

# Issues in Heterogeneous GPU Clusters

## A Historical and Usage Analysis

Patrick Carpenter

Department of Computer Science and  
Software Engineering  
Auburn University  
Auburn University, USA  
carpept@auburn.edu

William Symon

Department of Computer Science and  
Software Engineering  
Auburn University  
Auburn University, USA  
symonwi@auburn.edu

**Abstract**— In this paper, we discuss networking issues arising in the design, analysis and use for scientific computing of clusters equipped with graphics processing units. The adoption of graphics accelerators in clusters used for high-performance scientific computing is a fairly recent phenomenon and promises to be an important trend now and into the foreseeable future. After tracing some of the major historical developments leading to the modern-day conception of accelerated heterogeneous clusters, we discuss issues relating to performance, programming and support of such systems. In so doing, we hope to provide the reader with as nearly comprehensive an introductory exposition as possible in the time and space allotted.

**Keywords**— GPUs, clusters, performance, communication.

### I. INTRODUCTION

Beginning at around the end of the 20th century, the high-performance scientific-computing community began to realize the potential of graphics processing units (GPUs) for accelerating highly parallel scientific applications. These applications - such as those used in weather and climate prediction, oil exploration, astrophysical simulations and other so-called "grand challenge" problems - have incredible transformative power in terms of how they affect society. The investment by governments in high-performance computing systems designed to tackle these applications has been substantial and shows no signs diminishing in the near future.

Over the last decade, the use of GPUs to accelerate general-purpose computation - including scientific computation - has gone from being a mostly academic curiosity to a widely deployed and proven solution. There are many reasons for this transformation - indeed, there are probably many more than these authors will discuss - but a few of the seemingly most important ones include the following:

- GPUs are commodity components, and as such mesh better with the modern approach to high-performance computing.
- GPUs offer very attractive FLOPS/dollar and FLOPS/watt ratios, both of which are very important considerations in large-scale data and processing centers.

- GPUs support massive SIMD-like thread and data parallelism and use relatively high-throughput global memory, which enables even higher performance for certain kinds of jobs.
- GPUs have enjoyed greatly eased programmability since the introduction of NVIDIA's Compute Unified Device Architecture (CUDA) and OpenCL.

However, the use of graphics processing units for large-scale scientific computation comes at a cost. The incorporation of graphics accelerators into the design of high-performance cluster computers necessitates a change in the way we think about clusters; the analysis and design of GPU clusters require novel solutions to classical problems (performance, reliability, scalability, etc.) This paper seeks to address some of these issues.

Section II briefly recounts the history of GPU cluster computing from early usage of GPUs for scientific computing, to the first GPU cluster, to the present-day conception of heterogeneous accelerated clusters, including near-future research directions. Section III looks in more detail at how performance of these networked systems can be quantified and how hardware and software advances have influenced performance. Section IV examines the evolution of usability and program models for GPU clusters and the implications for high-performance scientific computing. Section V discusses traditional cluster support services (primarily resource management and reliability) in terms of how the introduction of accelerators changes things. We conclude in Section VI with some additional thoughts highlighting the main ideas. Throughout, we endeavor to compare and contrast heterogeneous GPU-CPU clusters with traditional CPU-only counterparts. For other good, general-purpose introductions to GPU cluster computing and associated issues, the interested reader might consult the work of Kindratenko et al. [1], which is somewhat less current but which addresses issues as relevant today as they were when the paper was written in 2009, or the NVIDIA CUDA [2] documentation (e.g., the CUDA programming or best practices guides) which discusses multi-GPU programming issues and approaches.

## II. PAST, PRESENT AND FUTURE

### A. Introduction

The purpose of this section is to provide the reader with a broad overview of the historical developments and contemporary trends in GPU cluster computing. Subsection 2.2 discusses early usage of GPUs to perform general-purpose computation, before GPUs began to be used in the design of cluster systems. Subsection 2.3 introduces Fan et al.'s landmark 2004 paper which introduces the high-performance scientific computing community to the concept of GPU clusters, and marks the earliest attempts to design and analyze such systems. Subsection 2.4 traces some of the major developments in the design, programming and support of GPU clusters, including a detailed discussion on NVIDIA's Compute Unified Device Architecture (CUDA) and the role it has played in expanding the scope of GPU computing for scientific applications. Section 2.5 looks at how CUDA 4.0 – which has only very recently been released – offers improved capabilities for GPU cluster computing, and argues that this portends increased reliance on accelerated systems.

### B. General-Purpose Programming with Graphics Processing Units

The potential of GPUs for general-purpose scientific computing has been recognized since at least the end of the 20th century [3, 4]. Early general-purpose programming on graphics processing units (abbreviated GPGPU) used the programmable graphics pipeline to perform scientific computation using the GPU as a coprocessor to the CPU. In other words, while the CPU handled I/O, coordination and some computation, a large amount of highly parallel computation was offloaded onto the GPU. This technique led to a marked acceleration of many scientific applications and made relevant somewhat older techniques which had been applied primarily to the SIMD supercomputers of yesteryear.

GPU architecture is typically hierarchical and parallel, with deep instruction pipelines and high-bandwidth, long-latency main memory [2]. NVIDIA's CUDA views GPUs as consisting of an array of multiprocessors, each of which consists of a number of streaming processors. Large amounts of global memory are made available to all streaming processors, whereas each multiprocessor can make use of faster local memory. Code running on GPUs is decomposed into kernels; kernels are treated as a grid (1D or 2D) of thread blocks (1D, 2D or 3D) each of which runs on one multiprocessor. This architecture and threading model has become the de facto way in which practitioners think of GPUs for general-purpose computation, although it CUDA would not be released until much later.

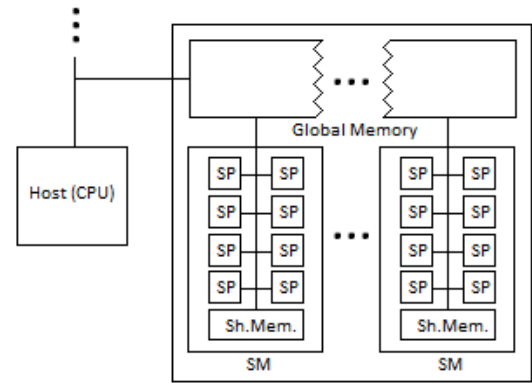


Figure 1. High-level outline of GPU architecture. GPUs have a hierarchical design which must be taken into account to achieve best performance.

### C. Introducing Heterogeneous GPU Clusters

In 2004, Fan et al. [5] introduced the scientific computing community to the idea of incorporating GPUs in the design of cluster systems. They hoped that by doing so they would be able to leverage many of the benefits of GPUs - such as the attractive Flops/dollar and Flops/watt ratios and high memory bandwidth - for large-scale scientific computations. Before this, applications of GPU computing were mostly limited to small-scale applications running on one GPU. In the paper, the authors discuss potential applications of GPU clusters and demonstrate a 4.6x speedup over a CPU-only cluster for a real-world application.

### D. Advances in GPU Clusters

The Top 500 is a ranking of the 500 most powerful supercomputers in the world [6]. As late as November 2007, no GPU cluster system was in the top five. In June 2008, Roadrunner – a graphics-accelerated cluster system developed at Los Alamos National Laboratory (LANL) – came out of nowhere to top the list as the first system to break the petaflop barrier (i.e., it was capable of performing more than one quadrillion floating-point instructions per second). Roadrunner maintained its preeminent position atop the list until November 2009, when Oak Ridge National Laboratory (ORNL)'s Jaguar system – a proprietary massively-parallel supercomputer developed by Cray, and the second system to break the petaflop barrier – overtook it. At the same time, the Chinese Tianhe-1 system – which uses two AMD GPUs per node – emerged onto the scene. In June of 2010, the Chinese Nebulae-Dawning system – which uses NVIDIA Tesla GPUs – moved into second place between Jaguar and Roadrunner. Today, the Chinese Tianhe-1 system has moved into first place, and three of the top five systems are graphics-accelerated systems. Since late 2007, GPU clusters have joined (and seem to be surpassing) Cray systems as the dominant HPC architecture.

### E. GPU Clusters – Today and Tomorrow

The introduction of CUDA 4.0 between February and April 2011 gives a strong indication of the direction in which the community can expect GPU technology to be heading. NVIDIA [2] cites three main areas of improvement in CUDA 4.0: usability, multi-GPU programming capabilities, and developer

tools. Each of these areas bears discussion in terms of its impact on GPU cluster technology:

1) *Usability & Programmability*: For the purposes of this discussion, the most important innovations in this category include how host threads share GPUs and no-copy pinning of system memory. With CUDA 4.0, multiple host threads can share a single GPU, and a single host thread can share multiple GPUs; this represents a significant paradigm shift in terms of application structuring to leverage varying GPU resources. No-copy pinning of system memory improves further upon the benefits of pinned host memory (mentioned earlier)

2) *Multi-GPU Support*: Unified Virtual Addressing (UVA) and GPUDirect 2.0 bring improvements in both programmability (UVA brings the host/device domain closer to a truly global address space) and in performance (by reducing GPUs' dependence on the CPU to perform communication and I/O). The importance of these advances to GPU cluster computing should not be understated.

3) *Developer tools*: With the release of CUDA 4.0, NVIDIA has added multi-GPU support to the `cuda-gdb` debugger.

In summary, we see a trend towards increased use of and reliance on the use GPUs in high-performance scientific computing applications. The specific benefits of some of these technologies will be described later in this paper.

### III. PERFORMANCE ISSUES

#### A. Introduction

The purpose of this section is to expound on the approaches, techniques and tools which have been, are being and promise to become essential in design for and analyzing performance. After all, the primary motivation for the introduction of special-purpose graphics accelerators in general-purpose CPU clusters has nothing to do with programmability or services (in many respects, leaving GPUs out would drastically facilitate programming and service implementation – problems which have been and are continuing to be addressed) and everything to do with speed, and in particular, reducing the amount of time devoted to (certain common kinds of) computation.

#### B. Basic Concepts

1) *GPU Performance*: of GPUs are able to offer incredible floating-point processing power compared to CPUs. This is due to a fundamental design difference: whereas CPUs are designed in such a way as to minimize the performance impact on pipelined superscalar microprocessor architectures of branch divergence and main-memory accesses, GPUs are designed with 3D graphics workloads in mind, which typically require that the same program be applied to all the elements in a structure. Since scientific applications often fit neatly within this class of access pattern (consider, for instance, the ubiquity of linear algebraic algorithms in scientific applications), GPUs seem a good fit for them. There are two main categories of optimization in GPU programming: instruction optimization

and memory optimization. In the category of instruction optimizations, two techniques are of fundamental importance: reducing branch divergence and increasing the occupancy (the number of active warps divided by the maximum number of warps possible). In the category of memory optimizations, techniques for exploiting the various levels of the GPU memory hierarchy in the most efficient way possible have been widely discussed in the literature [2]. Profilers are available for CUDA-enabled GPUs and these have proven as useful as their CPU counterparts. In general, scientific applications which display large amounts of data-parallelism experience the highest levels of performance improvement when ported to the GPU.

2) *Cluster Performance*: Cluster computes consist of many separate processing units connected together in some fashion to allow them to work as a continuous whole. Cluster computing can often provide a less finically intensive way to obtain a large amount of computing resources than designing and building a single unit of the same capacity. Several of the main advantages of cluster computing stem from the fact that clusters can be constructed utilizing commodity off the shelf hardware components. The first advantage this grants the end user is cost. Due to the skyrocketing of specialized hardware design and construction cost the ability to use a product that is already in the market for a fixed price gives the user a huge finical advantage. In addition the high availability of these commodity products can lead to lower down time as well as increased reliability as the products being used have hopefully been thoroughly tested in the marketplace. Finally, since COTS hardware can be used, many organizations already have this hardware deployed on a network throughout their organization, often as common workstations. Through the use of cycle stealing this workstations can be utilized as a cluster computer during their downtime, granting the organization a large amount of computing resources for practically no cost. In addition to the finical advantages obtained through the use of cluster computing performance increases can be seen as well. First the data bandwidth is increased due to the large number of processing units which are available to perform computations allowing for more computations to be performed in a shorter amount of time. Reliability is also increased as the data processing is spread across multiple processing units. By distributing the processing tasks across many nodes if one node fails a minimal amount of data will be lost that has to be recalculated, resulting in a minimal time increase to the process as a whole. Finally as new hardware becomes available it will be easier to upgrade and maintain the system sine commodity products were used in the first place. In addition since the commodity products are consistently improving the system should be able to be upgraded fairly regularly helping to ensure that any unforeseen future processing requirements can be meet successfully.

3) *GPU Clusters – Considerations*: The marriage of GPUs to cluster computing solutions makes necessary certain considerations which are not necessary when considering either separately [5]. For instance, applications which were compute-

bound on traditional cluster systems may become communication-bound on accelerated systems, due to the dramatically increased processing power. Indeed, the best one can hope for when using an accelerated system is to change a compute-bound problem into a communication-bound problem. Clearly, this is especially attractive for the class of problems affectionately termed “embarrassingly parallel”. Another interesting consideration deals with the question of network topology; since GPUs generally function as computational co-processors to the host CPU, and since GPUs are typically connected to CPUs within a node, there are limitations on the way in which systems can be physically and logically configured and, depending on the interconnect which is used (both inter-node and between CPU and GPU within a node) the effects on overall performance can be measurable.

### C. Analysis of Systems

The system developed by Fan et al. in their 2004 paper [5] consisted of 32 nodes connected with 1Gb Ethernet, with each node containing two 2.4Ghz Xeon processors, 2.5 GB of memory and an NVIDIA GeForce FX 5800 Ultra GPU. The GPU was connected to the host CPU via an AGP 8x bus capable of 2.1GB/sec downstream (CPU to GPU) and 133 MB/sec upstream (GPU to CPU). We will use this early system as means of illustrating several performance issues with GPU cluster systems.

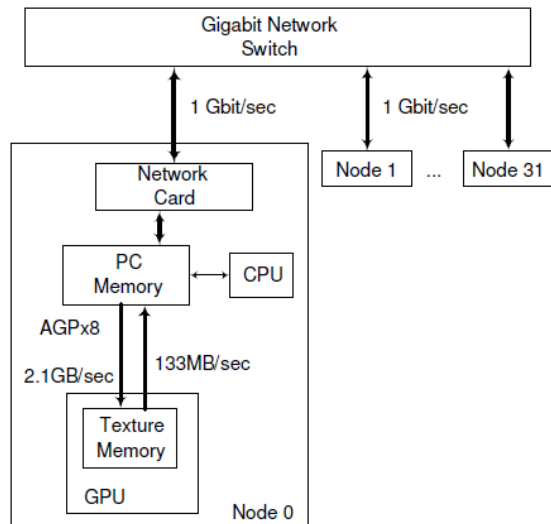


Figure 2: The architecture of the system developed by Fan et al. in their 2004 paper [5]

Consider one GPU communicating with another GPU in the system. In the above system, the sending GPU must first write all results to global memory, and then signal the CPU. The CPU then transfers the data to the host memory and sends it (e.g. using MPI) to another node’s CPU. The CPU then must communicate this information to the GPU via a transfer to GPU memory, at which point the communication is complete. In the Fan system, the delays are the following: upstreaming data of size N bytes from the sending GPU to the host; sending N bytes

of data between hosts using MPI; downstreaming data of size N bytes to the receiving GPU. At the time Fan et al. built their system, these operations could not be pipelined, although recent innovations in CUDA (including GPUDirect and pinned host memory) make this possible.

## IV. PROGRAMMING ISSUES

### A. Introduction

The purpose of this section is to explore some of the issues involved in programming GPU clusters. We couch this exploration in terms of applicable parallel programming models and the hybrid nature (in terms of Flynn’s taxonomy) of GPU clusters and, to an equally valid extent, GPUs themselves. Subsection 4.2 elaborates the traditional message-passing programming model for GPUs and GPU clusters. Subsection 4.3 discusses global address space aspects of GPUs (in a sense, the GPU is a global address space parallel environment, and the trend is away from explicit message-passing). Section 4.4 looks at the Map-Reduce programming paradigm and cloud computing and asks whether there is a place for GPUs in this arena. In order that GPU cluster computing remain viable in the future, it must keep pace with advances in programming models and evolving scientific software development paradigms.

### B. Message Passing

Message passing is a form of programming for parallel processes which requires cooperation between the processes themselves. In message passing one process explicitly asks for permission to send information to another process. Once the second process responds in an affirmative manner, then first process can begin passing it data. This cooperation is not always apparent in code. One popular interface for accomplishing this is the Message Passing Interface (MPI).

“As general-purpose graphics processing units are adopted more and more widely, application developers will need well designed and high performance libraries to use on this new hardware” [7]. In his 2009 paper on the subject, Lawlor presents one possible solution for this problem known as cudaMPI. The cudaMPI library utilized Compute Unified Device Interface (CUDA) to provide an MPI-like interface for passing messages to achieve GPU-to-GPU communication as shown below in Figure 3.

The cudaMPI library functioned by being an arbitrator between the CUDA GPU and the MPI network allowing the popular MPI parallel programming interface to be used with data stored on the GPU instead of having to use the CUDA interface directly. At the time of publication, the cudaMPI interface had the required basic functionality to benchmark the standard but not all MPI functions were implemented in it.

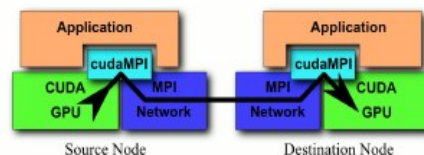


Figure 3. Application use cudaMPI to send GPU data across an MPI network [7]

This approach appears simplistic in implementation, essentially providing a bridge between two common interfaces and is currently incomplete, missing key MPI features as well as lacking GPU-communication primitives and addressing the issue of load balancing. If these outstanding features were added `cudaMPI` could provide a powerful interface to aid in the programming of GPU clusters.

### C. Global Address Space

Global address space (GAS) programming is a programming technique that can be utilized to pass data between parallel processes. In this programming model the parallel processes all have access to the same memory space. In a simplified view one process can simply write data to this memory area and then, when needed, this data can be read by another process. There are several variations on this theme but one of the most common ones is known as partitioned global address space (PGAS). In this technique the memory space is partitioned into a global space and a local space so that the programmer is aware of when data passing is taking place, unlike normal GAS.

As was seen in the section above, as the importance of GPU clusters increase researchers are looking for ways to apply processes that have already been proven to work in parallel processing to this new environment. This is the case with GAS as well. In their 2011 paper Karantasis declared that "... GPU-based clusters are expected to grow in the next year, and that much of their success will depend on the provision of the appropriate programming tools" [8]. Karantasis goes on to describe two cases, one utilizing Intel Cluster OpenMP and the other an extended version of Pleiad, to accomplish this goal.

In the case of Intel Cluster OpenMP an OpenMP implementation was integrated into their cluster scheme. The use of the low-level CUDA Driver API was then integrated with the OpenMP implementation. This allowed cooperating to be "feasible if the source code that is destined to run on the host is compiled with `icc` and the source code of the CUDA kernels that will operate on the GPU device side is compiled with NVIDIA CUDA compiler driver (`nvcc`).

The other alternative involved a modified version of Pleiad, a cluster middleware which is based on the java platform. Data sharing in this scenario "is based on objects and instead of being tightly coupled with a specific consistency protocol, Pleiad incorporates several implementations of consistency maintenance, that can be potentially interchanged even during run-time" [8]. `JCuda` was used to incorporate the functionality of Pleiad with `Cuda` on the underlying GPUs.

Tests were then run on both of these system configurations and the results proved that both of these approaches appeared to be valid solutions to the data passing issue within a GPU cluster. This can be used as a justification for "the implementation of middleware that will be based on the concept of shared memory abstraction and will also be able to offer an efficient programming model for heterogeneous GPU clusters" [8].

### D. Map-Reduce and the Cloud

`Mars`, a MapReduce framework for graphics processing units, was developed by He et al. [9] to address the difficulty of using

GPUs for parallel and distributed computation [10]. A. Mooley et al. later investigated the use of the MapReduce programming model on GPU clusters, a significant step forward compared to the work of He et al., which used only one GPU. A more recent and more promising study by Farivar in 2009 demonstrates a four-node GPU cluster outperforming a 62-node cluster for a representative Hadoop job. These advances are both recent and exciting in that they demonstrate the amenability of the increasingly popular MapReduce model for GPU clusters. Given that programmability has traditionally been a weakness of GPGPU, this line of research has the potential to radically change the way in which the scientific computing community views graphics accelerators and lead to an explosion in usage and performance.

## V. SUPPORT SERVICES

### A. Introduction

The purpose of this section is to investigate how the introduction of GPU accelerators into cluster computer design has affected typical cluster support services, such as resource allocation (e.g. job scheduling, task migration, etc.), reliability (e.g. check-pointing), etc. Subsection 5.2 discusses cluster resource allocation and management issues from the point of view of GPU clusters. Subsection 5.3 describes reliability and other assorted issues arising in the design and management of heterogeneous accelerated clusters. As special-purpose coprocessors, GPUs require that certain extra arrangements be made above and beyond what would otherwise be required for traditional clustered systems, and the challenges associated with providing these extra services cannot be ignored at scale.

### B. Resource Allocation and Management

Resource allocation and management on GPU clusters is interesting inasmuch as resource allocation and management becomes more complicated with increasing degrees of system heterogeneity [1]. In many ways, the introduction of GPUs into cluster systems constitutes a startling new dimension in heterogeneity; nodes now not only differ in terms of the processor speed or memory capacity, but also in terms of the computational abilities (code which needs a GPU cannot be executed on nodes not containing a GPU). In general, the easiest way to make resource allocation and management as painless as possible is to adopt a regular and consistent approach to distributing accelerator, such as by assigning one accelerator per node [5]. Sharing accelerated nodes between processes with different computational requirements is a difficult problem which requires more research to adequately address.

### C. Reliability and Miscellaneous Services

The problem of reliability - or fault tolerance - in high-performance scientific computing increases with the size of high-performance computers, including those augmented with graphics accelerators [11]. For this reason Laosooksathit et al. describe in a paper in 2010 two protocols for implementing a checkpoint/restart mechanism compatible with GPU clusters. The reason why special attention must be given at all to this problem stems from a traditional limitation of GPUs to perform

I/O and inter-node communication independently; in many respects, CUDA 4.0 changes the situation, but the idea of reliability in GPU clusters is still an interesting and challenging one. Since the purpose of GPUs is to perform massive amounts of computation, the failure of a GPU could result in significant loss of application progress.

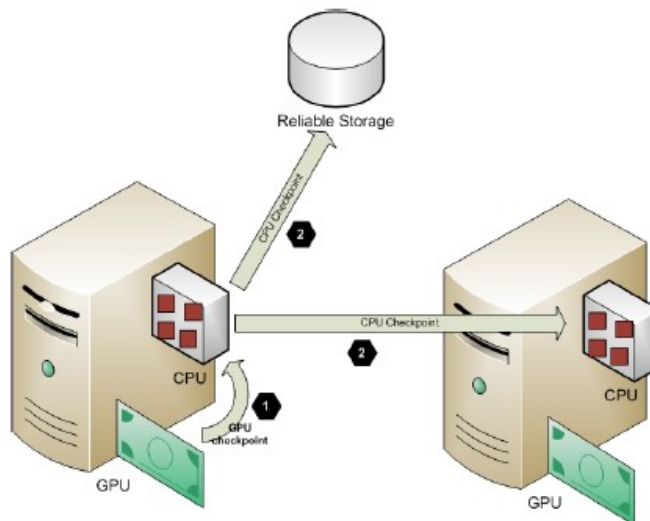


Figure 4: High-level GPU check-pointing protocol [11]

The authors describe two protocols: the first relies on simple CUDA memory transfer operations, and the second relies on CUDA streams to allow concurrent transfer and execution on the host (Figure 4 demonstrates the basic idea). Both are extensions of the VCCP check-pointing mechanism which is assumed to be used on the hosts. Although the authors do not describe an implementation of the protocol and related performance measurements, they do perform simulations which produce promising results: overheads of under a few percent (and often at only a fraction of a percent). These authors believe that more work is needed in these areas in order to better understand the need for and performance of robust fault-tolerance services on GPU clusters.

## VI. SUMMARY AND CONCLUSIONS

The purpose of this paper has been several-fold. First, we have sought to introduce the reader to the idea of GPU cluster computing, that is, using graphics processing units to increase the computational power of cluster systems. We motivated this introduction with a discussion of the importance of scientific computing and an explanation of its reliance on high-performance computing resources. We briefly recount the late historical developments leading to GPU clusters and point out some of the advances which have brought the field to the state of the art. We then change tact and delve into considerably more detailed observations regarding performance, programming and support service issues in GPU cluster computing. In the first of these categories, we reveal that the project of accelerating

computation in clusters makes communication costs an inherently more serious performance bottleneck. In the second category, we see how steps have been taken to facilitate the usage of GPU cluster systems by introducing more implicit, high-level programming features for use by application developers. In the third category, we survey some of the attempts which have been made – and which are currently underway – to bring GPU cluster systems closer to the mainstream of high-performance computing in terms of service and flexibility, and conclude that – in general – this is the area in which modern GPU clusters seem to be most lacking.

We believe that the community’s collective experience in evolving GPUs from special-purpose hardware for traditional graphics tasks to accelerators for parallel, distributed, high-performance scientific jobs will pave the way for advances in the field for years to come. Introducing GPUs to cluster computer systems has forced the experts to reevaluate the ways they think about performance, programmability and services for cluster systems, and the community has risen to the challenge.

## REFERENCES

- [1] V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. M. Hwu, "GPU Clusters for High-Performance Computing," In IEEE International Conference on Cluster Computing and Workshops, 2009 (Cluster '09)
- [2] NVIDIA Developer Zone, "CUDA Toolkit 4.0," Retrieved online 4/25/2011 from <http://developer.nvidia.com/cuda-toolkit-40>
- [3] W. Heidrich, R. Westermann, H. Seidel and T. Ertl, "Applications of pixel textures in visualization and realistic image synthesis," In Proceedings of ACM Symposium on Interactive 3D Graphics 1999.
- [4] B. Jobard, G. Erlebacher and M. Y. Hussaini, "Lagrangian-Eulerian advection for unsteady flow visualization," In Proceedings of IEEE Visualization 2001.
- [5] Z. Fan, F. Qiu, A. Kaufman and S. Yoakum-Stover, "GPU cluster for high performance computing," Proceedings of the 2004 ACM/IEEE conference on Supercomputing (November 2004)
- [6] Top500, "TOP500 Supercomputing Sites," retrieved online 4/25/2011 from <http://www.top500.org/>
- [7] O. S. Lawlor, "Message passing for GPGPU clusters: CudaMPI," In IEEE International Conference on Cluster Computing and Workshops, 2009 (CLUSTER '09), pp 1-8.
- [8] K. I. Karantasis, E. D. Polychronopoulos, "Programming GPU-clusters with shared-memory abstraction in software," 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2011.
- [9] B. He, W. Fang, Q. Luo, N. K. Govindaraju, T. Wang, "Mars: a MapReduce framework on graphics processors," In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08)
- [10] S. Shehu, "Scaling-up GPUs for 'Big Data' analytics - MapReduce and fat nodes," retrieved online 4/25/2011 from <http://www.smedirector.com/2010/10/16/scaling-up-gpus-big-data-analytics%E2%80%93mapreduce-fat-nodes/>
- [11] S. Laosooksathit, N. Naksinehaboon, C. Leangsuksan, A. Dhungana, C. Chandler, K. Chanchio and A. Farbin, "Lightweight checkpoint mechanism and modeling in GPGPU environment," In Proceedings of HPCVirt2010 Workshop (EUROSYS 2010).