Particle Advection Performance Over Varied Architectures and Workloads

Hank Childs*†, Scott Biersdorff*, David Poliakoff*, David Camp† and Allen D. Malony*
* *University of Oregon*
† *Lawrence Berkeley National Laboratory*

Abstract—**Particle advection is a foundational operation for many flow visualization techniques, including streamlines, Finite-Time Lyapunov Exponents (FTLE) calculation, and stream surfaces. The workload for particle advection problems varies greatly, including significant variation in computational requirements. With this study, we consider the performance impacts from hardware architecture on this problem, studying distributed-memory systems with CPUs with varying amounts of cores per node, and with nodes with one to three GPUs. Our goal was to explore which architectures were best suited to which workloads, and why. While the results of this study will help inform visualization scientists which architectures they should use when solving certain flow visualization problems, it is also informative for the larger HPC community, since many simulation codes will soon incorporate visualization via *in situ* techniques.**

Keywords-**GPGPU, Hybrid Parallelism, Flow Visualization, Performance Analysis**

I. Introduction

The hardware architectures on nodes of supercomputers are becoming increasingly varied. Some supercomputers have nodes with relatively modest computational capabilities, for example nodes that contain only four cores. Other supercomputers have individual nodes that have very high computational capabilities that could be considered supercomputers themselves, for example nodes that contain multiple accelerators (e.g., multiple GPUs). And many supercomputers have nodes that lie between these extremes, with dozens of cores per node, or the presence of just a single accelerator.

With this study, we consider diverse hardware architectures in the context of "particle advection." Particle advection – displacing particles so that they are tangent to the velocity field – is a foundational element for 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 data dependent and thus not known a priori. Further, the workload across particle advection problems can change dramatically. Streamline calculation typically involves advecting few particles for long distances, while FTLE calculation typically involves advecting many particles for short distances. In turn, studies considering this problem should examine a range of scenarios, varying over particle count, distance traveled, and vector field. Finally, visualization and analysis is increasingly being performed in an *in situ* setting [1], where visualization and analysis is performed at the same time as the simulation, and using some of its resources. This usage modality increases the need for understanding particle advection over many architectures.

This study is an extension of our previous work [2], which compared the performance of particle advection problems between CPU and GPU clusters. In this work, we aim to better understand performance over a spectrum of computation capabilities, and we do this by expanding the hardware configurations considered from two (one CPU configuration and one GPU configuration) to eleven. The result allows us to explore our fundamental research question: **What are the relationships between execution time and architecture for particle advection problems?** An important contribution of this paper is exploring this relationship from the perspective of high performance computing systems, specifically evaluating the usefulness of the compute capabilities provided on a supercomputer node for a complex data-intensive problem.

Another important contribution of this paper is the insights it provides for visualization scientists who are studying flow visualization problems. These scientists are actually faced with two related problems. First, when running *in situ*, what techniques can they employ that will fit within the constraints of the overall simulation? In this case, the hardware configuration is set, and the visualization scientist must choose an appropriate particle advection workload. Second, when running a stand-alone visualization program, what hardware should a visualization scientist choose to solve a given particle advection problem most quickly? In this case, the particle advection workload is set, and the visualization scientist must choose an appropriate architecture. We note that although this latter case may seem unusual, it actually occurs quite often; many supercomputing centers have multiple supercomputers connected to the same disk, and the visualization scientist has the flexibility to choose which supercomputer the visualization program runs on.

II. Related Work

A. Flow Visualization and Particle Advection

McLouglin et al. recently surveyed the state of the art in flow visualization [3], and the large majority of techniques they described incorporate 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 which determine the rate of separation through the volume, advect a particle for every node in a mesh and compare the divergence of nearby particles, determining the rate of separation throughout the volume.

B. *Using GPU Clusters for Visualization*

Many visualization algorithms have been ported to and optimized for the GPU [4]. While less work is devoted to parallel GPU clusters, there has been significant research in achieving load balancing and scalability for rendering, both for surfaces [5], [6], [7] and for volumes [8], [9], [10], [11]. Very few studies with parallel GPU clusters are devoted to the transformations that precede rendering, with notable exceptions on isosurfacing [12] and on line integral convolution (LIC) [13].

C. *Parallelizing Particle Advection for Visualization*

A summary of strategies for parallelizing particle advection problems on CPU clusters can be found in [14]. The basic approaches are to parallelize-over-data, parallelize-over-particles, or a hybrid of the two [15]. Recent results using parallelization-over-data demonstrated streamline computation on up to 32,768 processors and eight billion cells [16]. Alternate approaches include using preprocessing to study the patterns of the flow and then to schedule processing of blocks to optimize performance [17], and include using work-requesting to dynamically balance load [18].

D. *Effects of Hardware Architecture on Particle Advection Performance*

This topic is most directly aligned with this study. Multiple studies ([19], [20], [21], [22]) have focused on GPU implementations of particle advection problems for desktop machines with a single GPU. In all cases, the particle advection workloads considered required significant computational resources, and the GPU was found to be superior when compared to a CPU.

To our knowledge, our two previous studies are the only ones to have looked at hardware architecture effects on particle advection in distributed-memory systems. In our first study [23], we looked at streamlines on multi-core CPUs and showed the benefits of hybrid parallel techniques. In this study, we compared workloads that used only distributed-memory parallelism with those that used both shared- and distributed-memory. We observed that, when using the same hardware in both configurations, the hybrid parallel version regularly outperformed the distributed-memory version, sometimes by factors of more than 10X.

Our second study [2] — extended by the present study — looked at architectural effects on particle advection, and compared CPU and GPU clusters. The study again compared a variety of workloads, and found that GPU clusters often, but not always, outperform their CPU cluster counterparts. The study described in this paper represents a significant step forward in our understanding of the problem. The previous study, which considered just two data points — CPU clusters with eight cores per node and GPU clusters with one GPU per node — was too coarse, while this study, which considers nodes over a spectrum of computational ability, is better able to answer the question of which particle advection problems are best fitted for which computational environments.

III. Problem and Algorithm Overview

In this section, we present an overview of the particle advection problem (§III-A), as well as an overview of the parallel algorithm we used for our study (§III-B).

A. *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 a particle travels along, and it is formed by the sequence of advection steps from the seed location to the terminal location. The integral curve is defined by an ordinary differential equation, as its derivative at a given position is defined as the value of the simulation's velocity field at the same position. As a result, advecting particles is a data dependent process, and the calculation of advection steps must be carried out sequentially. Explicitly, the N^{th} advection step for a particle can only be calculated after the location of $(N-1)^{st}$ advection step's displacement is known.

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.

Further, 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 where the simulation data is pre-divided into pieces. Further, for the case of GPU-based supercomputers, the data is likely already located on the GPU.

B. Algorithm Overview

Our study uses the algorithm introduced in [2], and here we describe only the key elements of the algorithm that relate to this study. The algorithm has two phases: initialization and advection. The spirit of the implementations for both phases are the same for CPU and GPU clusters, but the details of the implementation differ, especially for advection.

1) Initialization Phase: The algorithm's initialization phase consists of three parts: (i) loading data, (ii) constructing a **piece map** of where data resides, and (iii) particle creation and initialization. For (i), each task reads its piece directly from disk. For the GPU implementation, the data is transferred to GPU memory as a texture map, along with other meta-data. For (ii), each task creates a map between the tasks and the spatial extents of their pieces. 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.

2) Advection Phase: The processing is driven by three queues, which each contain particles. We designate three different particle states, and each queue contains particles of a specific state. The **active queue** contains particles that need to be advected. The **finished queue** contains particles 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 — these particles must be sent to another node that has the piece of the vector field the particle will enter.

The goal of the advection phase is to promote all the 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 initial active queue. That said, finished tasks continue participating in the algorithm, since individual tasks that are finished may contain portions of the vector field 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. 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 sent back to that task. 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.

For details beyond this level, especially in terms of CPU and GPU implementation, we refer the reader to our previous work [2].

IV. Study Overview

A. Test Configurations

Our study was designed to understand how particle advection workloads varied over diverse hardware architectures. We varied four factors:

- Node architecture (11 options: 6 CPU, 5 GPU)
- Data set (3 options)
- Particle density (9 options)
- Duration of advection (5 options)

We ran the cross-product, meaning $11 \times 3 \times 9 \times 5 = 1,485$ tests overall. The variants for each factor are discussed in the remainder of this subsection.

1) Node Architecture: We ran with the following configurations:

- **GPU_8x1:** Eight nodes, with each node utilizing one NVIDIA M2090 GPU.
- **K20_GPU_8x1:** Eight nodes, with each node utilizing one NVIDIA K20 GPU.
- **GPU_4x2:** Four nodes, with each node utilizing two NVIDIA M2090 GPUs.
- **GPU_3/3/2:** Three nodes, with the nodes utilizing three, three, and two NVIDIA M2090 GPUs, for a total of eight GPUs utilized.
- **GPU_8x3:** Eight nodes, with each node utilizing three NVIDIA M2090 GPUs, for a total of twenty-four GPUs.
- **CPU_8x2:** Eight nodes, with each node utilizing two threads from a multi-core CPU.
- **CPU_8x4:** Eight nodes, with each node utilizing four threads from a multi-core CPU.
- **CPU_8x8:** Eight nodes, with each node utilizing eight threads from a multi-core CPU.
- **CPU_8x12:** Eight nodes, with each node utilizing twelve threads from a multi-core CPU.
- **CPU_8x16:** Eight nodes, with each node utilizing sixteen threads from a multi-core CPU.
- **CPU_8x24:** Eight nodes, with each node utilizing twenty-four threads from a multi-core CPU.

Again, there were five GPU tests and six CPU tests. Four of the GPU tests used the NVIDIA M2090 GPU, and one used the NVIDIA K20 GPU. Finally, the majority of our tests use eight nodes and have one MPI task per node, with the only exceptions being GPU_3/3/2 and GPU_4x2 (which had less than eight nodes), and GPU_8x3 (which had twenty-four MPI tasks).

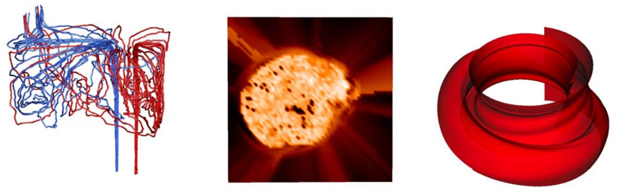
Figure 1. On the left, streamlines showing the mixing of air between twin inlets in a thermal hydraulics simulation. In the middle, the FTLE of a solar core collapse resulting in a supernova. On the right, a stream surface from the fusion data set, visualizing the magnetic field in a tokamak.

2) Data Sets: The underlying vector field can greatly influence performance characteristics, as sinks in the vector field can attract particles from far away and create load imbalance. For this reason, we considered three data sets to ensure the diversity of our tests. Each data set was a single time slice, meaning we studied steady state flow. Each data set had a resolution of $1,000^3$. Ten of our eleven node architectures had eight MPI tasks; for this case, each task operated on a data block of 500 x 500 x 500 cells. The GPU_8x3 test had twenty-four MPI tasks (three per node), and the data was divided into smaller pieces for this test.

Figure 1 shows different particle advection-based visualizations on the three data sets.

Thermal Hydraulics: In this simulation, two inlets pump air into a box, which circulates and exits through an outlet. The simulation was performed using the NEK5000 code [24].

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 the GENASIS simulation code [25].

Fusion: This data set comes from a simulation of magnetically confined fusion in a tokamak device by the NIMROD simulation code [26]. 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 is representative of data sets that have high circulation — particles traverse the torus-shaped vector field domain repeatedly.

3) Particle Density: We had nine particle density configurations, which determine the number of seeds placed into each data block. 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, and 100^3. These workloads are representative of use cases such as streamlines, stream surfaces, and coarser FTLE analysis, among others. Over all tasks, the lowest number of particles was just eight, while the highest number was eight million.

Finally, note that the GPU_8x3 configuration had a different number of blocks. For that case, the number of particles per block were adjusted so that the total number of particles matched the other tests, enabling comparisons.

4) Duration of Advection: The duration of the advection (i.e., the number of advection steps) corresponds to the number of advection steps taken. 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).

B. Runtime Environment

We present test results from Georgia Tech's Keeneland supercomputer, Oak Ridge National Laboratory's Titan supercomputer, and Lawrence Berkeley's NERSC machine Edison.

1) Keeneland: Keeneland was used for four of the five GPU tests: GPU_8x1, GPU_4x2, GPU_3/3/2, and GPU_8x3. A single compute node of Keeneland contains two 8-core 2.8GHz Intel Sandy Bridge (Xeon E5) processors and 32GB of RAM. It is accelerated by three NVIDIA M2090 GPUs with 5.6GB of RAM. Nodes are connected via a Mellanox FDR InfiniBand interconnect.

2) Titan: Titan was used for the K20_GPU_8x1 test. A single compute node of Titan contains a 16-core 2.2GHz AMD Opteron 6274 (Interlagos) processor and 32GB of RAM. It is accelerated by an NVIDIA Kepler GPU with 6GB of DDR5 memory. Nodes within the compute partition are connected by a three-dimensional torus.

3) Edison: Edison was used for all CPU tests. A single compute node of Edison contains two sockets and each socket has a 12-core 2.4 GHz Intel "Ivy Bridge" processor and 64GB of RAM. Nodes are connected via a Cray Aries with the Dragonfly topology.

C. Measurements

An important component of our research objective to expand investigation across more diverse hardware configurations was to apply a parallel performance measurement and analysis system that is portable on leading platforms. TAU Performance System® [27] provided this support with its broad set of portable instrumentation and measurement techniques (including for heterogeneous machines), its parallel performance data management infrastructure, its parallel performance data mining framework, and its integration with other performance technology (e.g., PAPI). All of these capabilities proved to be valuable in the multi-experiment, cross-architecture analysis we performed. Further, we made heavy use of the scripting features of TAU's PerfExplorer (powered by its relational database: TAUdb) to construct analysis pipelines that generated results specific to understanding particle advection performance.

The measurement approach also captured the key events (i.e., "idle", "advecting") identified from our previous study, and we augmented them with additional observations of MPI communication, multi-threading, and GPU operations.

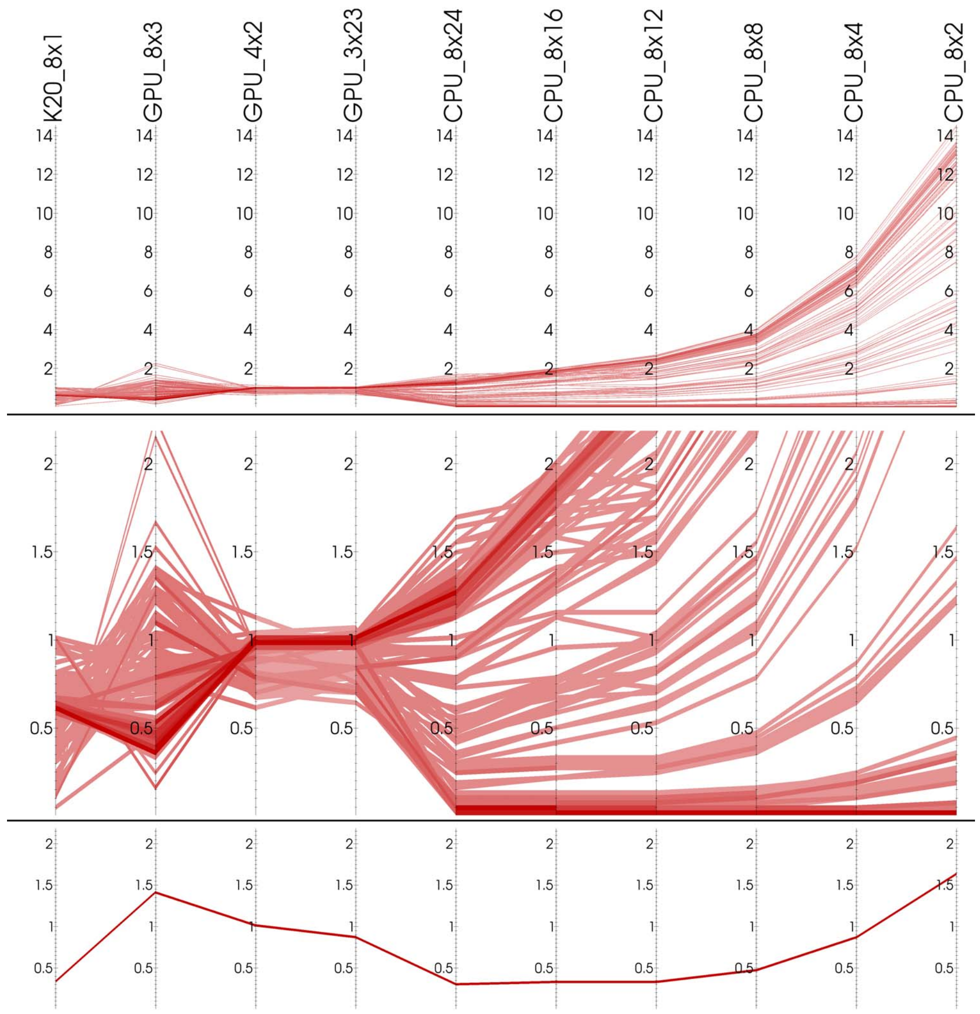

Figure 2. Parallel coordinates plot of execution time, by hardware configuration. Each hardware configuration (see §IV-A1) is an axis for the plot, starting with K20_GPU_8x1 on the left, and going to CPU_8x2 test on the right. The Y-axis is speedup relative to the time for the GPU_8x1 test (which is not displayed, since it is a constant 1). If a CPU_8x2 test time is twice as fast than the GPU_8x1 test, then it will be plotted with a Y-value of 0.5. If it is twice as slow, then it will be plotted with a Y-value of 2.0. The parallel coordinates are plotted based on density; regions where there are many lines are plotted dark red, while regions with few lines are plotted light red. The top figure shows all of the data, while the middle figure zooms in on the region where the performance is no more than 2X worse than GPU_8x1. Where the top two figures show results for all 135 workloads, the bottom figure shows the results for a single workload — the Fusion data set, with 25^3 particles per data block, and advecting for the *tiny* duration (50 steps) — and thus results in a single line. This line shows that the K20_GPU_8x1 test is about 2.5X faster ($1/0.4$) than GPU_8x1, while the CPU_8x2 test is about 1.6X slower than GPU_8x1. This bottom figure is included to better illustrate how to interpret parallel coordinate plots. Analysis of these results are discussed in §V.

All measurements were restricted to solving the algorithm presented in this paper; generation of data and transferring vector field data were not measured, since the study is aimed at *in situ* use cases and this data would already be in place in such a setting.

V. Results

We analyzed the results of the $1,485$ tests in multiple ways. Our analysis directions were driven by a global view, realized as a parallel coordinates plot that showed all test data. This graphic, shown in Figure 2, plots the speedups of each test with respect to the GPU_8x1 architecture. For the most part, we used this global view to focus on subsets of the hardware architectures where comparisons were informative with respect to understanding the impacts of hardware architecture on performance. Specifically, we explored comparisons with:

- CPU tests with varying numbers of cores, in §V-A;
- The two NVIDIA cards, M2090 and K20 (i.e., GPU_8x1 and K20_GPU_8x1), in §V-B1;
- A GPU configuration (K20_GPU_8x1) and the CPU tests, in §V-B2;
- Our base GPU configuration and the configurations where eight GPUs are packed onto fewer than eight nodes (GPU_4x2 and GPU_3/3/2), in §V-C1; and

- Our base GPU configuration and the configuration that used three GPUs per node (GPU_8x3), in §V-C2.

A. CPU Performance as a Function of Cores Per Node

With these comparisons, we wanted to better understand when additional cores will help with overall execution time. Figure 3 shows a parallel coordinates plot of efficiency as we add more cores. We identified that there are three distinct categories of workloads:

- **Group 1:** this group has 15^3 or more particles per data block and advects for 1000 steps or more. The number of members in Group 1 is 63. Members of this group exhibit outstanding performance increases when more cores are added.
- **Group 2:** this group has one particle per data block. The number of members in Group 2 is 15. Members of this group exhibit no performance increase when more cores are added.
- **Group 3:** this group contains the remaining workloads, which number 57. Members of this group exhibit performance increases when more cores are added, but they are not proportional with the number of cores added. This suggests that the efficiency is decreasing as the number of cores increases, due to less work per core.

Table I shows specifics for how many tests benefits from the addition of new cores, and how much benefit they derive. As expected, workloads with significant computational work benefits from more cores, while those with minimal work do not.

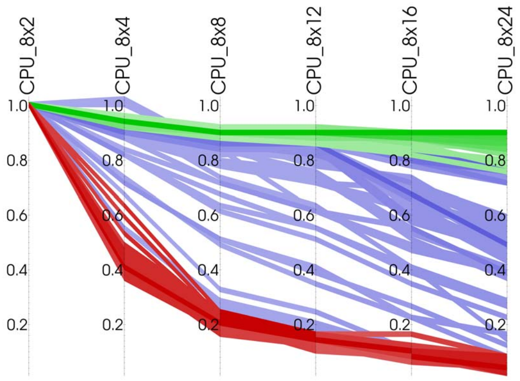

Figure 3. This parallel coordinates plot shows the efficiency from adding new cores over all the CPU tests. Each particle advection workload (i.e., a selected density, duration, and vector field) is a line on the parallel coordinates plot, illustrating workload performance as a function of the number of cores. The coloring comes from the groups defined in §V-A: Group 1 is green, Group 2 is red, and Group 3 is blue. Regions where multiple lines overlap are drawn darker. The plot normalizes execution time by the number of cores. Ideally, if the CPU_8x2 test takes time T, then CPU_8x4 will take $T/2$, CPU_8x8 will take $T/4$, and so on. In the worst case, the extra cores will be unused. In that case, the execution time will continue to take time T. In this plot, a CPU_8x24 test that takes T (i.e., no speedup over CPU_8x2) will be plotted as $1/12$, since it has twelve times the resources, and ideally should finish in one twelfth of the time.

	# Cores	Group 1	Group 2	Group 3
CPU Scalability	4	94%	47%	89%
	8	90%	22%	80%
	12	90%	14%	76%
	16	90%	10%	62%
	24	88%	5%	51%

Table I
THIS TABLE SHOWS THE AVERAGE NORMALIZED EFFICIENCY WITH RESPECT TO THE NUMBER OF CORES ON THE CPU AND TO THE GROUPS DEFINED IN §V-A. FOR EXAMPLE, FOR 12 CORES AND GROUP 3, THE VALUE IS 76%. IDEALLY, TWELVE CORES SHOULD BE ABLE TO FINISH SIX TIMES FASTER THAN THE BASELINE OF TWO CORES. THE 76% VALUE MEANS THAT THE TESTS IN GROUP 3 ACHIEVED 76% OF THE HOPED FOR SPEEDUP FROM THE ADDITIONAL CORES, OR THAT IT WAS ABOUT $4.5X$ FASTER (AND NOT THE FULL $6X$ FASTER). A VALUE OF 100% WOULD INDICATE THAT THE POTENTIAL SPEEDUP WAS REALIZED. GROUP 2 DEMONSTRATED SPEEDUP CONSISTENT WITH NOT MAKING USE OF THE ADDITIONAL CORES, WHICH IS TO BE EXPECTED GIVEN ITS MODEST COMPUTATIONAL WORKLOAD.

B. Comparisons Across Machines

Several of our desired comparisons involved associated tests run on different machines. Unfortunately, it is difficult to make meaningful comparisons across machines, since many factors may be different, most notably the networking infrastructure. To facilitate these comparisons, we ran a series of MPI benchmarks on the three machines considered in this study. As seen in Table II, asynchronous message communication is fastest on Edison, second fastest on Titan, and slowest on Keeneland. We reference this relationship in the analysis we perform in §V-B1 and §V-B2.

Size	Keeneland	Titan	Speedup	Edison	Speedup
10K	7.39	1.93	3.8	0.61	12.0
50K	24.72	5.66	4.4	1.97	12.6
200K	42.15	20.75	2.0	7.95	5.3

Table II
RESULTS OF THE MPI BENCHMARK TESTS RUN ON THE THREE MACHINES CONSIDERED IN THIS STUDY. THE VALUES REPORTED FOR EACH MACHINE IS THE TIME (IN MICROSECONDS) TO SEND AN ASYNCHRONOUS MESSAGE OF A GIVEN SIZE. THE SPEEDUPS LISTED ARE NORMALIZED TO KEENELAND.

1) Comparing Different GPUs: With these comparisons, we wanted to better understand the extent that faster GPUs can improve overall execution time. Titan's K20x GPUs have about twice the raw computational power of the M2090s in Keeneland (1.31 GFLOPs vs 0.67 GFLOPs, double precision). Further, as noted in §V-B, Titan's network is significantly faster than Keeneland's. As we compared particle advection workloads, we wanted to understand where the speedups come from: GPU or network. Most tests are dominated either by network time or by advection time. The former demonstrates benefits tracking the network improvements, while the latter demonstrate benefits tracking the GPU improvements. Specifically, workloads with densities of 50^3 particles or more spend at least half of their time doing advection, and thus should have speedups on the order of $2X$, matching the ratio of the GPUs. Further, those workloads with fewer particles see even better speedups, due to Titan's superior network. We note that the speedups in

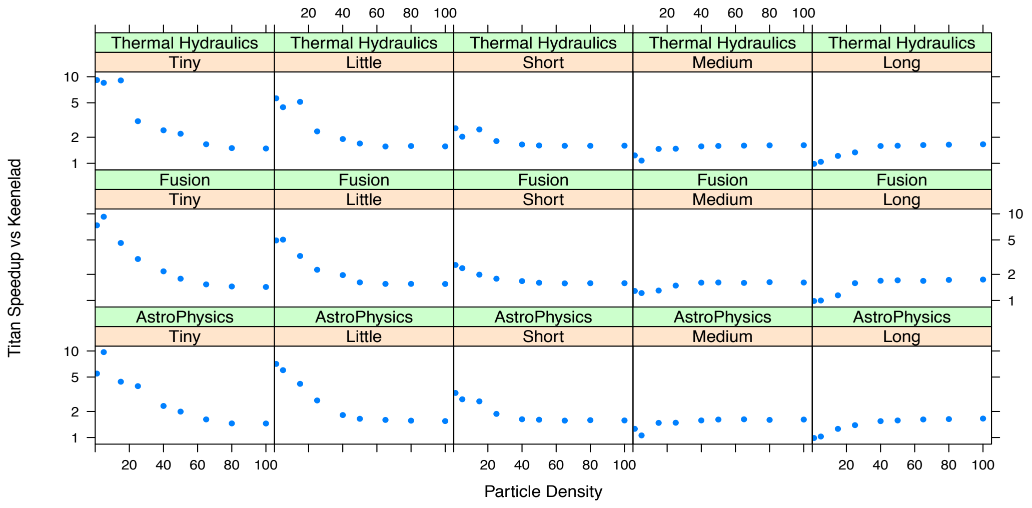

Figure 4. A 5x3 matrix of plots. The three rows correspond to the three vector fields (see §IV-A2). The five columns correspond to the five advection durations (see §IV-A4). Each plot shows the speedup of Titan over Keeneland as a function of particle density (§IV-A3).

some tests exceeded those of our MPI benchmarks, likely because they do not need to exchange many messages.

Figure 4 illustrates the above analysis. It shows that vector field is not playing a large role in determining the characteristics of the workload, and so the plots are similar from top to bottom. However, duration is very important to workload characteristics, and a consistent trend can be seen from left to right in the Figure. On the left, the durations are so small that the network speedups are dominant for low particle densities. On the right, the durations are so large that idle time makes the two machines nearly equivalent for low particle densities. But, as the particle density increases, the speedup approaches a fairly consistent value of approximately $2X$, since these tests make such heavy use of the GPU, and Titan's GPUs are twice as fast as Keeneland's. This trend is true over all durations and vector fields.

2) GPU Versus CPU: With these comparisons, we wanted to better understand how GPUs compared with CPUs. This comparison was particularly inspired by the observation from Figure 2 that the CPU_8x24 tests were never more than $1.7X$ slower than the GPU_8x1 tests. Since the network analysis in §V-B showed that Edison has the best network and Keeneland has the worst (and these were the machines compared in Figure 2), we switch our analysis here to be between the Edison CPU configurations and the Titan K20_GPU_8x1 configuration, since their network speeds are more comparable.

The results of our Titan-Edison comparisons can be seen in Figure 5. It shows that CPUs with many cores are competitive with GPUs over all tests. Further, it shows that CPUs are faster than GPUs with many tests with short execution times (i.e., < 0.2 seconds).

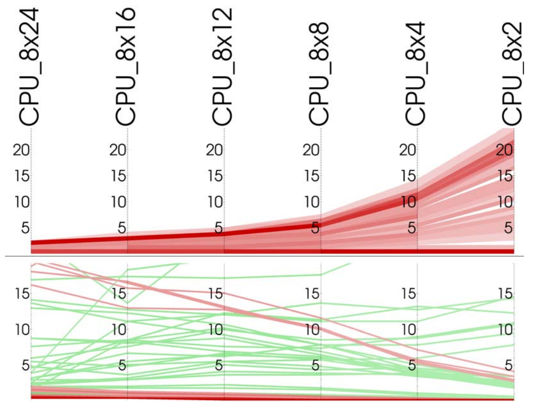

Figure 5. Comparing K20_GPU_8x1 and CPU configurations. The top image shows parallel coordinates of CPU test times normalized by K20_GPU_8x1 test times. The CPU_8x24 configuration was competitive, as it was only $2.5X$ slower than the GPU in the worst case. The GPU was much more dominant against fewer cores, though, with speedups of up to $25X$ versus CPU_8x2. The bottom image plots the K20_GPU_8x1 test time normalized by the CPU test times, showing where the CPU is faster than the GPU. Lines that are green correspond to the K20_GPU_8x1 tests that take less than 0.2 seconds. Lines that are red correspond to the K20_GPU_8x1 tests that take more than 0.2 seconds. Almost all tests which that take more than 0.2 seconds are ones where the GPU excels. However, one set of tests — a set of red lines going from upper left to lower right — takes more than 0.2 seconds and sees speedups of more than $15X$ for CPU_8x24. These tests correspond to 5^3 particles per data block and long durations, i.e., tests that will not overwhelm a CPU, but do not have enough work for a GPU.

C. Multiple GPUs Per Node

With these comparisons, we wanted to better understand the effects — positive or negative — of having multiple GPUs per node. The potential pitfall from having additional GPUs per node is that contention can arise on the PCI bus connecting the CPU and the GPU. Interestingly, Keeneland nodes have two PCI buses. This allows us to investigate the cases where there is no bus contention (a different PCI bus is used for each GPU in the GPU_4x2 case), as well as where there is bus contention (the PCI buses are shared in the GPU_3/3/2 and GPU_8x3 cases). Our experiments were constructed to see these hardware tradeoffs.

1) Eight GPU, Fewer Nodes Tests: With these comparisons, we fix the total number of GPUs at eight, and compare our base GPU configuration (GPU_8x1) with configurations that have fewer nodes (GPU_4x2 and GPU_3/3/2). While the contention pitfall remains for GPU_3/3/2, the potential benefit of these reduced node configurations is that there are fewer nodes participating in communication, which may lead to increased network performance.

For the most part, the tests had similar performance. Tests that ran for more than 0.2 seconds were in particularly good agreement, with the execution time for each workload being within 20% of the GPU_8x1 time in all cases. Tests that ran for less than 0.2 seconds were more likely to benefit from the fewer number of nodes, with approximately half of such tests being more than 20% faster than the GPU_8x1 time. See Table III for more details.

Node Type	Criteria	Total Tests	# Tests $\geq 80\%$ of GPU_8x1
GPU_3/3/2	$> 0.2s$	93	93
GPU_3/3/2	$< 0.2s$	42	22
GPU_4x2	$> 0.2s$	93	93
GPU_4x2	$< 0.2s$	42	17

Table III
ALL TESTS THAT TOOK LONGER THAN 0.2 SECONDS WERE WITHIN 80% OF THE TIME FOR THE GPU_8X1 TIME FOR THE SAME WORKLOAD. HOWEVER, WHEN THE TIME DROPPED BELOW 0.2 SECONDS, THE CONFIGURATIONS WITH REDUCED NUMBERS OF NODES WERE MORE THAN 20% FASTER ABOUT HALF THE TIME.

We also looked at the possible effects of contention. What we observed was small; no GPU_4x2 or GPU_3/3/2 was ever more than 6% slower than its counterpart GPU_8x1 test. However, we could observe that the number of slower tests for GPU_3/3/2 was more than for GPU_4x2. See Table IV for more details.

2) Eight Node, Multiple GPU Tests: With these comparisons, we fix the total number of nodes at eight, and compare our base configuration (GPU_8x1) with the configuration that has eight nodes and three GPUs per node (GPU_8x3). This configuration has a different benefit from that discussed in §V-C1: increased computational power. Further, while the contention pitfall still exists for this configuration, the observations from §V-C1 point to this factor being small. The challenge for this configuration is to actualize the computational power by providing the extra GPUs with enough work to do. This challenge already exists with a single GPU, and is only exacerbated by having three GPUs. Finally, we note that the GPU_8x3 configuration is different than the other configurations, since it has twenty-four processes executing, while all other configurations have eight processes. This also means that the workload is partitioned twenty-four ways (not eight ways), and so each of the twenty-four processes has less work to do, relatively speaking.

Configuration	# of Slower Tests	Avg. # of Particles
GPU_3/3/2	37	468K
GPU_4x2	17	261K
Expected For Identical	67.5	222K

Table IV
THE GPU_3/3/2 TESTS WERE SLOWER THAN GPU_8X1 37 TIMES, WHILE GPU_4X2 TESTS WERE SLOWER THAN GPU_8X1 ONLY 17 TIMES. OF COURSE, IF THE CHANGES IN CONFIGURATION LED TO NO TANGIBLE PERFORMANCE DIFFERENCES, THEN A GIVEN CONFIGURATION WOULD BE SLOWER HALF THE TIME, I.E., FOR 67.5 OF THE 135 TESTS. THE "EXPECTED FOR IDENTICAL" ROW IN THE TABLE CAPTURES THIS. OF THE 37 TESTS WHERE GPU_3/3/2 WAS SLOWER, WE LOOKED AT THE AVERAGE NUMBER OF PARTICLES ADVECTED (I.E., THE PARTICLE DENSITIES FROM §IV-A3). THIS AVERAGE WAS $468,000$ PARTICLES, COMPARED TO $261,000$ PARTICLES FOR THE GPU_4X2 AND A $222,000$ PARTICLE AVERAGE OVER ALL WORKLOADS. FROM THESE DATA POINTS, WE CONCLUDE THAT CONTENTION DOES AFFECT THE GPU_3/3/2 TESTS, AND THAT IT AFFECTS IT MOST WHEN THE PARTICLE DENSITY IS HIGH. WE ALSO COMMENT THAT THE REDUCED NUMBER OF NODES PROVIDES A COMMUNICATION BENEFIT TO OVERALL EXECUTION TIME, SO THE FULL EXTENT OF THE CONTENTION MAY BE SOMEWHAT GREATER THAN WHAT WE MEASURED.

Our tests found that the extra GPUs were helpful in cases where the particle density was high. Twenty-four of our workloads were able to achieve a speedup of $2.5X$ over their GPU_8x1 counterparts. All twenty-four of these tests come from the four highest densities ($50^3, 65^3, 80^3, 100^3$). Surprisingly, the best speedups came from high density workloads with small or medium durations, as seen in Table V. These workloads are in a sweet spot where there is enough work to saturate the extra GPUs, but they also finish quickly enough for load imbalance to not be an issue. This contrasts with tiny and little durations, which do not provide enough work, and with long durations, which end up having significant idle time while waiting for a handful of particles to finish executing.

Particle	Duration				
Density	Tiny	Little	Short	Medium	Long
50^3	0.76X	1.35X	2.02X	2.48X	2.41X
65^3	0.76X	1.54X	2.31X	2.69X	2.42X
80^3	0.94X	1.84X	2.52X	2.77X	2.48X
100^3	1.20X	2.22X	2.72X	2.85X	2.50X

Table V
THE AVERAGE SPEEDUP OF THE GPU_8X3 CONFIGURATION OVER GPU_8X1. EACH ENTRY IN THE TABLE IS THE AVERAGE OF THE EXPERIMENTS FOR THE GIVEN DURATION AND DENSITY. THE SPEEDUPS GET BETTER AS DENSITIES INCREASE, AND ARE AT THEIR HIGHEST FOR SHORT AND MEDIUM DURATIONS, WHICH ARE LESS PRONE TO LOAD IMBALANCE COMPARED TO THE LONG DURATION.

VI. Summary of Particle Advection-Hardware Findings

An important goal of this effort was to illuminate the best hardware for particle advection problems for visualization scientists. We summarize the findings from the previous sections:

On the value of additional cores on a node (§V-A):

- High density workloads with medium or longer duration will benefit from using more and more cores on a CPU node.
- Low density workloads will not benefit from adding more cores.
- The remaining workloads fall between these extremes.

On comparisons between GPUs (§V-B1):

- When particle advection densities become large (i.e., 50^3 or more), the GPUs became saturated, and so using faster GPUs led to speedups proportional to their FLOP rates.
- When particle advection densities are low, the network is the most important factor in performance.

On comparisons between CPU and GPU nodes (§V-B2):

- CPUs with lots of cores are competitive with GPUs.
- CPUs with few cores can be significantly slower than GPUs.
- CPUs often beat GPUs when a test's execution time is short.

On comparisons with a fixed number of GPUs, but a variable number of nodes (§V-C1):

- Runtimes do not vary considerably, and any configuration is likely acceptable.
- Very fast tests benefit from packing GPUs onto fewer nodes, since fewer nodes lead to increased network performance.
- Tests with many particles appear to stress contention on the PCI bus, although the effect is modest.

On increasing the number of GPUs (§V-C2):

- Additional GPUs are only valuable if there is sufficient work to occupy them.
- Problems with idle time from load imbalance do not significantly benefit from extra GPUs; these problems may benefit early in their execution, but the later phases, characterized by excessive waiting, makes overall times comparable to those with fewer GPUs. We observed the effects of such imbalances from workloads with long durations.

VII. Conclusion

This study explored the impact of hardware architecture of a data-intensive workload (particle advection). It considered eleven architectures and 135 workloads, for a total of $1,485$ different tests. We feel the results illuminate the benefit of increased computational power that will inform future *in situ* workloads, as well as informing the choice of which hardware architecture is best for different particle advection problems in a distributed-memory parallel setting. (See §VI for specific findings.) We think this latter result is important because it will help visualization scientists answer two important questions: One, when collaborating with simulation scientists to do *in situ* analysis, which particle advection techniques will be appropriate given the resources? And, two, when presented with multiple hardware architecture options for running particle advection-based *post hoc* analysis, which hardware architecture is the best choice?

This study suggests several interesting future directions. Our study's tests used at most eight nodes. As more nodes are added, the opportunities for load imbalance increase, with the resulting idle time eroding potential hardware speedups. While we believe our present study is an excellent start on the problem and a substantial contribution for understanding matches between particle advection workloads and hardware architectures in a distributed-memory setting, we believe that looking at higher scales is an important next step.

Acknowledgment

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.

The study used many machines. The research used resources of the Keeneland Computing Facility at the Georgia Institute of Technology, which is supported by the National Science Foundation under Contract OCI-0910735. It used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. It used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The research was supported by a Major Research Instrumentation grant from the National Science Foundation, Office of Cyber Infrastructure, "MRI-R2: Acquisition of an Applied Computational Instrument for Scientific Synthesis (ACISS)," Grant OCI-0960354. (Note that many predecessor runs occurred on these machines, although none appear in our results section.)

References

[1] H. Childs, K.-L. Ma, H. Yu, B. Whitlock, J. Meredith, J. Favre, S. Klasky, N. Podhorszki, K. Schwan, M. Wolf, M. Parashar, and F. Zhang, "In Situ Processing," in *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, Oct. 2012, pp. 171–198.

[2] D. Camp, H. Krishnan, D. Pugmire, C. Garth, I. Johnson, E. W. Bethel, K. I. Joy, and H. Childs, "GPU Acceleration of Particle Advection Workloads in a Parallel, Distributed Memory Setting," in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, Girona, Spain, May 2013, pp. 1–8.

[3] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen, “Over Two Decades of Integration-Based, Geometric Flow Visualization,” in *EuroGraphics 2009 - State of the Art Reports*, April 2009, pp. 73–92.

[4] M. Ament, S. Frey, C. Müller, S. Grottel, T. Ertl, and D. Weiskopf, “GPU-Accelerated Visualization,” in *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, Oct. 2012, pp. 223–260.

[5] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, “Chromium: a Stream-Processing Framework for Interactive Rendering on Clusters,” in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’02. New York, NY, USA: ACM, 2002, pp. 693–702.

[6] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe, and T. Ertl, “Hierarchical Visualization and Compression of Large Volume Datasets Using GPU Clusters,” in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, May 2004, pp. 41–48.

[7] P. Bhaniramka, P. C. D. Robert, and S. Eilemann, “OpenGL Multipipe SDK: A Toolkit for Scalable Parallel Rendering,” in *Proceedings of the IEEE Visualization Conference*. IEEE Computer Society, 2005, p. 16.

[8] C. Müller, M. Strengert, and T. Ertl, “Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems,” in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*. Eurographics Association, 2006, pp. 59–66.

[9] S. Marchesin, C. Mongenet, and J.-M. Dischler, “Multi-GPU Sort Last Volume Visualization,” in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, April 2008.

[10] T. Fogal, H. Childs, S. Shankar, J. Krüger, R. D. Bergeron, and P. Hatcher, “Large Data Visualization on Distributed Memory Multi-GPU Clusters,” in *Proceedings of High Performance Graphics (HPG)*, Jun. 2010, pp. 57–66.

[11] A. Ancel, J.-M. Dischler, and C. Mongenet, “Load-Balanced Multi-GPU Ambient Occlusion for Direct Volume Rendering.” in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*. Eurographics Association, 2012, pp. 99–108.

[12] S. Martin, H.-W. Shen, and P. McCormick, “Load-Balanced Isosurfacing on Multi-GPU Clusters,” in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, May 2010, pp. 91–100.

[13] S. Bachthaler, M. Strengert, D. Weiskopf, and T. Ertl, “Parallel Texture-Based Vector Field Visualization on Curved Surfaces Using GPU Cluster Computers ,” in *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, 2006, pp. 75–82.

[14] D. Pugmire, T. Peterka, and C. Garth, “Parallel Integral Curves,” in *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, Oct. 2012, pp. 91–113.

[15] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. H. Weber, “Scalable Computation of Streamlines on Very Large Datasets,” in *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC09)*, Nov. 2009.

[16] T. Peterka, R. Ross, B. Nouanesengsey, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang, “A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields,” in *Proceedings of IPDPS 11*, Anchorage AK, 2011.

[17] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen, “Load-Balanced Parallel Streamline Generation on Large Scale Vector Fields,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1785–1794, 2011.

[18] C. Müller, D. Camp, B. Hentschel, and C. Garth, “Distributed Parallel Particle Advection using Work Requesting,” in *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, Atlanta, GA, Oct 2013, pp. 1–6.

[19] D. Weiskopf and T. Ertl, “GPU-Based 3D Texture Advection for the Visualization of Unsteady Flow Fields,” in *In WSCG 2004 Conference Proceedings, Short Papers*, 2004, pp. 259–266.

[20] K. Bürger, F. Ferstl, H. Theisel, and R. Westermann, “Interactive streak surface visualization on the gpu,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1259–1266, 2009.

[21] C. Garth and K. Joy, “Fast, memory-efficient cell location in unstructured grids for visualization,” *IEEE Transactions on Computer Graphics and Visualization*, vol. 16, no. 6, pp. 1541–1550, Nov. 2010.

[22] M. Bussler, T. Rick, A. Kelle-Emden, B. Hentschel, and T. Kuhlen, “Interactive particle tracing in time-varying tetrahedral grids,” in *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. Eurographics Association, 2011, pp. 71–80.

[23] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy, “Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture,” *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 17, pp. 1702–1713, Nov. 2011.

[24] P. Fischer, J. Lottes, D. Pointer, and A. Siegel, “Petascale Algorithms for Reactor Hydrodynamics,” *Journal of Physics: Conference Series*, vol. 125, pp. 1–5, 2008.

[25] E. Endeve, C. Y. Cardall, R. D. Budiardja, and A. Mezzacappa, “Generation of Magnetic Fields By the Stationary Accretion Shock Instability,” *The Astrophysical Journal*, vol. 713, no. 2, pp. 1219–1243, 2010.

[26] C. Sovinec, A. Glasser, T. Gianakon, D. Barnes, R. Nebel, S. Kruger, S. Plimpton, A. Tarditi, M. Chu, and the NIMROD Team, “Nonlinear Magnetohydrodynamics with High-order Finite Elements,” *J. Comp. Phys.*, vol. 195, p. 355, 2004.

[27] S. Shende and A. D. Malony, “TAU: The TAU Parallel Performance System,” *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.