes.

Sparse matrix factorization is a well-known impediment to fast computation in applications such as Mechanical Computer-Aided Engineering (MCAE), making it an excellent target for GPU acceleration. Factoring large sparse linear systems can be done via many algorithms. Transforming the sparse matrix factorization into a hierarchy of dense matrix factorizations, the multifrontal method (Duff 83), is particularly attractive.

Multifrontal codes can effectively exploit the memory hierarchies of cache-based microprocessors, routinely going out-of-core to disk as needed. With the right data structures, the vast majority of the floating point operations can be performed with calls to highly tuned Basic Linear Algebra Subprograms (BLAS3) routines, such as the SGEMM (Single-precision GEMM) multiplication routine (Dongarra 1990), and near peak throughput could be expected. All of the major commercial MCAE applications use multifrontal solvers.

Very high levels of performance can be achieved on GPUs, as has been shown in recent GPGPU work on dense, single-precision linear algebra computations, *e.g.*, SGEMM, (Larson 2001, Fatahalian 2004, Govindaraju 2007). This then leads to the query as to whether such performance can be achieved in a multifrontal sparse solver. If so, then GPUs can be readily and cost-effectively used to accelerate MCAE codes. The following sections report on an experi-

ment designed to test this hypothesis and its relationship to an similar uses in FMS.

## Overview of a Multifrontal Sparse Solver

The non-zero structure of a small sparse matrix is depicted in Figure 2. An 'x' represents coefficients that are initially non-zero, while an '\*' represents those that fill-in during factorization. Choosing an optimal order in which to eliminate these equations is in general an NP-complete problem, so heuristics, such as METIS (Karypis & Kumar 1995), are used to try to reduce the storage and operations necessary. The multifrontal method treats the factorization of the sparse matrix as a hierarchy of dense sub-problems.

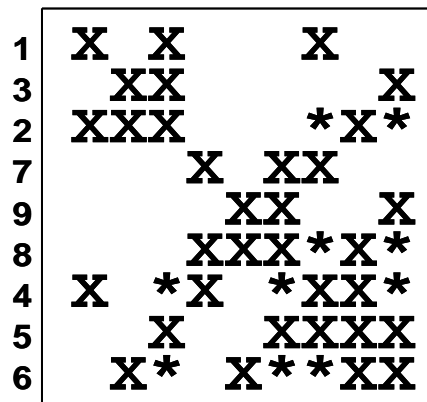
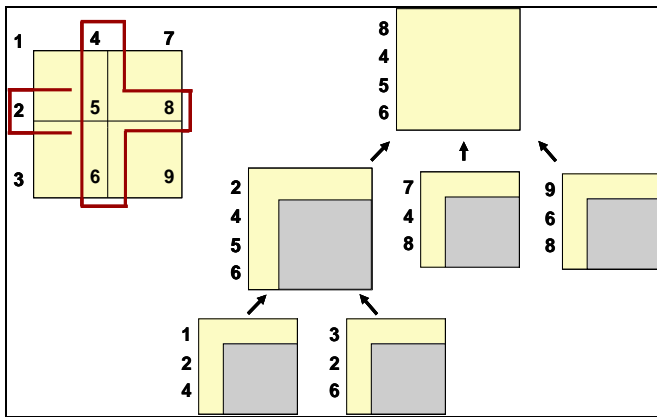


Figure 2 -Sparse matrix with symmetric non-zero structure

Figure 3 below depicts the multifrontal view of the matrix in Figure 2, above. The directed acyclic graph of the order in which the equations are eliminated is called the elimination tree. When each equation is eliminated (*i.e.*, used as the pivot), a small dense matrix called the frontal matrix is assembled. The numbers to the left of each frontal matrix are its row indices in Figure 2. Frontal matrix assembly proceeds as follows: the frontal matrix is cleared, it is loaded with the initial values from the pivot column (and row if it's asymmetric), then any updates generated when factoring the pivot equation's children (in the elimination tree) are accumulated.

Once the frontal matrix has been assembled, the variable is eliminated. Its Schur complement (the shaded area in Figure 3) is computed as the outer product of the pivot row and pivot column from the frontal matrix. Finally, the pivot equation's factor (a column of L) is stored and its Schur complement placed where it can be retrieved when needed for the assembly of its parent's frontal matrix. If a post-order traversal of the elimination tree is used, the Schur complement matrix can be placed on a stack of real values.



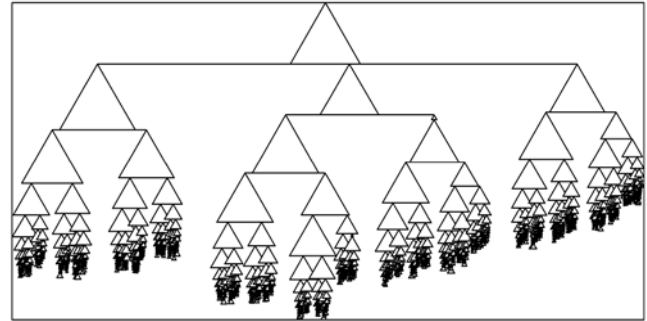
**Figure 3 – Multi-frontal view of sparse matrix from figure 1**

The cost of assembling frontal matrices is reduced by exploiting super-nodes. A super-node is a group of equations whose non-zero structures in the factored matrix are indistinguishable. For example, zeros filled-in during the factorization of the matrix in Figure 2 turn its last four equations into a super-node.

The cost of assembling one frontal matrix for the entire super-node is amortized over the factorization of all the constituent equations, reducing the multi-frontal matrices overhead. Furthermore, when multiple equations are eliminated from within the same frontal matrix, their Schur complement can be computed very efficiently as the product of two dense matrices.

Figure 4 illustrates the elimination tree for a matrix, as ordered by METIS. This particular elimination tree has 12,268 supernodes in it. There are thousands of leaves and one root. The leaves are relative small,  $O(10)$  equations being eliminated from  $O(100)$ . The supernodes near the root are much bigger. Hundreds of equations are eliminated from over a thousand.

Because dense factor operations scale as order  $N^3$ , approximately two dozen supernodes at the top of the tree contain half of the total factor operations. The objective of the work reported in the remainder of this paper was to attempt to use GPUs as inexpensive accelerators to factor the large supernodes near the root of the elimination tree. This should in turn lead to a significant and cost-effective increase in the throughput in MCAE as well as familiarize the authors with programming GPUs.



**Figure 4 – Supernodal elimination tree (Courtesy Cleve Ashcraft)**

## GRAPHICS PROCESSING UNITS

The NVIDIA GeForce 8800 GPU architecture consists of a set of multiprocessors. Each multiprocessor has a set of Single Instruction Multiple Data (SIMD) architecture processors. NVIDIA provided ISI with an early model GTS card with eight multiprocessors. The authors used the GTS card for code development and benchmarking. The newer GTX card has 16 multiprocessors.

Each multiprocessor of both models has 8 SIMD processors. The GPU supports single precision (32 bit) IEEE 754 (Arnold, 1992) formatted floating-point operations. Each SIMD processor can perform a multiply and an add instruction at every clock cycle. The clock rate on the GTS card the authors used is 675 MHz. Therefore, the peak performance is:

$$675 \text{ mhz} * 2 \text{ results/op} * 2 \text{ op/clock} \\ * 8 \text{ SIMD/mp} * 8 \text{ mp} = 172.8 \text{ GFLOP/s}$$

The GTX card, with a slightly higher clock rate and twice as many multiprocessors, has a peak performance of over 350 GFLOP/s.

Memory on the GTS GPU is organized into device memory, shared memory and local memory. Device memory is large (768 MBytes), is shared by all multiprocessors, is accessible from both host and GPU, and has high latency (over 100 clocks). Each GTS multiprocessor has a small (16 Kbytes) shared memory that is accessible by all the SIMD processors on the multiprocessor.

Shared memory is divided into banks and, if accessed so as to avoid bank conflicts, has a one-clock latency. Shared memory can be thought of a user-managed cache or buffer between device memory and the SIMD processors. Local memory is allocated for each thread. It is small and can be used for loop va-

riables and temporary scalars, such as registers would be used. There is also a constant memory and a texture memory that were not used in this effort.

In our experience, there are two primary issues that must be addressed to use the GPU efficiently: First the code must use many threads, without conditionals, operating on separate data to keep the SIMD processors busy. Second code must divide data into small sets, which can be cached in shared memory.

Once in shared memory, data must be used in many (10 – 100) operations to mask the time spent transferring between shared and device memory. It is not feasible to convert a large code such as JSAF or OneSAF to execute on the GPU. Instead, compute-bound subsets of the code must be identified that use a large percentage of the execution time. Only those subsets should be converted to run on the GPU. Their input data is transferred from the host to the GPU's device memory before initiating computation on the GPU. After the GPU computation is complete, the output data is transferred back to the host from GPU device memory.

To facilitate general-purpose computations on their GPUs, NVIDIA announced a new Compute Unified Device Architecture (CUDA) programming language (Buck, 2007). CUDA is a minimal extension of the C language and is loosely type-checked by the NVIDIA compiler (and preprocessor), `nvcc`, which translates CUDA programs (.cu) into C programs.

These are then compiled with the `gcc` compiler and linked as an NVIDIA provided library. Within a CUDA program, all functions have qualifiers to assist the compiler with identifying whether the function belongs on the host or the GPU. For variables, the types have qualifiers to indicate where the variable lives, *e.g.*, `__device__` or `__shared__`. CUDA does not support recursion, static variables, functions with arbitrary numbers of arguments, or aggregate data types.

CUDA supports the option of linking with an emulation library to test GPU code while executing only on the host. When emulated on the host, GPU code can have `PRINTF`s for debugging. This was found to be very convenient. There is also an option to create a log file with timing and other statistics for each GPU kernel execution. Using a simple PERL script for aggregation of timings, this appeared very useful and was used extensively for tuning and optimization.

The authors compared timings of the CUDA matrix multiply routine with the highly optimized version supplied in NVIDIA's CUBLAS library of basic nu-

merical linear algebra functions. The CUDA version was within a factor of two of the library version. Some of this difference is probably due to use of a more efficient (and complex) algorithm in the library version. This demonstrated that it is possible to write reasonably efficient code using CUDA.

### GPU Frontal Matrix Factorization Performance

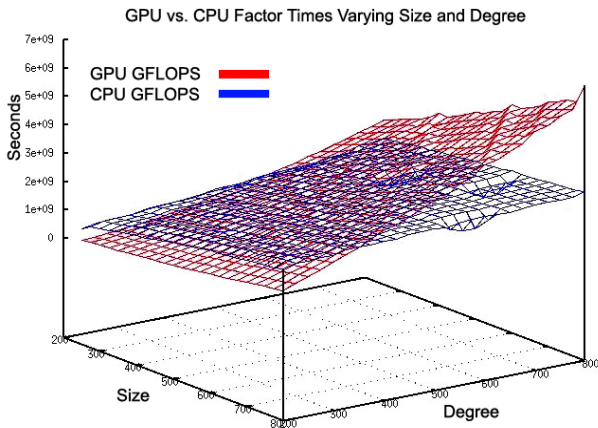
Performance results using the GPU to factor a variety of model frontal matrices are presented below in TABLE 1.

**Table 1 GPU frontal matrix factorization kernel Performance**

Size	De- gree	Secs	FLOPS
512	1024	0.204E+00	0.417E+10
1024	1024	0.256E+00	0.980E+10
1536	1024	0.334E+00	0.157E+11
2048	1024	0.437E+00	0.213E+11
512	2048	0.272E+00	0.101E+11
1024	2048	0.367E+00	0.185E+11
1536	2048	0.490E+00	0.255E+11
2048	2048	0.653E+00	0.307E+11
512	3072	0.386E+00	0.147E+11
1024	3072	0.535E+00	0.248E+11
1536	3072	0.752E+00	0.305E+11
2048	3072	0.934E+00	0.376E+11
512	4096	0.553E+00	0.176E+11
1024	4096	0.753E+00	0.290E+11
1536	4096	0.101E+01	0.364E+11
2048	4096	0.144E+01	0.378E+11

These range in the number of equations eliminated from the frontal matrix (size) as well as the number of equations left in the frontal matrix, *i.e.*, its external degree (degree). As expected, the larger the frontal matrix gets, the more operations one has to perform to factor it, and the higher the GPU performance.

The multifrontal code factors frontal matrices of various sizes, ranging from very small to very large. For small matrices the host is faster than the GPU. Tests were run to determine when the relative performance of the host and the GPU for a range of frontal matrices. Figure 5 below is a plot of the performance for various sizes and degrees, comparing the host and the GPU.



**Figure 5 - Comparison of the frontal matrix factorization performance of the GPU and its host**

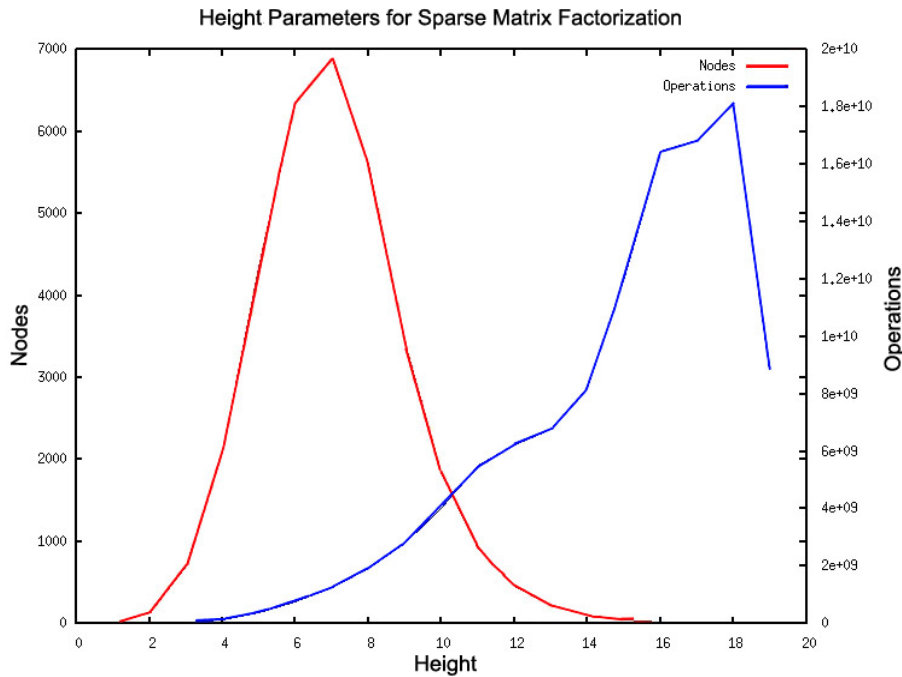
Ultimately the criteria that were chosen for deciding to use the GPU to factor a frontal matrix were if its size was greater than 127 or its leading dimension (size plus degree) was greater than 1023. Performance for the GPU and host are very close near this boundary. Small adjustments of the criteria or attempts to tune it by adding complexity had little effect on performance.

**Accelerated Multifrontal Solver Performance**

The performance impact of the GPU on overall multifrontal sparse matrix factorization is examined here. Three matrices were extracted from LSTC’s LS-DYNA (Livermore Software DYNAMIC finite element code), one of the premier MCAE applications extant. They were: hood, a two-dimensional problem; ibeam, a three-dimensional structure built with two-dimensional shells; and knee, a three-dimensional solid, extracted from a model of a prosthetic knee.

To better understand the impact of the GPU on the overall multifrontal factorization, a closer look at the ibeam problem is advisable. The x-axis of FIGURE 6 represents different levels in the elimination tree of the ibeam matrix. The root is to the right at level 19 and the leaves to the left. The red curve is the sum of the number of frontal matrices at each level. It increases exponentially until it peaks near 7000 at level 7. A few leaves of the tree appear even deeper. The blue curve plots the sum of the floating point operations needed to factor the frontal matrices at each level of the tree. The integral of this curve is approximately 101 billion, the total number of operations needed to factor the *ibeam* problem.

It is clear from the figure that the vast majority of the operations are in the top five levels. In fact 60 frontal matrices in the top six levels of the tree exceed the threshold for use of the GPU. Together, they comprise 65% of the total factor operations.

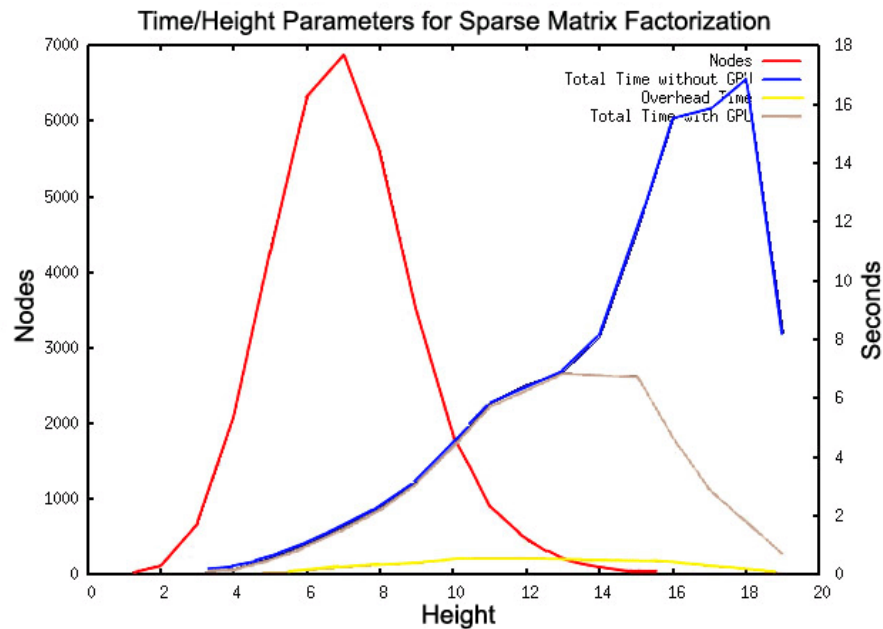


**Figure 6 - Number of super-nodes and factor work at each level of the ibeam elimination tree**

FIGURE 7 below depicts the sum of the time spent at each level of the *ibeam* elimination tree. The red curve represents the sum of the supernodes at each level. The yellow curve is the time spent assembling frontal matrices and stacking their Schur complements.

These are the overheads associated with using the multifrontal method. The blue curve is the total time

spent at each level of the tree when running on the host. The difference between the blue and yellow curves is the time spent factoring the frontal matrices. The brown curve is the time spent at each level of the elimination tree when using the GPU. The difference between the brown curve and the yellow one is the time spent on the GPU.



**Figure 7 - Number of Supernodes and time spent factoring each level of the *ibeam* elimination tree**

It is clear from looking at **FIGURE 7** that the GPU is very effective at reducing the time spent factoring the large frontal matrices near the root of the elimination tree. Factorization using the CPU alone took 109.08 seconds; with the GPU: 56.14 seconds. The difference between the brown and blue curves is the 52.94 seconds by which the GPU accelerated the overall factorization.

## CONCLUSIONS

This research will provide warfighters with the new capability to use Linux clusters in a way that will simulate the required throngs of entities and suitably global terrain. These are necessary to represent the complex urban battlefield of the 21st Century. It will enable experimenters to simulate the full range of forces and civilians, all interacting in future urban conflict zones. The use of GPUs as acceleration devices in distributed cluster environments shows apparent promise in any number of fields. Further exper-

imentation should extend the applicability of these concepts. The CUDA code proved to be easily exploited by experienced C programmers.

The work reported herein has demonstrated that a GPU can in fact be used to significantly accelerate the throughput of a multi-frontal sparse symmetric factorization code. The authors have demonstrated speed-ups as high as 1.97 for factorization, and 1.86 overall when accounting for preprocessing of the matrix and the triangular solves. This was done by designing and implementing a symmetric factorization algorithm for the GeForce 8800 in NVIDIA's new CUDA language and then offloading a small number of large frontal matrices, containing over half the total factor operations, to the GPU.

Having now familiarized themselves with the architecture and programming environment of the NVIDIA G8800 GPU, the authors believe they are now prepared to leverage GPUs to accelerate the performance of JSAF and other JFCOM simulation programs. Towards this end, they have designed a 512-node (1024 core) Linux cluster with 256 NVIDIA

G8800 GPUs. HPCMP has ordered such a system and it is scheduled for installation at JFCOM in the summer of 2007. The GPU-enhanced cluster will be used to support training and experimentation by J7 and J9.

## ACKNOWLEDGEMENTS

The authors are grateful for the unstinting support of the NVIDIA staff in this early foray into CUDA and GPU use, most especially Dr. Ian Buck and Norbert Juffa. Some of this material is based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-05-2-0204. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

## REFERENCES

- Ashcraft, C. and R. Grimes, (1989) The Influence of Relaxed Supernode Partitions on the Multifrontal Method, *ACM Transactions in Mathematical Software*, 15 1989, pp. 291-309
- Barrett, B. & Gottschalk, T.D., (2004), Advanced Message Routing for Scalable Distributed Simulations, *2004 IITSEC Conference*, Orlando, FL
- Brunett, S., & Gottschalk, T.D., (1998), A Large-scale Meta-computing Framework for the Mod-SAF Real-time Simulation, *Parallel Computing*, V24:1873-1900, Amsterdam
- Buck, I., (2007), GPU Computing: Programming a Massively Parallel Processor, *International Symposium on Code Generation and Optimization*, San José, California
- Buttari, A., J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, (2007) Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy, submitted to *ACM Transactions on Mathematical Software*, 2007.
- Ceranowicz, A. & Torpey, M., (2005), Adapting to Urban Warfare, *Journal of Defense Modeling and Simulation*, 2:1, January 2005, San Diego, Ca
- Charlesworth, A., & Gustafson, J., (1986), Introducing Replicated VLSI to Supercomputing: the FPS-164/MAX Scientific Computer, in *IEEE Computer*, 19:3, pp 10-23, March 1986
- CJCS, (2000), *Joint Vision 2020*, Director for Strategic Plans and Policy, J5: Strategy Division, Washington, D.C.: Government Printing Office
- CJWC, (1997), *Concept for Future Joint Operations*, Commander, Joint Warfighting Center, Fort Monroe, VA.
- Dongarra, J. J., J. Du Croz, S. Hammarling, and I. S. Duff (1990), A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Transactions on Mathematical Software* 16(1):1-17, March 1990
- Dongarra, J., (1993), Linear algebra libraries for high-performance computers: a personal perspective, *Parallel & Distributed Technology: Systems & Applications, IEEE*, Feb. 1993, Volume: 1, Issue: 1, pp: 17 - 24
- Duff, I and J Reid, (1983) The Multifrontal Solution of Indefinite Sparse Symmetric Linear Systems, *ACM Transactions on Mathematical Software*, 9 1983, pp 302-335
- Duff, Ian. (1986) Parallel Implementation of Multifrontal Schemes, *Parallel Computing*, 3 1986, pp 193-204.
- Fatahalian, K., J. Sugarman, and P. Hanrahan, (2004) Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication, In Proceedings of the *ACM Siggraph/Eurographics Conference on Graphics hardware*, pages 133-138, Eurographics Association, 2004
- Fatahalian, K., Sugarman, J. & Hanrahan, P., (2004), Understanding the efficiency of GPU algorithms for matrix-matrix multiplication, *Workshop on Graphics Hardware, Eurographics/SIGGRAPH*
- Govindaraju, N. and D. Manocha, (2007) Cache-Efficient Numerical Algorithms Using Graphics Hardware, *University of North Carolina Technical Report*, 2007.
- Gustafson, J.L., (2006.) The Quest for Linear Equation Solvers and the Invention of Electronic Digital Computing, *2006 International Symposium on Modern Computing*, Sofia, Bulgaria
- Joint Pub 1-02, (2000), *Department of Defense Dictionary of Military and Associated Terms*, Chairman of the Joint Chiefs of Staff, Washington, D.C.
- Karypis G. and V. Kumar, (1995) A fast and high quality multilevel scheme for partitioning irregular graphs, *International Conference on Parallel Processing*, pp. 113-122, 1995
- Larson, E. S. and D. McAllister, (2001) Fast matrix multiplies using graphics hardware, In Proceedings of the *2001 ACM/IEEE conference on Supercomputing*, pages 55-55, ACM Press, 2001
- Lastra, A., Lastra, M. Lin, and D. Minocha, (2004), *ACM Workshop on General Purpose Computations on Graphics Processors*.

- Lucas, R., & Davis, D., (2003), Joint Experimentation on Scalable Parallel Processors, *2003 IITSEC Conference*, Orlando, FL
- Lucas, R.F., Wagenbreth, G., Tran, J.J., & Davis, D. M., (2007), *Multifrontal Computations on GPUs*, unpublished ISI White Paper, on line at: [www.isi.edu/~ddavis/JESPP/2007\\_Papers/SC07/mf2\\_gpu\\_v0.19a-nms.doc](http://www.isi.edu/~ddavis/JESPP/2007_Papers/SC07/mf2_gpu_v0.19a-nms.doc)
- Messina, P. C., Brunett, S., Davis, D. M., Gottschalk, T. D., (1997) Distributed Interactive Simulation for Synthetic Forces, In Mapping and Scheduling Systems, *International Parallel Processing Symposium*, Geneva
- Pham, D. C., T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, Stephen Weitzel, Dieter Wendel, and K. Yazawa, (2006) Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor, *IEEE Journal of Solid State Circuits*, Vol. 41, No. 1, January 2006
- Salomon, B., Govindaraju, N. K., Sud, A., Gayle, R., Lin, M. C., & Manocha, D., "Accelerating Line of Sight Computation Using Graphics Processing Units", *Proc. of Army Science Conference*, 2004
- Wagenbreth, G., Yao, K-T., Davis, D., Lucas, R., and Gottschalk, T., (2005), Enabling 1,000,000-Entity Simulations on Distributed Linux Clusters, *WSC05-The Winter Simulation Conference*, Orlando, Florida,
- Whaley, R.C. & Dongarra, J.J., (1998), Automatically Tuned Linear Algebra Software, *IEEE/ACM Conference on Supercomputing, SC98.*, pp.:38 - 38
- Wilkinson, J. H., (1965) *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, UK, 1965
- Yao, K-T., Ward, C. & Wagenbreth, G., (2006), Agile Data Logging and Analysis, *2003 IITSEC Conference*, Orlando, FL (Arnold 1992) Arnold, M.G., T.A. Bailey, J.R. Cowles & M.D. Winkel, Applying Features of IEEE 754 to Sign/Logarithm Arithmetic, *IEEE Transactions on Computers*, August 1992, Vol. 41, No. 8, pp. 1040-1050
- Yozo H, X. S. Li and D. H. Bailey, (2001) Algorithms for Quad-Double Precision Floating Point Arithmetic, *15th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, 2001, pg. 155-162