A High Performance Route-Planning Technique for Dense Urban Simulations

John J. Tran, Ke-Thia Yao, Gene Wagenbreth, Dan M. Davis and Robert F. Lucas
Information Sciences Institute/USC
Marina del Rey, California
{jtran, kyao, genew, ddavis and rflucas}@isi.edu

David J. Brakeman
Nakuru Software Inc
Seattle, Washington
nakuru@comcast.net

ABSTRACT

To exploit the explicit and implicit advantages of data parallelism and heavily threaded modern multi-core processors, specifically the NVIDIA family of general purpose graphic processing units (GPGPU), research efforts such as "Accelerating Line of Sight Computation Using GPUs" (Manocha 2005) and "Implementing a GPU-Enhanced Cluster for Large-Scale Simulations" (Lucas 2007) addressed various problems found in military simulations, yet other practical uses for the GPU in these types of simulation applications remain to be explored. An example application that has immediate use for a fast and large-scale graph-based construct is a route-planning algorithm found in complex urban conflict simulation, *e.g.* the Joint Semi-Automated Forces (JSAF) simulation. JSAF currently employs a heuristic A* search algorithm to do route planning for its millions of entities –- the algorithm is sequential and thus very computationally expensive. Using the GPU, the JSAF simulation can off-load the route-planning component to the GPU and remove one of its major bottlenecks.

The objective of this research effort is to build a framework that utilizes all the features and raw computational power of the GPU architecture to solve the above challenge. Our research effort addresses the many challenges of parallel programming on the GPU: data locality, massive thread counts, and race conditions, to name a few. Our project will greatly benefit the modeling and simulation community facing issues specific to route planning and of particular interest are those simulations dealing with dense urban environments, homeland security, and mass casualty and disaster simulations. We achieve this goal by providing a practical and seemingly "endless" source of raw computing powers found in GPUs for massively large graph-based family of problems.

ABOUT THE AUTHORS

John J. Tran, a researcher at ISI/USC, is currently pursuing a doctorate in Computer Science from the Viterbi School of Engineering at the University of Southern California. He received both his BS and MS Degrees in Computer Science and Engineering from the University of Notre Dame, where he focused on object-oriented software engineering, large-scale software system design and implementation, and high performance parallel and scientific computing. He has worked at the Stanford Linear Accelerator Center, Safetopia, and Intel. His current research centers on Linux cluster engineering, effective control of parallel programs, and communications fabrics for large-scale computation.

Ke-Thia Yao is a research scientist in the Distributed Scalable Systems Division of the University of Southern California Information Sciences Institute. Currently, he is working on the JESPP project, which has the goal of supporting very large-scale distributed military simulation involving millions of entities. Within the JESPP project he is developing a suite of monitoring/logging/analysis tools to help users better understand the computational and behavioral properties of large-scale simulations. He received his B.S. degree in EECS from UC Berkeley, and his M.S. and Ph.D. degrees in Computer Science from Rutgers University.

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 long 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 SF Express, a major agent based simulation project. Prior to that, he was a Software Engineer on the ASAS project at JPL and worked at Martin Marietta, Denver. An active duty Marine Cryptologist, he retired as a Commander, USNR, Cryptologic Specialty. He received a B.A. and a J.D., both from the University of Colorado in Boulder.

Robert F. Lucas is the Director of the Computational Sciences Division and and Research Professor 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, a member of the research staff of the Institute for Defense Analysis's Center for Computing Sciences and the Technical Staff of the Hughes Aircraft Company. Dr. Lucas received his BS, MS, and PhD degrees in Electrical Engineering from Stanford University.

David J. Bakeman has over 20 years experience in the computer software industry. He has worked in the modeling and simulation industry for the last 14 years. Prior to that he spent seven years at Hewlett Packard developing real time embedded firmware for spectrum analyzers. He is known for his ability to work on his own initiative, develop schedules and meet those schedules. He works well with people and has lead teams on many occasions. David is a recognized expert and originator of the latest advances to the Compact Terrain Database (CTDB) format utilized by the Joint Semi Automated Forces (JSAF) simulation tool. David earned his BS in Computer Science from Montana State in Bozeman.

A High Performance Route-Planning Technique for Dense Urban Simulations

John J. Tran, Ke-Thia Yao, Gene Wagenbreth,
Dan M. Davis and Robert F. Lucas
Information Sciences Institute/USC
Marina del Rey, California
{jtran, kyao, genew, ddavis and rflucas}@isi.edu

David J. Bakeman
Nakuru Software Inc
Seattle, Washington
nakuru@comcast.net

INTRODUCTION

Problem Description

The need for higher fidelity simulations is on the rise. Modern conflict space has shifted from open space theatre of conventional warfare to unconventional, and very often asymmetric, warfare that is carried out in dense urban settings. The United States Joint Forces Command (USJFCOM) has conducted a series of complex experiments involving multi-force engagement in dense urban environments using the Joint Semi-Automated Synthetic Force (JSAF) simulators. Similarly, the Department of Homeland Securities shares similar interests in complex urban setting simulations, *e.g.* how to prepare for a mass disaster impacting densely populated US cities. Generally speaking, there are two possible ways to improve simulation fidelity: (a) by increasing entity counts (quantitatively) and (b) by increasing accuracy (qualitatively) of entity behaviors and resolution of the environment. Numerous efforts have been made to increase the former, *e.g.* SF Express (Brunnet, *et al.* 1998) and Noble Resolve (USJFCOM 2006). These included the use of the Scalable Parallel Processors (SPP) or clusters of compute nodes (Wagenbreth, *et al.* 2005). As for the latter, JFCOM M&S teams have made great strides to improve entity behavior models (Ceranowicz, *et al.* 2002 and 2006) by adding intelligence to the simulation entity behaviors, and with these improvements entities behave in more realistic fashions. Because JSAF has been required to participate in more urban operations, the density of the road and trail networks has dramatically increased. This dictates an increase in computational costs (in terms of how entities relate to the environment), which is the heart of this research effort.

Motivation

The use of specialized hardware to solve complex problems is not a new phenomenon. In fact, many scientific and research needs have driven computer technology innovations. Graphics Processing Units (GPU) were designed and optimized for visualization in support of the video game industry. They provide extremely fast floating-point operations[1]. As the gaming industry expands the drive to mass-produce these powerful graphics boards, the production costs are brought down to the point where it is economically viable (if not, sensible) to consider the GPU for general purpose computing engines. The High Performance Computing Modernization Program (HPCMP) Office recognizes the potentials behind the general-purpose graphics-processing unit (GPGPU) computing paradigm and has awarded a GPU-based cluster to USJFCOM/J9 (Figure 1).

Figure 1. Joshua cluster at JFCOM consisting of 256 dual quad core Opteron processors and 256 NVIDIA GTS8800 GPUs (Photo Source: Ed Aaron, USJFCOM).

In addressing the various computational challenges JSAF developers face, we observe and propose some potential areas of improvements to the JSAF simulation. One of these includes the use of GPU for the route-planning stage. The general argument goes: given that during the route-planning stage for each of the tens of thousands of simulation entities throttles the route computation component of the software and virtually brings the simulation federations to a grinding halt, the computationally expensive code can be offloaded to the GPU. This will address one of the goals in the overall attempt to increase simulation

[1] Currently most of the graphics cards natively support only 32 bit floating point precisions.

fidelity and will help to support or detract from the view that this approach is useful.

BACKGROUND

Graphics Processing Units (GPU)

The GPU or Graphics Processing Units are specialized hardware (traditionally in the form of a video card) and are now typically attached to the computer using a PCI-Express bus. GPUs are designed to have much higher arithmetic processing performance than their CPU host, which is necessary for the graphics' niche for which they are designed. They have more processors, each with more Arithmetic Logic Units (ALUs), explicit memory hierarchy, and all executing the same instruction sequence on parallel data. The design of the GPU technology focuses on three important aspects: (1) The GPU promotes explicit and implicit parallelism, (2) effectively implements voluminous (billions of) computing instructions, and (3) throughput performance that more than offsets latency penalties.

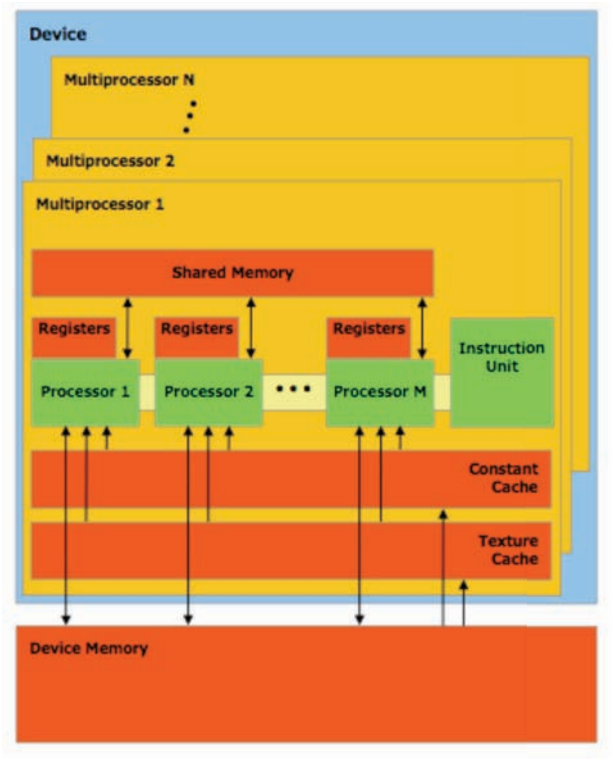


Figure 2. The SIMD Multiprocessor Model. (Source NVIDIA Corporation).

To support the above design goals, GPUs utilize highly threaded instruction blocks. The underlying motivation behind the GPU design is that there is lots of work to be done and it is far better to keep as many threads simultaneously busy as feasible. This philosophy relies on the notion of data parallelism, which is a sharp contrasts with a typical CPU design goal, that of task parallelism. The GPU achieves data parallelism with spatial division (as opposed to temporal division) (Owens, *et al*. 2008).

Compute Unified Device Architecture (CUDA)

The CUDA programming model is held to be a clean extension to the C programming language. CUDA gives programmers the ability to the exploit SPMD (single program multiple data) programming model on the GPU. CUDA programs are highly threaded. Access to shared memory space is achieved through gather and scatter operations. As per nVidia, here are two notes concerning CUDA programs: (1) There is no explicit synchronization mechanism with CUDA programs, and (2) the wholesale executions in parallel are the real gain of GPU-based applications.

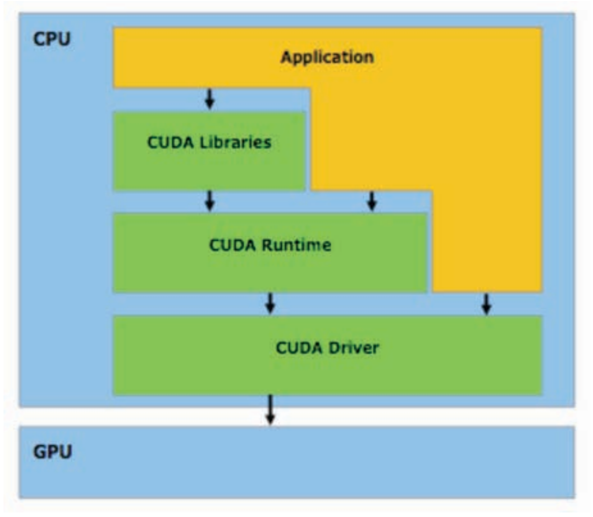


Figure 3. The CUDA Software Stack. (Source NVIDIA Corporation).

The CUDA compiler is a C pre-processor and specialized compiler that assists programmers with parallel programming. Programmers write programs as normal. Computationally intensive sections are explicitly tagged for execution on the GPU and are executed on segments of data on the GPU concurrently by many threads (NVIDIA 2008).

Graph Algorithm

Route-finding

Route finding is a class of algorithms that finds the "best" path, given a network of paths with *N* vertices (or nodes), between any number of vertices *N(i,j)*. The criteria for determining these paths (roads or edges) is determined by a cost function *F[N(i,j)]*. The overall goal is to determine the min (or max) of all *F*'s of all of edges along the path.

Table 1 below summarizes some of the route finding algorithms. Note that the Floyd-Warshall (FW) exists both in the serial (CPU-based implementation) and parallel (GPU-based implementation) form (Micikevicius 2005 in *GPU Gem 2*). The ASSP algorithm is the same as the SSSP algorithm, with the only difference being that it is implemented for all the of M paths in a particular network.

Table 1. Route-planning Algorithm Classifications

	Properties			
	A	**B**	**C**	**D**
A*	Serial	Priority Queue	N^2	N log N
MM	Serial & Parallel	Adjacency Matrix	N^2	N^3 log N
FW	Serial & Parallel	Adjacency Matrix	N^2	N^3
SSSP	Parallel	Adjacency List	N^2	N log N
ASSP	Parallel	Adjacency List	N^2 * M	N^2 log N

For the properties in Table 1, column A denotes the implementation model, B denotes the connectivity graph representation, C denotes the space (storage size), and D denotes the big O notation for time complexity.

JSAF SIMs & A Algorithm*

In its current implementation the JSAF simulator uses the serial A* search algorithm[2] to compute the "optimal" path for its clutter entities. The A* algorithm operates in *O(N log N)* time. We used the following as bases for our approach: (1) JSAF is a distributed federated simulation platform, and (2) JSAF, in its current state, is coarse-grain scalable. The distributed federation per processor model implies that for every compute node, the JSAF load-balancer assigns a group of entities to its compute "basket." Our design should restrict the computational space bound to these entities. Secondly, because JSAF is coarse-grain scalable, our design exploits the higher resolution simulation per node by speeding up the computational time for the same amount of work on each node.

The A* algorithm is a heuristic implementation of the A algorithm first introduced by Hart, Nilsson, and Raphael in 1968 (Hart *et al*. 1968) and it works as followed:

1. Given a starting point
2. Searches all routes leading to the destination point
3. Keep the minimum path based on a cost function

A* involves the use of heuristics to improve performance over the traditionally greedy BFS (breadth first search) algorithm because the algorithm maintains a set of nodes (or vertices) already visited in a priority queue (Dechter and Pearl 1985). The A* algorithm belongs to the class of single source single path family and it is by nature a serial algorithm since only one node or vertices is considered at a time.

EXPERIMENTATION

Hardware Platform

For our experiment we developed the graph algorithms on an NVIDIA experimental board (on loan for research from NVIDIA) and conducted the timing on the Joshua cluster at J9. Figure 4 details the cluster specifications :

- CPU - 256 X (2) AMD Santa Rosa 2220 2.8 GHz dual-core processors, for a total of 4 cores per node
- GPU - 256 x (1) NVIDIA 8800 GTS video cards with 512MB
- Memory - 256 x 16 GB DIMM DDR2 667 per node – 4 GB per processor core

Figure 4. Joshua Configuration

Please note that for the current implementation, we use a single node for the experiment. However, future work is planned that will expand to the entire cluster.

Graph Algorithms

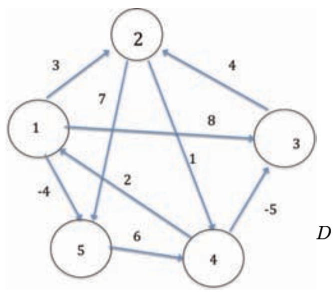
The treatment of the various graph algorithms and CUDA implementation is discussed in detail in the Harish & Narayaman paper (Harish & Narayanan 2007). For the purpose of this paper, we employ two efficient network path graph algorithms and explore two methods for graph representations: (1) Matrix multiplication variant using adjacency matrix, and (2)

[2] Described in great detail in: "A* Search Algorithm." http://en.wikipedia.org/wiki/A*_search_algorithm

Dijkstra's variant of the shortest path using adjacency list (implemented as an array). These two algorithms and graph representations, are in our view, improvements to the JSAF implementation of the A* algorithm. Table 1 provides a synopsis of the route-finding algorithm and its classifications.

Matrix-Multiplication

One of the approaches to implementing an all-to-all (all-pairs) shortest path algorithm is a technique similar to matrix multiplication. There are many advantages to this approach. These include: (1) a compact representation of connectivity graph using the adjacency matrix and (2) the efficient straightforward algorithm used for matrix multiplication on the GPU is easily modified to perform the all-to-all algorithm. Figure 5 illustrates a graph and its corresponding matrix representation. Our implementation of this algorithm is straightforward and is taken directly from the *Introduction to Algorithms* (Cormen 2001) text and done with minimal changes to the matrix multiplication routine found in CUDA example library.



$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

Figure 5. Connectivity Graph and its Adjacency Matrix Representation

Single Source Shortest Path (SSSP)

The SSSP algorithm we developed for our experiment is based on Dijkstra's shortest path algorithm. The CUDA implementation is similar to the Harish and Narayan implementation. We modified their implementation to efficiently address race conditions and synchronization. The algorithm iterates over node pairs keeping track of updated nodes for which better paths have been found. The algorithm terminates when no better paths have been found. A CUDA thread is used for each node. For each updated node, Harish and Narayan update all the node's neighbors. This can result in multiple threads simultaneously updating the same node. Our modification (listed below) is to have each thread update its own nodes if neighbors have been updated. A node is only updated by its own thread, perhaps multiple times per iteration. The race condition is avoided.

```
1. while(anything changes)
2. for every node
3.  for every neighbor of node
4.     if neighbor has been updated
5.          nodedist = \
 min(nodedist,neigbordist+dist(node,neighbor))
```

Application to JSAF Simulations

At time of writing, we are in the process of integrating the two GPU implementations of the shortest path code with the JSAF simulations. We are working with a number of constraints: (1) the system must be able to support 10 to 20 thousand vertices, (2) the time spent building and copying graphs between the CPU and GPU, (3) the current JSAF interactions with the route planning subsystem is stored and processed *vis-à-vis* the a MySQL DBMS.

Integration with MySQL DBMS

Recently in conjunction with Nobel Resolve 2008 the need arose for a routing algorithm to support traversal of a network maintained within a MySQL database. The A* algorithm utilized by the JSAF program was readily adaptable to the situation given the similarities in network representation.

The basic algorithm was implemented in three different ways. The first was an external MySQL client. This client queries the MySQL database at startup for the network maintaining its own local network representation. The second was as a MySQL UDF (User Defined Function). A UDF is implemented in C and C++ as a shared library which when installed on the MySQL server machine can be called from standard MySQL queries. This version requires the database to be loaded from the database into its own local representation. The final version was implemented as a MySQL stored procedure. This implements the priority queue as a MySQL temporary table. There is no need to maintain a separate network since this version can directly access the database when needed.

The first two versions both require a local copy of the network. However, once the network is loaded they can process potential routes very efficiently. A major drawback is that some external process must update their copies if anything changes. The final approach has the advantage of direct access to the network, but suffers from reduced performance. The performance might be offset by the fact that this method can be directly used in the MySQL cluster environment.

FINDINGS AND ANALYSIS

Our findings are summarized in Table 2. A number of conclusions can be drawn from the preliminary results.

For practical consideration, the matrix-multiplication approach, although yielding excellent results, is not practical for real application implementation for two reasons: (1) the large memory footprint, and (2) the cost of copying the connectivity graph between the GPU and CPU can be very expensive. Because of the first drawback, we do not see the MM algorithm to be practical for use in the JSAF environment, as the problem size can exceed 20k nodes.

Although we did not have a chance to implement the MM route finding algorithm on the GPU, the performance gain between the GPU implementation and the CPU implementation of a standard matrix-multiplication application is measured at 30 times (we tested this locally). We project this performance gain to be comparable with an implementation of the MM route finding algorithm.

We find the SSSP approach to be more appropriate (Figure 6). The measured gain between the GPU over the CPU is two times and this is based on an optimized implementation of the route finding SSSP algorithm. For ASSP, we would only need to run the algorithm *M* times (one for each of the nodes in consideration).

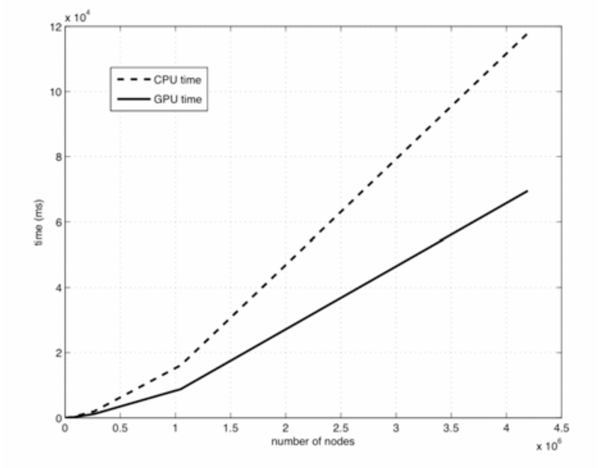


Figure 6. Timing Comparison between the SSSP route finding algorithm of a lattice grid on the GPU against the CPU.

Note that we did not integrate the GPU implementation into the actual JSAF system. As such, we cannot provide side-by-side comparison between the serial and parallel version for the JSAF route finding component.

Table 2. Comparison of performance findings for our early timing results

	Practical	**Performance**
All-to-All (MM)	GPU – Not practical N is capped at 20k	GPU – O(N^3)/C C = 128
	CPU – Not practical N is capped at 20k	CPU – (N^3)/C C = 4
One-to-All (SSSP)	GPU – Practical N is 1 Million	N log(N)/C C = 128
	CPU – Practical N is 1 Million	N log(N)/C C = 4

All-to-All (ASSP)	GPU = yes practical
	N = 1M * # GPU
		GPU - N^2 log(N)/C
		C = number core * 128

	CPU = yes not efficient however	CPU – N^2 log(N)/C C = number core

CONCLUSIONS AND FUTURE WORK

The findings observed are very encouraging in that we can see visible and tractable improvement in using GPU for computationally intensive routines. For the GPU case, we have 128 cores dedicated for parallel computations. This is in comparison to the CPU, where we have 4 cores. Furthermore, these cores also share processing time with other OS-related tasks.

We will focus our future work in three areas. First is to fully couple the GPU routines with the JSAF applications in a production environment. Our goal is to have the code automatically detects the presence of GPU(s) and use them as needed. Second, we see our implementation as extending to fit the cluster environment in which JSAF currently operates. The end result would be to have a cluster of GPU(s) available to support scalable coarse- and fine-grain application routines. Finally, we are interested in providing to programmers a generic black-box interface that would encapsulate the GPU as a sophisticated algorithmic co-processor. In doing so, we would be able to hide the many complicated implementation details associated with GPU programming, e.g. data-locality, memory coalescing, and synchronization.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the following for their support: NVIDIA Corporation, the United States Joint Forces Command, Andy Ceranowicz, Edwin Aaron, and our birddog Tom Stanzione. 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

Brunnet, S., Davis, D., Gottschalk, T., Messina, P., & Kesselman, C. (1998). “Implementing Distributed Synthetic Forces Simulations in Metacomputing Environments.” *Proceedings of the Seventh Heterogeneous Computing Workshop*, IEEE Computer Society.

Ceranowicz, Andy, Torpey, M., Helfinstine, B., Evans, J., & Hines, J. (2002) "Reflections on building the joint experimental federation," *Interservice/Industry Training, Simulation, and Education Conference Proceedings*.

Ceranowicz, Andy, Torpey, M., & Hines, J. (2006) “Sides, Force, and ROE for Asymmetric Environments,” *Interservice/Industry Training, Simulation, and Education Conference Proceedings*.

Cormen, Thomas M., Leiserson, Charles E., Rivest, Ronald L., and Stein, Cliff (2001). *Introduction to Algorithms, 2nd Ed.* MIT Press, 2001.

Dechter, Rina; Judea Pearl (1985). "Generalized best-first search strategies and the optimality of A*". Journal of the ACM 32 (3): pp. 505–536

Hart, P.E., Nilsson, N.J., & Raphael, B. (1968) “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” *IEEE Transaction on Systems Science and Cybernetics SSC4 (2).*

Lucas, Robert F., Wagenbreth, G., & Davis, Dan M. (2007) Implementing a GPU-Enhanced Cluster for Large-Scale Simulations. *Interservice/Industry Training, Simulation, and Education Conference Proceedings*.

Harish, Pawan Narayanan, P.J. (2007) “Accelerating large graph algorithms on the GPU using CUDA” *Proceedings of the IEEE International Conference on High Performance Computing* (HiPC 2007). Goa.

NVIDIA (2008). *NVIDIA CUDA Programming Guide*. http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf.

Manocha, Dinesh. (2005) “General-purpose computations Using Graphics Processor. “ *IEEE Computer Society*.

Owens, John D., Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. GPU Computing. *Proceedings of the IEEE*. 96(5), May 2008.

Pharr, Matt, Ed. (2008) *GPU Gems 2: Programming Techniques for High-Performance Graphics and*

General-Purpose Computation. Addison-Wesley Professional.

USJFCOM. *Noble Resolve*. Taken from http://www.jfcom.mil/about/experiments/nobleresolve.html

Wagenbreth, G., Yao, K-T, Davis, D., & Lucas, R. (2005) "Enabling 1,000,000-Entity Simulations on Distributed Clusters." *Proceedings of the 2005 Winter Simulation Conference*