# Cluster-Wide Multi-GPU Computing

Jeff A. Stuart[*]
UC Davis

John D. Owens[†]
UC Davis

## 1 Introduction

GPUs are no longer a niche for high-performance computing (HPC), they are at the forefront and at the time of this writing, a GPU cluster now holds the record as the world's fastest supercomputer. In order to effectively program such a large machine, the community must have software support in the form of application middleware, and extended hardware support from vendors. We classify the three types of support as 1) mechanisms, 2) application-programming interfaces (APIs), and 3) programming models.

As GPU architectures are very guarded in comparison to CPU architectures, it is near impossible for academics to meaningfully modify hardware. Thus, we must focus on the software aspect and hope use our lessons learned to help steer GPU and chipset architects in their future design decisions. We present our results from implementing various types of middleware below.

## 2 Mechanisms

Perhaps the most lacking mechanism in the GPU is the ability for the GPU to send messages/signals to the CPU and its PCI-e siblings, including in many cases other GPUs. We developed a mechanism we refer to as callbacks [Stuart et al. 2010b]. Callbacks provide the GPU programmer with many new functions. The GPU can make system calls, initiate DMA transfers, and execute arbitrary system code. Callbacks also grant the GPU pseudo autonomy from the CPU. We say pseudo autonomy because we must use software tricks to achieve callbacks at this stage, due to lacking the proper support in hardware.

### 2.1 APIs

On top of callbacks, we built an API called "Distributed Computing for GPU Networks" (DCGN, pronounced Decagon) to allow GPUs and CPUs to pass messages amongst each other [Stuart and Owens 2009]. DCGN is very similar in appearance to MPI. DCGN provides support for the basic collectives (broadcast, scatter, gather, all-to-all) as well as both synchronous and asynchronous point-to-point communications.

Mapping of GPU control primitives (threads, blocks, SMs) to a typical MPI rank is non intuitive, thus we introduced "slots". Slots intelligently multiplex ranks from MPI to a GPU. The user creates a mapping at application-startup time, and chooses an arbitrary number of slots per GPU. We give this control because even in our limited tests, there was no single-best mapping of ranks to GPU control primitives.

### 2.2 Models

MapReduce [Dean and Ghemawat 2004] is a programming model from functional programming that works with data-parallel tasks. All previous MapReduce implementations on the GPU were limited to an in-core problem on a single GPU.

[*]e-mail: stuart@cs.ucdavis.edu
[†]e-mail:jowens@ece.ucdavis.edu

We created a new cluster-wide, multi-GPU MapReduce library called (GPMR) to take advantage of the new clusters GPU clusters available today [Stuart and Owens 2011]. We do not implement the entire MapReduce execution environment as some aspects (HDFS, Fault tolerance) are orthogonal to running MapReduce on a cluster of GPUs. We focused on extending and modifying the MapReduce model to minize communication. GPMR uses a method for grouping segments of input data into "chunks". We added several substages to the Map phase including previously existing substages (Partial Reduce, Combine) and added our own substage (Accumulation). All the substages reduce communication, either over the PCI-e bus or the network interconnect, or both.

### 2.3 Applications

To test our mechanisms, APIs, and models, we implemented several applications. For DCGN, we implemented several toy applications and benchmarks. We discuss all of these in detail in the paper.

We implemented a high-quality volume renderer using GPMR [Stuart et al. 2010a]. Volume rendering transfers well to MapReduce. We subdivide the domain into many small tiles. The map stage casts partial rays through each tile. The reduce stage first sorts each set of rays and performs alpha blending, then stitches the final image.

Another application we built is a generic resource-manager for heterogeneous (CPU and GPU) clusters [Budge et al. 2009]. The applications runs user-supplied kernels on both the CPU and GPU, handles data movement, intelligently schedules work based on data locality, and tries to use hints from the programmer to intelligently schedule compute resources.

## References

BUDGE, B., BERNARDIN, T., STUART, J. A., SENGUPTA, S., JOY, K. I., AND OWENS, J. D. 2009. Out-of-core data management for path tracing on hybrid resources. *Computer Graphics Forum 28*, 2 (Apr.), 385–396.

DEAN, J., AND GHEMAWAT, S. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design & Implementation*.

STUART, J. A., AND OWENS, J. D. 2009. Message passing on data-parallel architectures. In *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*.

STUART, J. A., AND OWENS, J. D. 2011. Multi-GPU MapReduce on GPU clusters. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*.

STUART, J. A., CHEN, C.-K., MA, K.-L., AND OWENS, J. D. 2010. Multi-GPU volume rendering using MapReduce. In *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing / MAPREDUCE '10: The First International Workshop on MapReduce and its Applications*, 841–848.

STUART, J. A., COX, M., AND OWENS, J. D. 2010. GPU-to-CPU callbacks. In *UCHPC 2010: (Euro-Par 2010 Workshops)*.