Eurographics Symposium on Parallel Graphics and Visualization (2013)
F. Marton and K. Moreland (Editors)

GPU Acceleration of Particle Advection Workloads in a Parallel, Distributed Memory Setting

David Camp[1], Hari Krishnan[1], David Pugmire[2], Christoph Garth[3], Ian Johnson[1], E. Wes Bethel[1], Kenneth I. Joy[4], and Hank Childs[1]

[1]Lawrence Berkeley National Laboratory, CA, USA
[2]Oak Ridge National Laboratory, TN, USA
[3]University of Kaiserslautern, Germany
[4]University of California, Davis, CA, USA

Abstract

Although there has been significant research in GPU acceleration, both of parallel simulation codes (i.e., GPGPU) and of single GPU visualization and analysis algorithms, there has been relatively little research devoted to visualization and analysis algorithms on GPU clusters. This oversight is significant: parallel visualization and analysis algorithms have markedly different characteristics – computational load, memory access pattern, communication, idle time, etc. – than the other two categories.

In this paper, we explore the benefits of GPU acceleration for particle advection in a parallel, distributed-memory setting. As performance properties can differ dramatically between particle advection use cases, our study operates over a variety of workloads, designed to reveal insights about underlying trends. This work has a three-fold aim: (1) to map a challenging visualization and analysis algorithm – particle advection – to a complex system (a cluster of GPUs), (2) to inform its performance characteristics, and (3) to evaluate the advantages and disadvantages of using the GPU. In our performance study, we identify which factors are and are not relevant for obtaining a speedup when using GPUs. In short, this study informs the following question: if faced with a parallel particle advection problem, should you implement the solution with CPUs, with GPUs, or does it not matter?

Categories and Subject Descriptors (according to ACM CCS): D.1.3 [Computer Graphics]: Concurrent Programming—Parallel programming

1 Introduction

Very large simulation data is increasingly visualized in environments where GPU acceleration is available. Traditionally, this environment has come from "visualization clusters": smaller GPU-based supercomputers that are optimized for loading, processing, and rendering data [BVS*11]. The alternative to these specialized clusters is to perform visualization on the general-purpose supercomputers where the simulations themselves are run. These supercomputers now regularly also contain GPUs, as a way of obtaining additional computing power with reduced cost and power consumption.

Surprisingly, most instances of today's parallel visualization software – e.g., VisIt [CBW*12], ParaView [AGM*12], and VTK [SML96] – have CPU implementations for their algorithms, and only exploit the GPU through basic OpenGL calls for rendering. This is because development of these software packages began over a decade ago when GPUs were a rarity in supercomputing environments. However, as GPUs become more prevalent on supercomputers, we must design and evaluate appropriate algorithms.

With this study, we consider parallel particle advection in distributed memory GPU environments. Particle advection – displacing particles so that they are tangent to the velocity field – is a foundational element of many visualization algorithms for flow analysis, including streamlines, pathlines, stream surfaces, and calculating Finite-Time Lyapunov Exponents (FTLE). Particle advection is a particularly difficult form of a non-embarrassingly parallel algorithm, as the work needed to complete the problem is not known a priori. Additionally, the workload varies greatly from problem to problem. Streamline calculation often involves advecting few particles for long distances, while FTLE calculation of-

© The Eurographics Association 2013.

DOI: 10.2312/EGPGV/EGPGV13/001-008

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

ten involves advecting many particles for short distances. In turn, our study considers a range of scenarios, varying over particle count, distance traveled, and vector field. While one contribution of this paper is our GPU-based parallel particle advection algorithm, we believe its most significant contribution is the performance study, which illuminates the factors which make CPU and GPU performance differ and the magnitude of their effects.

2 Related Work

McLouglin et al. recently surveyed the state of the art in flow visualization [MLP*09], and the large majority of techniques they described incorporated particle advection. As mentioned in the introduction, the computational workload for these particle advection-based techniques vary. On the low end of computational demands, streamlines, which display the trajectory of particles placed at seed locations, can involve advecting just a few particles. In the middle, stream surfaces, which advect a seeding curve (or, rather, particles along that curve) to create a surface, require potentially tens of thousands of particles to be advected. At the high end, FTLE calculations advect a particle for every node in a mesh and compare how much nearby particles diverge, determining the rate of separation throughout the volume. Further, although FTLE techniques are often assumed to only consider short distances, application areas such as ocean modeling require long distances [OPF*12], representing an extreme computational load.

Many visualization algorithms have been ported to, and optimized for, the GPU [AFM*12]. While less work is devoted to parallel GPU clusters, there still has been significant research in achieving load balancing and scalability for rendering, both for surfaces [HHN*02, SMW*04, BRE05] and volumes [MSE06, MMD08, FCS*10, FK10, ADM12]. Very few papers are devoted to studying visualization algorithms on parallel GPU clusters, with a notable exception on isosurfacing [MSM10]. The study most closely related to our own uses the GPU to perform LIC flow visualization [BSWE06], although the parallelization approach is significantly different and focuses on the problem of dense particle seeding on curved surfaces.

A summary of strategies for parallelizing particle advection problems on CPU clusters is summarized in [PPG12]. The basic approaches are to parallelize-over-data, parallelize-over-particles, or a hybrid of the two [PCG*09]. Recent results using parallelization-over-data demonstrated streamline computation on up to 32,768 processors and eight billion cells [PRN*11]. Alternate approaches use preprocessing to study the patterns of the flow and schedule processing of blocks to optimize performance [NLS11].

The most comparable study to our own focused on streamlines with multi-core CPUs and showed that hybrid parallel techniques are highly beneficial [CGC*11]. However, the study does not consider the performance characteristics of many-core devices, i.e., do the benefits from shared memory parallelism persist when the number of cores per node reaches hundreds or thousands? Or do limitations emerge from the reduced computational power of an individual core on a many-core device or from the latency in accessing that device?

3 Particle Advection Overview

The fundamental unit of work for particle advection is an **advection step**, which is the displacement of a particle for a short distance from one location to a nearby one. An **integral curve** is the total path the particle travels along and it is formed by the sequence of advection steps from the seed location to the terminal location. The calculation of advection steps must be carried out sequentially, as advecting particles is a data dependent process. Explicitly, the N^{th} advection step for a particle must know the location of where the $(N-1)^{st}$ advection step terminated.

A traditional scheduling view, which considers a fixed number of operations with known dependencies between these operations, is too simplistic when it comes to particle advection, since the total number of operations (i.e., the total number of advection steps) is not known a priori. The number of advection steps for any given particle varies, based on whether it advects into a sink, exits the problem domain, or meets some other termination criteria.

When considering data sets so large that they can not fit into memory, there are scheduling difficulties in getting the particle and appropriate region of the vector field on the same resource to carry out the advection step. In this study, we employed a parallelization-over-data approach; the problem domain is divided into pieces and each task operates on one piece of the domain. Particles are advected on a task as long as they remain within that task's piece. Particles that advect into other pieces are communicated to the corresponding task. Our motivation for studying this particular parallelization strategy was that it mirrored the conditions encountered with in situ processing on GPU-based supercomputers, where the simulation data is predivided into pieces and likely already located on the GPU.

4 Algorithm Overview

The algorithm extends our previous work [CGC*11] to GPUs. It has two phases: initialization (§4.1) and advection (§4.2). Further, the advection implementations differ for the GPU and CPU and are discussed separately (§4.2.3 and §4.2.4, respectively). Implementation details are discussed in §4.3.

4.1 Initialization Phase

The algorithm's initialization phase consists of three parts: (i) loading data, (ii) constructing a **piece map** of where

© The Eurographics Association 2013.

data resides, and (iii) particle creation and initialization. For (i), each task reads its domain piece directly from disk. The GPU implementation then transfers the data as a texture map to GPU memory, along with other meta-data. For (ii), each task creates a map between domain pieces and tasks. For (iii), each task will create the starting number of particles, defined by user input, and prepare them for processing by placing them in a queue.

4.2 Advection Phase

4.2.1 Queues

The particles are maintained in three queues, which drive how they are processed. The **active queue** contains particles that need to be advected. The **finished queue** contains particle that have completed advecting. The **inactive queue** contains particles that cannot be further advected on the current task, but also cannot be placed in its finished queue.

4.2.2 Advecting Particles

The goal of this phase is to promote all particles from the active queue to the finished queue. Each task continuously iterates over a loop until all tasks declare themselves **finished**. An individual task declares itself finished when all particles it is responsible for have completed, i.e., the size of its finished queue is equal to the size of its active queue at the beginning of the algorithm. However, finished tasks continue participating in the algorithm, since individual tasks that are finished may contain portions of the domain that are necessary for other tasks to finish.

Each task's loop iteration consists of three steps: (i) advect, (ii) inspect, and (iii) communicate. For (i), the task examines its active queue and instructs a group of particles to advect. The size of the group and details of the advection vary between GPU and CPU implementations (see §4.2.3 and §4.2.4). For (ii), the particles resulting from step (i) are placed in one of two queues. Particles that have advected outside the task's piece are placed in the inactive queue. Particles that are done advecting and originated on the current task are placed in the finished queue, while those that originated on a different task are placed in the inactive queue. For (iii), all particles in the inactive queue are sent to the appropriate task using the piece map. Further, messages from other tasks are read. The particles in those messages correspond to particles that are done advecting (and placed in the finished queue) or need more advecting on this task (and placed in the active queue). Finally, the task assesses if it is finished and the tasks coordinate to determine if they are all finished.

4.2.3 GPU Advection

In the GPU advection implementation, there are three memory allocations on the GPU: (i) vector field data, (ii) meta-data, and (iii) particle data. The vector field (i) is stored as a texture map, enabling hardware interpolation for arbitrary locations in space. The meta-data (ii) contains information for determining if a particle has exited the task's piece (i.e., the piece boundary and origin) and also information about when to stop advecting each particle. The vector field and meta-data are transferred only once, during the initialization phase, to the GPU and used throughout the advection phase. The particle data (iii) contains the location, time, and status for each particle that needs to be advected. The size of the particle data array is set with a user defined value.

During the advection phase, particles are pulled from the active queue and their information is copied to the particle data array for transfer to the GPU memory. Most commonly, the active queue contains less than two million particles and they are all sent together to the GPU for advection. If there are more than two million particles, only the first two million are sent. After the data transfer is complete, the GPU cores advect the particles. After the particles are advected, the GPU advection data is transferred to the CPU memory. Then the GPU particle advection data is transferred to the CPU particle memory. This advection process is repeated until all particles are advected.

4.2.4 CPU Advection

In the CPU advection implementation, a team of worker threads handle the advection of the particles. The number of worker threads is arbitrary, although we typically chose the number to reflect the number of cores available to each task. Each worker thread fetches a particle from the active queue and advects the particle. The particle is then transferred directly to the inactive queue or finished queue, as described in §4.2.2.

4.3 Implementation Details

Each task has a thread to handle all MPI communication and a thread (GPU) or threads (CPU) to handle the advection. The communication thread handled the inspect and communicate steps (i.e., (ii) and (iii) as described in §4.2.2). The only synchronization between the threads occurred when accessing the particle queues (§4.2.1), and this is handled using mutexes and conditions.

The GPU code was written in CUDA. We did not know what CUDA block size would be best to divide this work up among the multiprocessors. With some manual auto-tuning, we found that a CUDA block size of a 128 particles was the best block size for a large amount of particles. The auto-tuning results found that with a small number of particles, less than the number of cores on the GPU, a smaller CUDA block size would perform better. But this performance increase was very small, less than 0.3 ms, so a CUDA block size of a 128 was used for all numbers of particles being advected.

Both CPU and GPU use a 4^{th}-order Runge-Kutta technique, with a constant step size, to advect particles. The tasks do non-blocking point-to-point communication to exchange particles, i.e., *MPI_Isend* and *MPI_Irecv*. Finally, since this study focuses on particle advection workloads,

© The Eurographics Association 2013.

no visualization-centric transformations (e.g., streamlines, stream surfaces, or FTLE) are actually performed at the end of the algorithm.

5 Study Overview

5.1 Configurations

Our study was designed to provide coverage over a variety of particle advection workloads. We varied four factors:

1. CPU versus GPU (2 options)
2. Data set (3 options)
3. Number of particles (10 options)
4. Duration of advection (5 options)

We ran the cross-product, meaning $2 \times 3 \times 10 \times 5 = 300$ tests overall. The variants for each factor are discussed below.

5.1.1 CPU Versus GPU

All tests were run with and without GPUs. The implementation details are described in §4.2.3 and §4.2.4.

5.1.2 Data Sets

We considered three data sets to ensure coverage. Each data set was a single time slice, meaning we studied steady state flow. Each task operated on a data block of 500 x 500 x 500 cells, i.e., a $1,000^3$ data set divided over the eight nodes. Figure 1 shows different particle advection-based visualizations on all three data sets.

Fusion: This data set is from a simulation of magnetically confined fusion in a tokamak device by the NIMROD simulation code [SGG*04]. To achieve stable plasma equilibrium, the field lines of the magnetic field need to travel around the torus in a helical fashion. This data set has the unusual property that most integral curves are approximately closed and traverse the torus-shaped vector field domain repeatedly.

Thermal Hydraulics: In this data set, twin inlets pump air into a box, with a temperature difference between the inlets. The air mixes in the box and exits through an outlet. Mixing of "hot" and "cold" air, residence time in the box, and identification of both stagnant and highly turbulent regions are areas of active study. The simulation was performed using the NEK5000 code [FLPS08].

Astrophysics: This data set is from a simulation of the magnetic field surrounding a solar core collapse, resulting in a supernova. The simulation was computed by a GENASIS simulation [ECBM08], a multi-physics code being developed for the simulation of astrophysical systems involving nuclear matter [CRE*05].

5.1.3 Number of Particles

We placed a variable number of seeds into each of our data blocks, reflecting the varying number of particles required to carry out particle advection-based analyses. The lowest number of particles per data block was just a single particle and the highest number of particles per data block was 250^3. These workloads are representative of use cases such as streamlines, stream surfaces, and coarser FTLE analysis, among others. The options for the numbers of particles per data block were 1^3, 5^3, 15^3, 25^3, 40^3, 50^3, 65^3, 80^3, 100^3, and 250^3. Over all tasks, the lowest number of particles was just 8, while the highest number was over 120 million.

5.1.4 Duration of Advection

The duration of the advection (i.e., the number of advection steps) depends on the **termination criterion** and thus varies from use case to use case. Some termination criteria require that particles advect for a fixed amount of simulation time, while others require that they travel a certain distance. Still others have application-specific requirements, such as ensuring the particle enters a given region. Of course, each of these termination criteria are met by carrying out some number of advection steps, and the total number of advection steps dictates how much computation is necessary. To reflect this variation in particle advection workload, we made five categories for duration: *tiny* (50 steps), *little* (250), *short* (1,000), *medium* (5,000), and *long* (20,000).

5.2 Runtime Environment

We performed tests on Dirac, a machine at Lawrence Berkeley's NERSC supercomputing center. Each node on Dirac contains two Intel quad-core Nehalem processors (eight cores overall) running at 2.4 GHz, and capable of 19.2 GigaFLOPs. The GPU on a Dirac node is an NVIDIA Tesla C2050 (Fermi), with 448 CUDA cores running at 1.15 GHz and capable of 515 GigaFLOPs. Comparing the two, the GPU has 26.8 times the FLOP capability of the CPU per node. Our study used eight nodes for all tests.

5.3 Measurements

For each GPU test, we identified when "events" were occurring, for how long, and on which task. The events were:

- **"High-activity" advection** and **"low-activity" advection**. These events capture when the GPU is asked to do advection with more than 1,000 particles (high-activity) and less than 1,000 particles (low-activity). We experimented with additional gradations in activity level (i.e., differentiating further than low and high), but found that using only two categories simplified our analysis and the presentation of results.
- **CPU overhead**. Preparing buffers to send to the GPU and interpreting their results.
- **Latency**. Sending data between the CPU and GPU.
- **Idle**. The designation when the task is doing no work, often because it is waiting for work from other tasks.

© The Eurographics Association 2013.

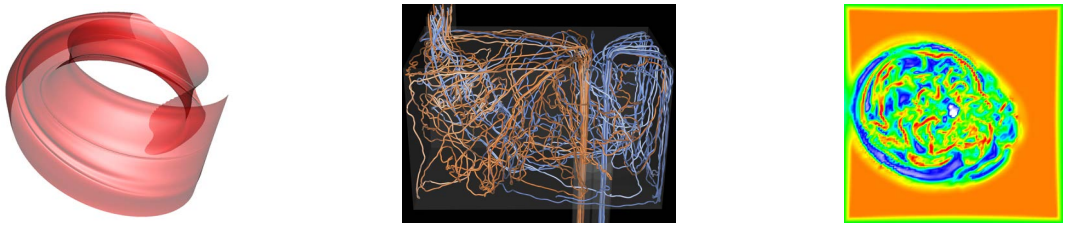

Figure 1: *On the left, a stream surface from the fusion data set, visualizing the magnetic field in a tokamak. In the middle, streamlines showing the mixing of air between twin inlets in a thermal hydraulics simulation. On the right, the FTLE of a solar core collapse resulting in a supernova. The left and middle images are reprinted with permission from [CCG*12] and [CGC*11], respectively.*

6 Results and Analysis

6.1 Comparisons of CPU and GPU Tests

Our test configurations – varying over data set, number of particles, and duration of advection – created 150 CPU-GPU comparisons. The test with the maximum speedup for the GPU implementation was 10.5X faster, coming from the fusion simulation and using the maximum number of particles and longest duration. The test with the worst speedup for the GPU implementation was 18X faster on the CPU, again coming from the fusion simulation and using the longest duration, but this time with one particle per data block. Table 1 compares speedups between the implementations with respect to the factors varied for our tests. The primary findings from this table are:

- The majority of GPU tests outperform their CPU counterparts.
- The CPU is faster most often when there are few particles, while the GPU is always a clear winner when the number of particles per data block exceeds 10,000.
- The GPU is always superior for short durations. This was a surprising result, which suggests that CPU-GPU transfer times do not play a large role in overall performance.
- Long durations provide the most counter-intuitive result, as they provide the examples with the most extreme benefit both for the CPU and for the GPU. The CPU is faster when there are few particles, while the GPU is faster when there are many.
- The results are consistent across all three data sets.

Additionally, Figure 2 gives further insights about speedup as a function of overall test time and computational workload.

Test Factors		>5x CPU	2X-5X CPU	2X CPU-2X GPU	2X-5X GPU	>5X GPU
Data Set	Astro	3	2	6	12	27
	T.H.	4	2	7	10	27
	Fus.	3	3	6	9	29
# of Particles	1	6	2	7	0	0
	5^3	4	5	6	0	0
	15^3	0	0	6	9	0
	25^3	0	0	0	10	5
	40^3	0	0	0	2	13
	50^3	0	0	0	3	12
	65^3	0	0	0	2	13
	80^3	0	0	0	3	12
	100^3	0	0	0	2	13
	250^3	0	0	0	0	15
Duration	Tiny	0	0	8	16	6
	Little	0	0	6	6	18
	Short	0	5	2	4	19
	Med.	4	2	1	4	19
	Long	6	0	2	1	21
	Total	10	7	19	31	83

Table 1: *This table examines each of the three testing factors: data set, number of particles, and duration. For each table entry, we count how many CPU-GPU tests have that performance ratio (column) for the given factor variant (row). For example, the table entry for the row for medium duration and the column for similar CPU and GPU times (i.e., between 2X CPU and 2X GPU) is* 1*. This is because the only test that matched this criterion was the one with the thermal hydraulics data set and 125 particles per data block, which had a GPU speedup of 1.58X, since its GPU time was 6.36s and its CPU time was 10.1s.*

6.2 Underlying Performance Drivers

Consider Figure 3, which plots GPU speedup as a function of the total number of advection steps. The figure shows that the amount of advection work can be misleading when predicting the amount of GPU speedup. In particular, the red line spotlights eight tests that all have approximately one million advection steps, but have different performance characteristics. One grouping has the CPU going approximately five times faster than its GPU counterpart, while the other has the GPU approximately five times faster.

We incorporated the events described in §5.3 to better understand the underlying factors behind performance. Since tests where the CPU is faster have qualitatively different performance characteristics than those where the GPU is faster, we split our analysis into two groups: CPU-advantage and GPU-advantage. For each event and advantage group, we plotted the proportion of the time spent performing that event as a function of speedup. We then calculated a best fit

© The Eurographics Association 2013.

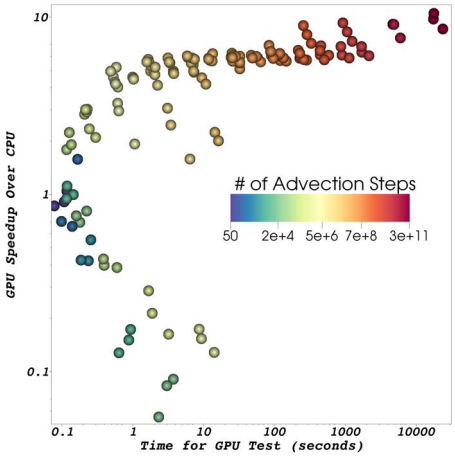

Figure 2: *Each glyph in the plot represents one of the 150 pairs of tests. The independent axis shows the time for a pair's GPU test, while the dependent axis shows the speedup of the GPU test over the CPU test. The coloring is of the total number of advection steps, using a logarithmic scale. By way of example, a test that required 70 million advection steps and took 10 seconds for its GPU test and 2.5 seconds for its CPU test would be placed at (10, 4) and colored orange. Since some CPU tests were faster than their GPU counterparts, the speedup axis goes below 1X.*

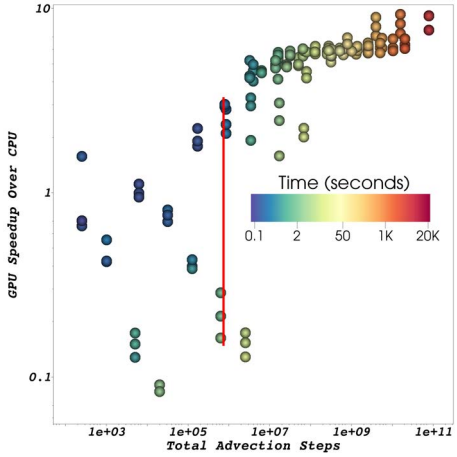

Figure 3: *Each glyph in the plot represents one of the 150 pairs of tests. The independent axis shows the total number of advection steps taken, while the dependent axis shows the speedup of the GPU test over the CPU test. The coloring is of elapsed time for the GPU test, using a logarithmic scale. The red line spotlights one of several groupings of tests where the raw computational workload is the same, but the speedup results are drastically different. See §6.2 for more discussion.*

line and recorded its slope and (Pearson) correlation coefficient. The results are listed in Table 2 and led to the following insights:

- The single highest slope and largest correlating factor for good performance on the GPU was with high activity advection. The plot showing this correlation is shown in Figure 4.
- Of the tests where the CPU had the advantage, there was not a single instance of high activity advection. Based on our tests, if an advection workload *ever* creates the conditions for high activity advection, the GPU is the superior option.
- However, low activity advection correlates with improved CPU performance in both groupings. Low activity advection corresponds to only some of the GPU's processor cores being used. This is often not a good approach, since individual processor cores on the CPU are faster.
- Latency correlated with reduced performance, but the slope was small enough to conclude that this factor is negligible. CPU overhead had no impact on performance.
- Idle time was negatively correlated for both CPU- and GPU-advantage groupings. This is because idle time gates both implementations; its presence prevents the CPU from being faster than the GPU and vice-versa.

6.3 GPU Speedups

The GPUs on Dirac have 26.9X more FLOPs than its CPU, well exceeding the maximum speedup we observed

	Event	Slope	Correlation Coefficient
GPU Advantage	High activity	11.6	0.81
	Low activity	-5.3	-0.63
	CPU overhead	-0.05	0.01
	Latency	-0.25	-0.44
	Idle	-6.0	-0.50
CPU Advantage	High activity	0.0	0.0
	Low activity	3.6	0.5
	CPU overhead	-0.03	-0.333
	Latency	-0.13	-0.69
	Idle	-3.4	-0.50

Table 2: *Correlating speedup with the proportion of time spent performing an event.*

with our tests (10.5X). We repeated our tests in a serial setting and again observed speedups on the order of 10X. We were only able to achieve a speedup commensurate to the FLOP ratio when we greatly reduced the size of the vector field data, simplifying memory accesses. We concluded that the unstructured memory accesses required for particle advection affect the GPU more than the CPU, which is an expected insight.

6.4 Performance Over Time

We show performance over time of three exemplars from our GPU tests in Figure 5:

- The top plot shows an example of high activity advection over the entire test. It is from the test with 15.6M parti-

© The Eurographics Association 2013.

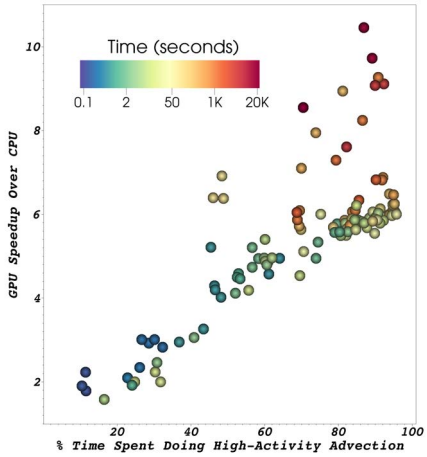

Figure 4: *Each glyph in the plot represents one of the 123 pairs of tests where the GPU was faster. The independent axis shows the percentage of time spent doing high activity advection, while the dependent axis shows the speedup of the GPU test over the CPU test. The coloring is of elapsed time for the GPU test, using a logarithmic scale. This plot shows the highest correlation between performance and any of the events we studied, with a correlation coefficient of 0.81. More information can be found in §6.2 and Table 2.*

cles per data block, advecting for 20,000 steps, and using the astrophysics data set. The GPU test took 17,205 seconds, while the CPU test took 167,226 seconds, making the GPU implementation 9.7X faster.

- The middle plot shows an example of high activity advection eroding into low activity advection over time, with the latter stages representing the workload from particles that cross domain boundaries. It is from the test with 3,375 particles per data block, advecting for 5,000 steps, and using the astrophysics data set. The GPU test took 0.38 seconds, while the CPU test took 1.17 seconds, making the GPU implementation 3.1x faster.
- The bottom plot shows an example of low activity advection over the entire test. It is from the test with 625 particles per data block, advecting for 1,000 steps, and using the astrophysics data set. The GPU test took 0.049 seconds, while the CPU test took 0.02 seconds, making the CPU implementation 2.5x faster.

7 Conclusion and Future Work

We presented an algorithm for GPU acceleration of particle advection and a study designed to illuminate when the GPU was beneficial in carrying out particle advection problems, and why. Our initial hypothesis was that latency in accessing the GPU would be a dominant factor in performance and that this study would explore the tension between the increased computational power of the GPU and the cost in accessing it. To our surprise, latency was virtually not a factor in the workloads we studied, and the findings showed

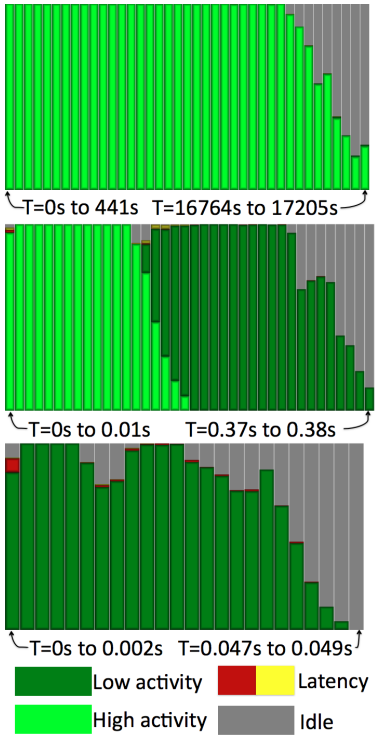

Figure 5: *Behavior over time from the three exemplar GPU tests described in §6.4. Each column represents a time interval and is colored according to events that occurred. The proportion devoted to a given color within a column reflects the proportion of time – aggregated over all eight GPUs – spent performing that event.*

that heavy advection work was the single greatest predictor of GPU benefit. A secondary finding was that idleness slowed both CPU and GPU implementations down, meaning that neither will be markedly superior to the other if there is significant idle time.

Finally, this study suggests many interesting future research directions, such as investigating higher concurrency levels, performance characteristics as a function of concurrency, truly heterogeneous algorithms that use both the CPU and the GPU, and comparisons with emerging architectures, such as the Intel Xeon Phi. The workload considered could also be expanded to consider unsteady flow, adaptive placement of particles, and parallelization schemes other than parallelization-over-data.

8 Acknowledgments

This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center

© The Eurographics Association 2013.

(NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[ADM12] ANCEL A., DISCHLER J.-M., MONGENET C.: Load-balanced multi-gpu ambient occlusion for direct volume rendering. In *EGPGV* (2012), Childs H., Kuhlen T., Marton F., (Eds.), Eurographics Association, pp. 99–108. 2

[AFM*12] AMENT M., FREY S., MÜLLER C., GROTTEL S., ERTL T., WEISKOPF D.: GPU-Accelerated Visualization. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. Oct. 2012, pp. 223–260. 2

[AGM*12] AYACHIT U., GEVECI B., MORELAND K., PATCHETT J., AHRENS J.: The ParaView Visualization Application. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. Oct. 2012, pp. 383–400. 1

[BRE05] BHANIRAMKA P., ROBERT P. C. D., EILEMANN S.: Opengl multipipe sdk: A toolkit for scalable parallel rendering. In *IEEE Visualization* (2005), IEEE Computer Society, p. 16. 2

[BSWE06] BACHTHALER S., STRENGERT M., WEISKOPF D., ERTL T.: Parallel texture-based vector field visualization on curved surfaces using gpu cluster computers. In *EGPGV* (2006), Heirich A., Raffin B., dos Santos L. P. P., (Eds.), Eurographics Association, pp. 75–82. 2

[BVS*11] BETHEL E. W., VAN ROSENDALE J., SOUTHARD D., GAITHER K. P., CHILDS H., BRUGGER E., AHERN S.: Visualization at Supercomputing Centers: The Tale of Little Big Iron and the Three Skinny Guys. *IEEE Computer Graphics and Applications (CG&A) 31*, 1 (Jan./Feb. 2011), 90–95. 1

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., PUGMIRE D., BIAGAS K., MILLER M., HARRISON C., WEBER G. H., KRISHNAN H., FOGAL T., SANDERSON A., GARTH C., BETHEL E. W., CAMP D., RÜBEL O., DURANT M., FAVRE J. M., NAVRÁTIL P.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. Oct. 2012, pp. 357–372. 1

[CCG*12] CAMP D., CHILDS H., GARTH C., PUGMIRE D., JOY K. I.: Parallel Stream Surface Computation for Large Data Sets. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Oct. 2012), pp. 39–47. 5

[CGC*11] CAMP D., GARTH C., CHILDS H., PUGMIRE D., JOY K. I.: Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 17* (Nov. 2011), 1702–1713. 2, 5

[CRE*05] CARDALL C. Y., RAZOUMOV A. O., ENDEVE E., LENTZ E. J., MEZZACAPPA A.: Toward Five-Dimensional Core-Collapse Supernova Simulations. *Journal of Physics: Conference Series 16* (2005), 390–394. 4

[ECBM08] ENDEVE E., CARDALL C. Y., BUDIARDJA R. D., MEZZACAPPA A.: Generation of Strong Magnetic Fields in Axisymmetry by the Stationary Accretion Shock Instability. *ArXiv e-prints* (Nov. 2008). 4

[FCS*10] FOGAL T., CHILDS H., SHANKAR S., KRÜGER J., BERGERON R. D., HATCHER P.: Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *Proceedings of High Performance Graphics (HPG)* (June 2010), pp. 57–66. 2

[FK10] FOGAL T., KRÜGER J.: Tuvok, an Architecture for Large Scale Volume Rendering. In *Proceedings of the 15th International Workshop on Vision, Modeling, and Visualization* (November 2010). 2

[FLPS08] FISCHER P., LOTTES J., POINTER D., SIEGEL A.: Petascale Algorithms for Reactor Hydrodynamics. *Journal of Physics: Conference Series 125* (2008), 1–5. 4

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., AHERN S., KIRCHNER P. D., KLOSOWSKI J. T.: Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 693–702. 2

[MLP*09] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over Two Decades of Integration-Based, Geometric Flow Visualization. In *EG 2009 - State of the Art Reports* (April 2009), Pauly M., Greiner G., (Eds.), Eurographics Association, pp. 73–92. 2

[MMD08] MARCHESIN S., MONGENET C., DISCHLER J.-M.: Multi-gpu sort last volume visualization. In *EG Symposium on Parallel Graphics and Visualization (EGPGV'08), Eurographics* (April 2008). 2

[MSE06] MÜLLER C., STRENGERT M., ERTL T.: Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV06)* (2006), Eurographics Association, pp. 59–66. 2

[MSM10] MARTIN S., SHEN H.-W., MCCORMICK P.: Load-balanced isosurfacing on multi-gpu clusters. In *EGPGV '10: Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2010* (May 2010), pp. 91–100. 2

[NLS11] NOUANESENGSY B., LEE T.-Y., SHEN H.-W.: Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 1785–1794. 2

[OPF*12] ÖZGÖKMEN T. M., POJE A. C., FISCHER P. F., CHILDS H., KRISHNAN H., GARTH C., HAZA A. C., RYAN E.: On multi-scale dispersion under the influence of surface mixed layer instabilities. *Ocean Modelling 56* (Oct. 2012), 16–30. 2

[PCG*09] PUGMIRE D., CHILDS H., GARTH C., AHERN S., WEBER G. H.: Scalable Computation of Streamlines on Very Large Datasets. In *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC09)* (Nov. 2009), pp. 1–12. 2

[PPG12] PUGMIRE D., PETERKA T., GARTH C.: Parallel Integral Curves. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. Oct. 2012, pp. 91–113. 2

[PRN*11] PETERKA T., ROSS R., NOUANESENGSEY B., LEE T.-Y., SHEN H.-W., KENDALL W., HUANG J.: A study of parallel particle tracing for steady-state and time-varying flow fields. In *Proceedings of IPDPS 11* (Anchorage AK, 2011). 2

[SGG*04] SOVINEC C., GLASSER A., GIANAKON T., BARNES D., NEBEL R., KRUGER S., PLIMPTON S., TARDITI A., CHU M., THE NIMROD TEAM: Nonlinear Magnetohydrodynamics with High-order Finite Elements. *J. Comp. Phys. 195* (2004), 355. 4

[SML96] SCHROEDER W. J., MARTIN K. M., LORENSEN W. E.: The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *VIS '96: Proceedings of the 7th conference on Visualization '96* (1996), IEEE Computer Society Press, pp. 93–ff. 1

[SMW*04] STRENGERT M., MAGALLÓN M., WEISKOPF D., GUTHE S., ERTL T.: Hierarchical visualization and compression of large volume datasets using gpu clusters. In *In Eurographics Symposium on Parallel Graphics and Visualization (EGPGV04) (2004* (2004), pp. 41–48. 2

© The Eurographics Association 2013.