Evaluating the Benefits of An Extended Memory Hierarchy for Parallel Streamline Algorithms

David Camp
dcamp@lbl.gov
Lawrence Berkeley National Laboratory and Department of Computer Science, University of California, Davis

Hank Childs
hchilds@lbl.gov
Lawrence Berkeley National Laboratory and Department of Computer Science, University of California, Davis

Amit Chourasia
amit@sdsc.edu
San Diego Supercomputer Center, University of California, San Diego

Christoph Garth
cgarth@ucdavis.edu
Department of Computer Science, University of California, Davis

Kenneth I. Joy
kijoy@ucdavis.edu
Department of Computer Science, University of California, Davis

DISCLAIMER: This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Acknowledgements: This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). The authors acknowledge the San Diego Supercomputer Center (SDSC) at The University of California at San Diego for time on the Dash supercomputer that led to the research results reported within this paper. Further, we thank Mahidhar Tateneni for his outstanding support and expertise in running on the machine.

Evaluating the Benefits of An Extended Memory Hierarchy for Parallel Streamline Algorithms

David Camp*
Lawrence Berkeley National Laboratory and Department of Computer Science, University of California, Davis

Hank Childs†
Lawrence Berkeley National Laboratory and Department of Computer Science, University of California, Davis

Amit Chourasia‡
San Diego Supercomputer Center, University of California, San Diego

Christoph Garth§
Department of Computer Science, University of California, Davis

Kenneth I. Joy¶
Department of Computer Science, University of California, Davis

Abstract

The increasing cost of achieving sufficient I/O bandwidth for high end supercomputers is leading to architectural evolutions in the I/O subsystem space. Currently popular designs create a staging area on each compute node for data output via solid state drives (SSDs), local hard drives, or both. In this paper, we investigate whether these extensions to the memory hierarchy, primarily intended for computer simulations that produce data, can also benefit visualization and analysis programs that consume data. Some algorithms, such as those that read the data only once and store the data in primary memory, can not draw obvious benefit from the presence of a deeper memory hierarchy. However, algorithms that read data repeatedly from disk are excellent candidates, since the repeated reads can be accelerated by caching the first read of a block on the new resources (i.e. SSDs or hard drives). We study such an algorithm, streamline computation, and quantify the benefits it can derive.

Index Terms: Programming Techniques [D.1.3]: Concurrent Programming—Parallel Programming, Computation by Abstract Devices [F.1.2]: Modes of Computation—Parallelism and Concurrency, Computer Graphics [I.3.3]: Picture/Image Generation—Display Algorithms

1 Introduction

As supercomputers get ever larger, the cost of achieving sufficient I/O bandwidth is, unsurprisingly, increasing. But supercomputing architects have been experimenting with a new approach to decrease this cost. Where the typical approach has a simulation write data directly to a parallel file system (i.e. "spinning disk"), the new approach introduces a new participant, solid state drives (SSDs), and has the simulation write data to the SSDs instead. The simulation can then immediately resume, while, concurrently, the data is copied from the SSDs to the file system, shielding the simulation from slow parallel file system performance. Although the SSDs introduce a new cost, they lessen the importance of I/O bandwidth, allowing for the SSDs to be coupled with a slower (and less expensive) parallel file system, providing a cost reduction overall.

To applications, this I/O configuration appears to have two distinct bandwidth characteristics. On write, the bandwidth appears to be good, since it is be accelerated by SSDs. On read, however, the bandwidth will be poor, since the reads are backed by a slower parallel file system and the presence of SSDs can not accelerate this activity.

I/O is often the slowest part of a visualization pipeline [10], hence suboptimal I/O read performance will result in poor overall visualization performance. However, in this paper, we ask whether SSDs can effectively increase I/O performance – and therefore visualization performance – by treating them as an extended part of the memory hierarchy. While the first read of any block of data will remain slow, the SSDs can be used as a cache to store those blocks, considerably accelerating subsequent reads. Further, local hard drives are appearing increasingly commonly in the I/O subsystem and can similarly be used as an extension to the memory hierarchy in the same fashion as SSDs. We also study how these hard drives can accelerate I/O and visualization performance.

Although many paradigms for processing data do not read blocks of data repeatedly, streamline calculations do. Streamlines, or more generally integral curves, are one of the most illuminating techniques to obtain insight from simulations that involve vector fields and they serve as a cornerstone of visualization and analysis across a variety of application domains. Drawing on an intuitive interpretation in terms of particle movement, they are an ideal tool to illustrate and describe a wide range of phenomena encountered in the study of scientific problems involving vector fields, such as transport and mixing in fluid flows. Moreover, they are used as building blocks for sophisticated visualization techniques (e.g., [14, 17, 19]), which typically require the calculation of large amounts of integral curves. Successful application of such techniques to large data must crucially leverage parallel computational resources to achieve well-performing visualization.

Among visualization techniques in general, streamline-based approaches are notoriously hard to parallelize in a distributed memory setting [23], because runtime characteristics are highly problem- and data-dependent. In terms of parallelization approach, streamline computations may be parallelized over data, parallelized over streamlines, or some hybrid between the two. When parallelizing over streamlines (or, equivalently, over their seed points), particles are advected and blocks of data loaded dynamically based on the trajectory taken. This is exactly the data processing pattern that can benefit from an extended memory hierarchy and we study this approach here.

In this paper, we study the benefits a local disk – either SSD or local hard drive – can provide to accelerate a parallel streamline algorithm. We perform a variety of tests and present results to show what I/O benefits can be gained with the use of an SSD or local hard drive compared to a parallel file system.

2 Related Work

2.1 Parallel Particle Advection

The parallel solution of streamline-based problems has been considered in previous work using a multitude of differing approaches. Generally, both data set, represented as a number of disjoint blocks,

*e-mail: dcamp@lbl.gov
†e-mail: hchilds@lbl.gov
‡e-mail: amit@sdsc.edu
§e-mail: cgarth@ucdavis.edu
¶e-mail: kijoy@ucdavis.edu

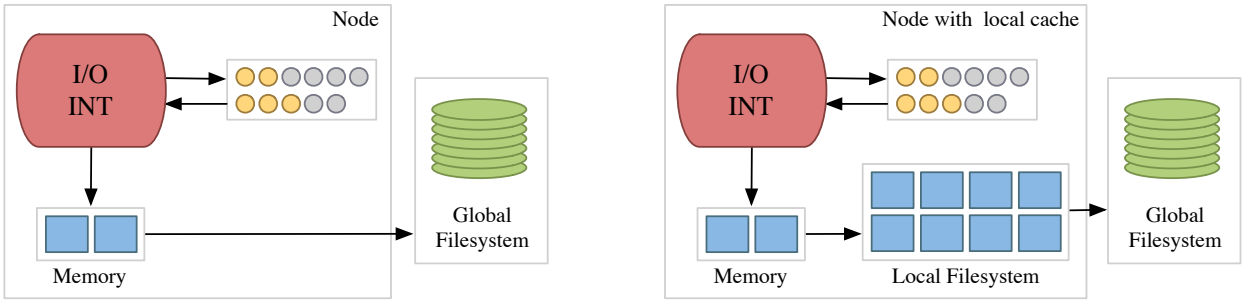

Figure 1: The *parallelize-over-seeds* algorithm with no local cache (left) and local cache (right) versions. Each process integrates streamlines (INT) and manages its own memory cache by loading blocks from disk (I/O). In the local cache version, each MPI task has an extra block cache on the local file system which it can check for data blocks before going to the global file system. Each MPI task is identical and only one is shown here. Similarly, MPI communication is limited to gathering results and is also not shown. See Section 3 for more discussion of this algorithm.

and computation, in the form of integration work, can be distributed. An early treatment of the topic was given by Sujudi and Haimes [27], who made use of distributed computation by assigning each processor one data set block. A streamline is communicated among processors as it traverses different blocks. Other examples of applying parallel computation to streamline-based visualization include the use of multiprocessor workstations to parallelize integral curve computation [18], and research efforts that focus on accelerating specific visualization techniques [4]. Similarly, PC cluster systems were leveraged to accelerate advanced integration-based visualization algorithms, such as time-varying Line Integral Convolution (LIC) volumes [21] or particle visualization for very large data [11].

Focusing on data size, out-of-core techniques are commonly used in large-scale data applications where data sets are larger than main memory. These algorithms focus on achieving optimal I/O performance to access data stored on disk. For vector field visualization, Ueng et al. [29] presented a technique to compute streamlines in large unstructured grids using an octree partitioning of the vector field data for fast fetching during streamline construction using a small memory footprint. Taking a different approach, Bruckschen et al. [3] described a technique for real-time particle traces of large time-varying data sets by isolating all integral curve computation in a pre-processing stage. The output is stored on disk and can then be efficiently loaded during the visualization phase.

More recently, different partitioning methods were introduced with the aim of optimizing parallel integral curve computation. Yu et al. [30] introduced a parallel integral curve visualization that computes a set of representative, short trajectory segments termed *pathlets* in time-varying vector fields. A preprocessing step computes a binary clustering tree that is used for seed point selection and block decomposition. This seed point selection method mostly eliminates the need for communication between processors, and the authors are able to show good scaling behavior for large data. However, this scaling behavior comes at the cost of increased preprocessing time and, more importantly, loses the ability to choose arbitrary, user-defined seed-points, which is often necessary when using streamlines for data analysis as opposed to obtaining a qualitative data overview. Chen and Fujishiro [8] applied a spectral decomposition using a vector-field derived anisotropic differential operator to achieve a similar goal with similar drawbacks.

Finally, recent studies have aimed to better understand the characteristics of parallel streamline performance. Pugmire et al. presented a comparison of three parallelization algorithms [23]: the first two correspond to parallelization over seed points and over data blocks, respectively, while the third algorithm (termed *Master-Slave*) adapts its behavior between these two extremes based on the observed behavior during the algorithm run. Peterka et al. focused on streamlines at extreme scale, including concurrency levels up 32K cores and data sets as big as $2304 \times 4096 \times 4096$ [22].

2.2 SSDs in Supercomputing

In the past few years, solid-state storage has successfully transitioned from small, embedded devices such as media players or mobile phones, to larger systems, e.g. desktop computers. Trends such as rising storage density, lower power consumption and higher performance with respect to conventional hard drives, coupled with decreasing price, has made SSDs an attractive technology for data centers and high performance computing (HPC). The increasing cost to achieve sufficient bandwidth has led HPC architects to rethink and redesign the I/O subsystem of supercomputers [7], using SSDs to increase bandwidth and reduce latency. The *Dash* cluster [15] at San Diego Supercomputer Center and the *Hyperion* cluster at Lawrence Livermore National Laboratory [16] are two forerunners in adopting this approach and demonstrating the advantages in the HPC space. The *Dash* cluster, which we used to perform our experiments, uses NAND Flash I/O nodes on a SATA bus, whereas the *Hyperion* cluster uses Fusion-I/O PCI-express cards. The HPC community has been eager to explore this new design and performance modeling of Flash-augmented HPC clusters is an area of active research ([2], [7], [20]).

Szalay et al. [28] showed that you can combine energy efficient CPUs with SSDs to increase sequential read throughput by an order of magnitude while keeping power consumption constant.

He et al. [15] describe three data analysis applications – external memory breadth first search, astrophysics analysis from the Palomar Transient Factory, and biological pathways analysis – where performance improves from 5X to 100X by using solid-state memory instead of spinning disk storage. Each of the algorithms considered in their paper use a data processing paradigm that benefits from the extended memory hierarchy that SSDs provide. In this paper, we further the evidence for the benefits of an extended memory hierarchy, by demonstrating its utility in the visualization space.

3 Streamline Calculation

Streamline computation for visualization purposes typically relies on numerical integration methods that access vector field data in random-access fashion to iteratively construct a streamline. Since individual streamlines are computationally mutually independent, the problem of computing streamlines lends itself well to parallelization over the individual particles. For other parallelization approaches, we refer the reader to the discussion in [23].

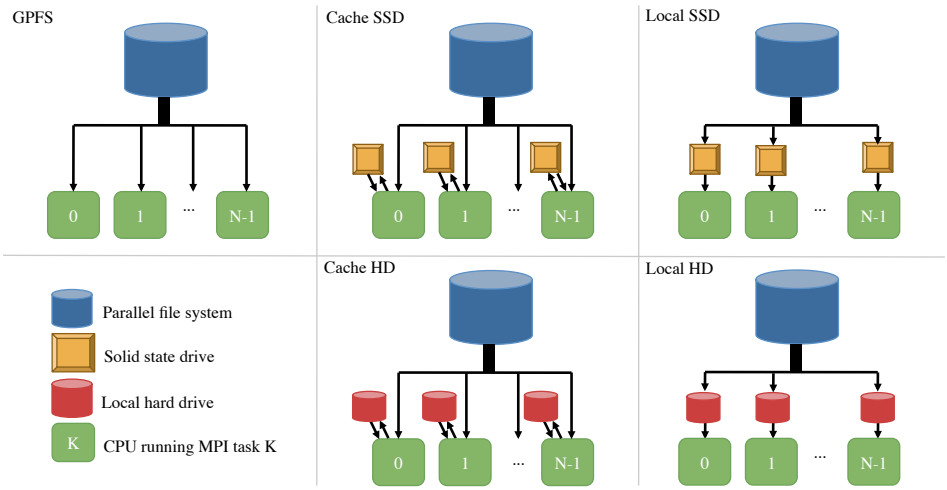

Figure 2: Configurations for loading data. See Section 4.1 for a complete description.

As previously stated, we are studying the parallelize-over-seeds approach (see Figure 1). We assume that the problem mesh is decomposed into a number of spatially disjoint data blocks and that any of these blocks can be loaded when required, i.e. when a streamline integration must continue into a block that is not present in memory. The algorithm partitions the set of seed points over MPI tasks, assigning each MPI task a fixed number of seed points from which streamlines are then integrated. The initial partitioning of seed points to nodes is based on spatial proximity of the seeds, following the reasoning that the particles traced from spatially close seed points are likely to traverse the same regions, and thus blocks, of a data set. Parallelize-over-seeds benefits from a small amount of communication; it only occurs at initialization and termination.

Each MPI task maintains a cache in its primary memory for loaded data blocks, relieving the need to load blocks repeatedly from the parallel file system. A new block is loaded when no streamline can continue with the current block or with any blocks in the memory cache. Naturally, only a small number of blocks can be cached and still satisfy primary memory constraints. In our algorithm, blocks are removed from the memory cache when a fixed maximal number of blocks is exceeded (20 for this study). They are evicted in least recently used order to make room for new blocks. We find this approach effective in reducing overall I/O cost, and especially for dense seeding. Overall, the performance of the parallelize-over-seeds scheme depends crucially on loading and caching of blocks, with small cache size meaning that blocks must be brought in from the parallel file system repeatedly (cf. [23, 5] for an in-depth investigation of these properties).

The extended memory hierarchy considered in this paper effectively increases the size of the cache, albeit with additional overhead for sending and retrieving blocks. When utilizing the SSDs or local hard drives, a block evicted from the primary memory cache is then saved to the local disk. Further, when a data block is needed and is not found in the primary memory cache, the algorithm checks first to see if the data block is on the local disk before loading it from the remote file system. With this change we expect to gain performance both from the faster local disks loads and also from the reduced load on the parallel file system.

4 Experimental Configuration

To best understand the performance characteristics of a parallelize-over-seeds streamline algorithm with an extended memory hierarchy, we perform a series of tests that vary both I/O configuration (4.1) and streamline configuration (4.2). We also describe the machine and software framework used to perform the experiment (4.3).

4.1 I/O configurations

We examine five different I/O configurations (see Figure 2). In the first variant, which we denote *GPFS*, each MPI task loads data blocks directly from the parallel file system, establishing a baseline for performance without an extended memory hierarchy. The acronym GPFS stands for the "General Parallel File System" [25]. In the second variant, *Cache SSD*, each MPI task can load data blocks from either the SSD or from the parallel file system. For each load, it starts by checking the SSD, since its load are considerably faster. If the SSD does not contain the data, then it loads the block from the parallel file system and stores it back to the SSD, meaning subsequent loads of the block will come from the faster SSD. In the third variant, *Local SSD*, a preprocessing step is applied where the entire data set is copied to each SSD before execution begins. In this variant, every processor is able to fetch data directly from its SSD and does not have to deal with the parallel file system. Note that this scenario is only possible when the data to be processed is smaller than the size of the SSD and, further, requires a large initialization cost. The fourth and fifth variants, *Cache HD* and *Local HD*, are identical to the second and third, except that the local hard drive is used in the place of the SSD. These tests determine the performance differences between SSDs and local hard drives.

4.2 Streamline Configuration

In [23], the authors present an overview of the complexities of characterizing parallel streamline performance. We briefly summarize their points here. The time to calculate streamlines primarily comes down to three factors: integration time, block retrieval time, and idle time. These four factors in turn depend on the input data set and the seed points. Specifically:

- **Seed Set Size** The amount of integration time will be closely related to the number of seeds.
- **Seed Set Distribution** In the parallelize-over-seeds algorithm (studied in this paper), dense seed sets are favorable, since they lead to less block loads, as so many are re-used from the memory cache. Note that in a parallelize-over-data algorithm, however, this configuration leads to significant idle time.

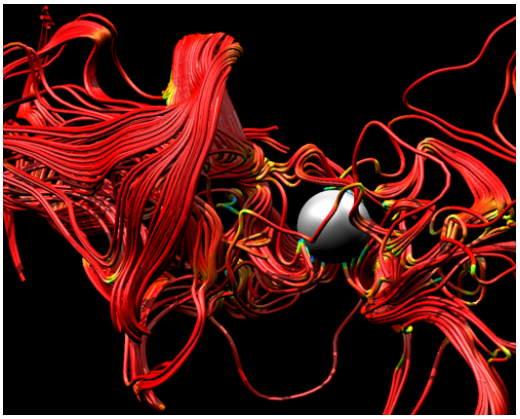

Figure 3: A core-collapse supernovae simulation from the GENASIS code. The streamlines show activity in the magnetic field at the core.

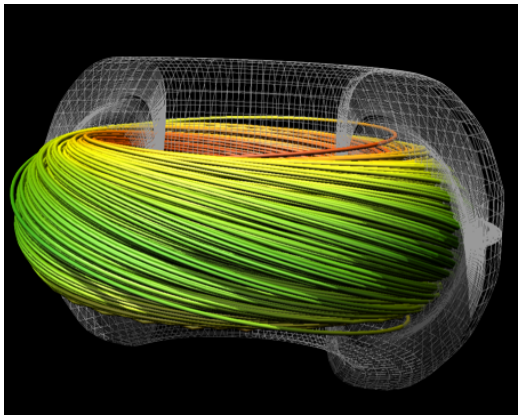

Figure 4: A tokamak simulation from the NIMROD code. The streamlines wrap around the tokamak many times, demonstrating the streamlines used for Poincare analysis [24].

- **Data Set Size** High resolution meshes do not necessarily equate to prohibitive numbers of block loads. A single streamline, for example, likely only traverses a few blocks. However, an increase in mesh resolution typically results in proportionally higher number of block loads. Additionally, the ability to fit the entire data set in primary memory or the entire data set in the extended memory hierarchy is a crucially determining factor for performance. In this paper, we consider data sets that fit in the extended memory hierarchy, although considerably larger data sets will continue to reap benefits from larger caches.

- **Vector Field Complexity** The structure of the vector field determines the path taken by particles, which affects both the number of blocks to load and the computation time. Furthermore, particles can converge to relatively small regions or diverge from a small region into a large part of the dataset, resulting in analogous characteristics to those found when considering the influence of the seed set distribution. For the parallelization-over-seeds approach, converging streamlines represent the most favorable case as block reuse is increased in this scenario.

In order to account for these factors, our study looks at three data sets, using a varying number of seed points.

Astrophysics This data set results from a simulation of the magnetic field surrounding a solar core collapse resulting in a supernova (see Figure 3). The search for the explosion mechanism of a core-collapse within a supernovae and the computation of the nucleosynthesis in these spectacular stellar explosions is one of the most important and most challenging problems in computational nuclear astrophysics. Understanding the magnetic field around the core is very important and streamlines are a key technique for doing so. The simulation was computed by a GENASIS simulation [12], a multi-physics code being developed for a simulation of astrophysical systems involving nuclear matter [6]. GENASIS computes the magnetic field at each cell face. For the purposes of this study, a cell-centered vector is created by differencing the values at faces in the X, Y and Z directions. Node-centered vectors are generated by averaging adjacent cells to each node. To see how this algorithm would perform on very large data sets, the magnetic field was upsampled to a total of 512 blocks with 1 million cells per block, for a total resolution of 800^3 and a data size of 4 gigabytes. The seed set was placed randomly in a small box around the collapsing core. The small and large seed sets contained 2,500 and 10,000 seed points respectively. Both small and large seed sets are integration with a time of 4,000 time units.

Fusion The second data set is from a simulation of magnetically confined fusion in a tokamak device (see Figure 4). The development of magnetic confinement fusion, which will be a future source of low cost power, is an important area of research. Physicists are particularly interested in using magnetic fields to confine the burning plasma in a toroidal shape device, known as a tokamak. To achieve stable plasma equilibrium, the field lines of these magnetic fields need to travel around the torus in a helical fashion. Using streamlines the scientist can visualize the magnetic fields. The simulation was performed using the NIMROD code [26]. This data set has the unusual property that most streamlines are approximately closed and traverse the torus-shaped vector field domain repeatedly which stresses the data cache. For the tests conducted here, we resampled the data to 512 blocks with 1 million cells per block. The seed set was placed randomly on a small box inside the torus. Here, 2,500 seed points were used for the small seed, and the large seed sets contain 10,000 seeds with both cases using an integration time of 20 time units.

Thermal Hydraulics The third data set results are from a thermal hydraulics simulation (see Figure 5). Here, twin inlets pump air into a box, with a temperature difference between the air inserted by each inlet; eventually the air exits through an outlet. The mixing behavior and the temperature of the air at the outlet are of interest. Non-optimal mixing can be caused by long-lived recirculation zones that effectively isolate certain regions of the domain from the heat exchange. This simulation was performed by using the NEK5000 code [13] on an unstructured grid comprised of twenty-three million hexahedral elements. Again, resampling was performed to a regular mesh of 512 blocks with 1 million cells per block. The seed set was placed densely around one of the inlets to examine the behavior of particles entering through it. The resulting streamlines illustrate the turbulence in the immediate vicinity of the inlet. Small seed sets contained 2,500 seed points and the large case consists of 10,000 seed points. Both cases were advected for 12 time units.

4.3 Runtime Environment

All experiments were conducted on the Dash machine (see Section 2.2). Dash is a data intensive compute cluster comprising of thirty-two compute and two I/O nodes connected by DDR Infiniband. All nodes are equipped with two Intel 2.4 GHz Xeon Nehalem E5530 processors with 48 GB of local DDR3 DRAM mem-

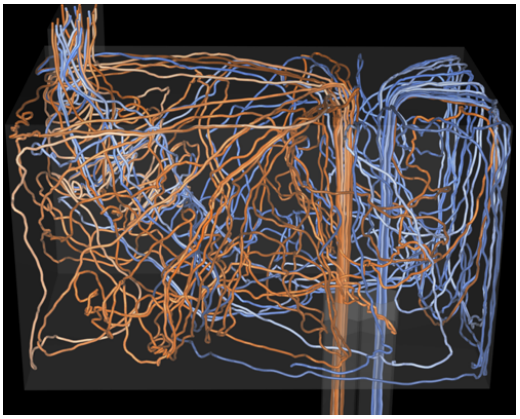

Figure 5: A thermal hydraulics simulation from the NEK5000 code. The streamlines show the behavior of warm and cool air inlets (orange and blue respectively) and representative behavior for how long particles remain in the assembly before exiting through the outlet (top left).

ory. Each I/O node has an additional 16 Intel X25-E 64 GB flash-based SSD with the total capacity of 1TB. For local hard drives, compute nodes are equipped with Seagate Momentus 7200 RPM hard drive with approximately 250 GB capacity. Finally, we used Dash's GPFS "spinning disk" for all remote file access in our tests.

Our measurements are based on the software infrastructure provided by the VISIT [1, 9] visualization system, an end user visualization and analysis tool for large data sets. We augmented the existing implementation of parallelize-over-seeds that is included in recent VisIt releases to support the caching mechanism outlined above and added instrumentation for the measurements described in Section 5.1.

Benchmarks were performed during full production use of the system to capture a real-world scenario. No special measures were taken to exclude operating system I/O caching. The default queuing system (QSUB) was used to distribute the nodes and cores as required. Each benchmark run was performed using 64 cores (8 nodes).

5 EXPERIMENTS

5.1 Measurements

To obtain insight into the relative benefits of the local cache approach to streamline integration, we have obtained a number of timings and other statistics beyond the pure execution time T_{Total} of the corresponding combination of algorithm and test cases.

The application keeps track of the time spent executing various functions using an event log consisting of pairs of timestamp and event identifier. Events include, for instance, start and end of integration for a particular streamline, begin and end of an I/O operation, and time spent performing communication using MPI calls. Timestamps are taken as wall time elapsed since the start of the MPI task. These event logs are carefully implemented to have negligible impact on the overall runtime behavior, and analyzed later to provide the summary statistics discussed in the following and represented in Table 1.

The pure integration time represents the actual computational workload. This time should be almost independent across each data set with the same seed point size, since the integration workload is in terms of the number of integration steps taken are identical in each of the small or large test case. Therefore we have not listed the integration time for each test, although the integration time is part of the total running time for each test. In total, we perform

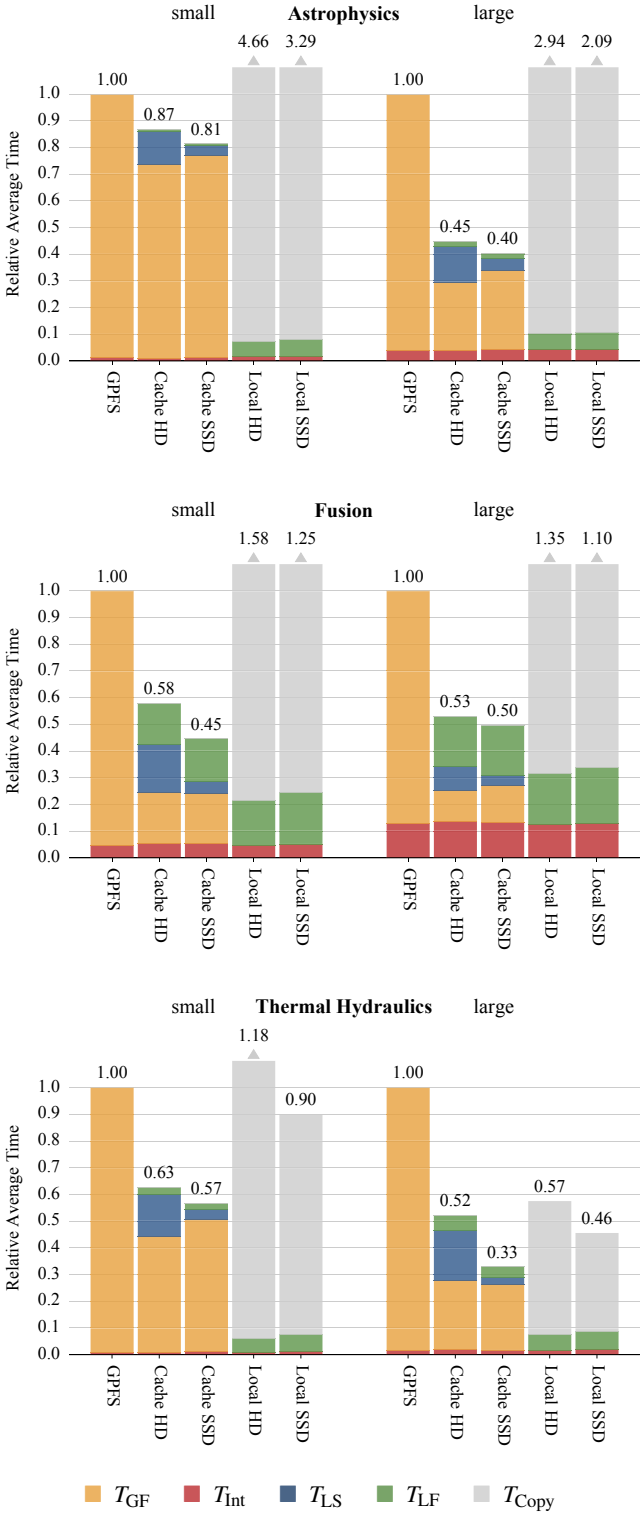

Figure 6: An overview of timings spent for performed tests. Tests are grouped according to dataset and seed set size. Within each group, timings are indicated relative to the time measured for the *GPFS* I/O configuration. Yellow is T_{GF}, the I/O time from **G**PFS **F**etches, red is T_{Int}, the **Int**egration time, blue is T_{LS}, the time for **L**ocal disk **S**tores, green is T_{LF}, time for **L**ocal disk **F**etches, and gray is T_{copy}, the initialization time to copy data to the local disk.

		I/O Conf.	T_{Total}	$T_{\text{Total}*}$	T_{IO}	N_{GF}	T_{GF}	N_{LF}	T_{LF}	N_{LS}	T_{LS}	$\%_{\text{Accel}}$
Astro	small	GPFS	25.17s	25.17s	24.89s (98%)	5275	24.89s	–	–	–	–	0%
		Cache HD	21.81s	21.81s	21.60s (99%)	4456	18.31s	819	0.20s	3275	3.10s	16%
		Cache SSD	20.47s	20.47s	20.20s (98%)	4456	19.14s	819	0.18s	3275	0.88s	16%
		Local HD	117.37s	1.80s	1.37s (76%)	–	–	5275	1.37s	–	–	100%
		Local SSD	82.85s	1.97s	1.53s (77%)	–	–	5275	1.53s	–	–	100%
	large	GPFS	40.80s	49.80s	39.21s (96%)	9994	39.21s	–	–	–	–	0%
		Cache HD	18.26s	18.26s	16.71s (91%)	7059	10.40s	2935	0.77s	5890	5.55s	29%
		Cache SSD	16.48s	16.48s	14.75s (89%)	7059	12.03s	2935	0.78s	5890	1.95s	29%
		Local HD	119.83s	4.26s	2.54s (59%)	–	–	9994	2.54s	–	–	100%
		Local SSD	85.16s	4.28s	2.57s (59%)	–	–	9994	2.57s	–	–	100%
Fusion	small	GPFS	80.90s	80.90s	77.20s (95%)	51251	77.20s	–	–	–	–	0%
		Cache HD	46.79s	46.79s	42.48s (90%)	8642	15.55s	42609	12.55s	8453	14.38s	83%
		Cache SSD	36.07s	36.07s	31.68s (87%)	8642	15.05s	42609	12.85s	8453	3.78s	83%
		Local HD	128.07s	17.31s	13.61s (78%)	–	–	51251	13.61s	–	–	100%
		Local SSD	101.13s	19.68s	15.53s (78%)	–	–	51251	15.53s	–	–	100%
	large	GPFS	107.04s	107.04s	93.16s (87%)	79467	93.16s	–	–	–	–	0%
		Cache HD	56.79s	56.79s	42.39s (74%)	9907	12.59s	69560	20.16s	9714	9.64s	87%
		Cache SSD	53.07s	53.07s	38.78s (73%)	9907	14.57s	69560	20.14s	9714	4.07s	87%
		Local HD	144.65s	33.89s	20.60s (60%)	–	–	79467	20.60s	–	–	100%
		Local SSD	117.75s	36.30s	22.62s (62%)	–	–	79467	22.62s	–	–	100%
Thermal Hydraulics	small	GPFS	98.88s	98.88s	98.22s (99%)	19085	98.22s	–	–	–	–	0%
		Cache HD	61.81s	61.81s	60.85s (98%)	10278	42.88s	8807	2.39s	9888	15.58s	46%
		Cache SSD	55.92s	55.92s	54.95s (98%)	10278	49.20s	8807	2.16s	9888	3.59s	46%
		Local HD	116.48s	6.04s	5.13s (84%)	–	–	19085	5.13s	–	–	100%
		Local SSD	88.85s	7.38s	6.29s (85%)	–	–	19085	6.29s	–	–	100%
	large	GPFS	220.68s	220.68s	217.23s (98%)	48188	217.23s	–	–	–	–	0%
		Cache HD	115.39s	115.39s	111.47s (96%)	14099	57.38s	34089	12.54s	13717	41.54s	71%
		Cache SSD	72.56s	72.56s	68.80s (94%)	14099	54.56s	34089	9.13s	13717	5.10s	71%
		Local HD	126.84s	16.40s	12.93s (78%)	–	–	48188	12.93s	–	–	100%
		Local SSD	100.44s	18.97s	15.14s (79%)	–	–	48188	15.14s	–	–	100%

Table 1: Results from our thirty tests. For each of the thirty tests, four runs were performed (130 total) and the fastest result was reported, in an effort to remove effects from contention. Section 5 contains complete test descriptions. We summarize notation from that section that relates to this table:

- T_{Total} is the total execution time, averaged over all MPI tasks.
- $T_{\text{Total}*}$ is the total execution time with out the transfer time of the data set to local disk, averaged over all MPI tasks.
- T_{IO} is the time spent doing I/O, averaged over all MPI tasks.
- N_{GF} is the number of **G**PFS block **F**etches, averaged over all MPI tasks.
- T_{GF} is the time spent doing **G**PFS block **F**etches, averaged over all MPI tasks.
- N_{LF} is the number of **L**ocal disk block **F**etches, averaged over all MPI tasks.
- T_{LF} is the time spent doing **L**ocal disk block **F**etches, averaged over all MPI tasks.
- N_{LS} is the number of **L**ocal disk block **S**tores, averaged over all MPI tasks.
- T_{LS} is the time spent doing **L**ocal disk block **S**tores, averaged over all MPI tasks.
- $\%_{\text{Accel}}$ is the percentage of block fetches that were accelerated via local disk.

30 different tests, to cover a wide range of configurations and real-world scenarios. They consist of three different data sets, two different seed point set sizes, and the five I/O configurations described in Section 4.1. Each of these tests were run four times and the best time was used in reporting results.

5.2 Global Parallel File System (GPFS)

In this test, each MPI process loads data directly from GPFS and advects the particle trajectory. This test represents a baseline and is used to quantify the relative improvements the other I/O configurations can achieve. In Figure 6, we use the GPFS tests as a reference value of 1.0. Furthermore, we track the number of blocks N_{GF} fetched and the average time spent (over all MPI tasks) fetching blocks T_{GF} from GPFS.

5.3 Local SSD and Local HD

In the local disk tests, each MPI process loads data directly from the local SSD or HD to calculate the streamline. This is theoretically optimal since all of the data is local and can be read quickly. To compare the performance difference between them, we examine both the use of an SSD and a hard drive disk.

In order for this test to be executed, the data set has to be transferred from the remote file system to each of the processing nodes; this adds significant overhead if included in the tests. Note that due to the data-dependent nature of block accesses, the entire dataset must be copied. In our experiments, the transfer is performed in parallel with each node copying the data set to the local drive before streamline integration is started. Refer to Figure 6 or Table 1 for exact transfer times, denoted T_{Copy}. Again, we track the number of blocks fetched, N_{LF}, and the average time (over all MPI tasks) to fetch blocks from local disk T_{LF}.

5.4 Cache SSD and Cache HD

In these tests, each MPI process loads the data block from GPFS upon first occurrence and caches the data block to local storage when the memory cache is full. Naturally, if a block was earlier evicted to local storage, it is reloaded from there.

Storing block on local storage adds to overall I/O time, and number of blocks and corresponding I/O time are measured as above (N_{LS} and T_{LS}). Furthermore, we again track frequency and time spent reloading blocks (N_{LF} and T_{LF}).

6 Results and Analysis

We were most interested in the "Cache HD" and "Cache SSD" tests. "Local HD" and "Local SSD" suffered from such prohibitively large initialization costs that only the Thermal Hydraulics tests ran long enough to amortize them. And "GPFS" tests do not use the extended memory hierarchy; they were only run to provide a baseline. On the whole, most "Cache HD" or "Cache SSD" configurations ran twice as fast as compared to its baseline "GPFS" configuration. The "small, Thermal Hydraulics" had a positive outlier with the Cache SSD configuration (three times faster) and a negative outlier with "large, Astrophysics" (only 20% faster).

We can consider a rough model in determining how much the extended memory hierarchy can benefit a streamline calculation. Assume a "GPFS" version of a test has T_{Int} integration time, fetches N_{GF} blocks from the file system, and takes A_{GF} seconds to fetch a block. Then the time to run the test is:

$$T_{Total} = T_{Int} + N_{GF} \cdot A_{GF} \tag{1}$$

Now assume that A_{LS} is the time to store a block to local disk and A_{LF} is the time to fetch a block from local disk. Further, assume that N_A of the reads can be accelerated and $N_{\bar{A}}$ can not be accelerated ($N_A + N_{\bar{A}} = N_{GF}$). Then we can predict the performance with an extended memory hierarchy:

$$T_{Total} = T_{Int} + N_{\bar{A}} \cdot A_{GF} + N_{\bar{A}} \cdot A_{LS} + N_A \cdot A_{LF} \tag{2}$$

Given this framework, we can make additional observations about the tests:

- A_{GF} is not constant. Block loads took between 11ms and 51ms, and seemed to get faster with more blocks were loaded. We speculate that operating system-level caching provided this gain. However, we feel that the increasing performance with increasing usage underscores the importance of having a diverse set of streamline tests.

- A_{LS} and A_{LF} remained more steady for both SSDs and local hard drives. For both technologies, A_{LF}, the fetch time for a block, was approximately 0.25ms. However, their storage times, A_{LS}, did vary. For SSDs, storage ranged from 0.25ms to 0.40ms, while HDs ranged from 1ms to 3ms. Both technologies got slower with increased usage, presumably due to contention issues.

- The number of blocks stored to local cache varied greatly in our tests, from 3,275 to 13,717. SSDs performed from two to eight times faster than the hard drives, making an appreciable differences in the overall performance. This is clearly seen in "large, Thermal Hydraulics," which had 72.56s for "Cache SSD" and 115.39s for "Cache HD."

- $\%_{Accel}$, the number of block loads that can be accelerated by local cache, is a good predictor of performance gain. This follows from our framework above, but is also supported by our tests. "small, Astro" has only 16% of its blocks accelerated, and only is 20% faster. But "large, Fusion" has 87% of its blocks accelerated and is 50% faster. Of course, streamline performance is complex; "large, Thermal Hydraulics" has a smaller number of blocks accelerated, 71%, but gets almost a three times performance increase, partially because that test is almost entirely I/O-bound (and does much less integration).

Our experiments appear to have well-covered the complex space of parallel streamline computation. Despite identical data sizes and similar numbers of seed points, our six basic tests had an order of magnitude variation in the number of block loads, from 5,275 to 79,467. Further, as discussed in the preceding paragraph, we feel that a key factor in predicting the benefits of the extended memory hierarchy is $\%_{Accel}$, the percentage of block loads that can be accelerated. Our tests had values of 16%, 29%, 46%, 71%, 83%, and 87%, providing reasonable coverage of this space.

7 Conclusion and Future Work

We presented a study measuring the benefits of an extended memory hierarchy for parallel streamline algorithms. Overall, maintaining a local cache created an overall speedup of approximately a factor of two. Further, our experiments were designed to emphasize different aspects of streamline performance and those aspects in turns emphasized different hardware characteristics. Specifically, we found that the fast storage times of SSDs led to significant gains over local hard drives in some cases. Further, we found that the approach of having the operating system copy the data to local cache as a pre-processing step was prohibitively slow, and that progressively bringing data off the parallel file system and having the application cache the data manually was a superior strategy.

In this study, each MPI task uses the local disk without coordinating with the other MPI tasks on a node. In the future, we wish to better make collective use of the local disk and study further performance gains. Of course, this sort of scheme would require a mechanism for synchronizing writes between the MPI tasks, as well as knowledge of blocks in the cache.

Finally, this study looked at a data set that could fit entirely within the local disk provided by the Dash machine. As SSDs are

an optimization intended for the truly huge next generation of supercomputers, and since the storage available in an SSD is likely to stay relatively constant, it is not reasonable to assume that the data will always be able to fit. As the extended memory hierarchy effectively extends the cache size, and since a larger cache size is always beneficial, the extension to the parallelize-over-seeds algorithm will continue to help. However, a study considering even larger data (larger than the size of local disk) would be useful in quantifying the effect. We would also like to repeat these tests on the upcoming Gordon supercomputer at SDSC which will have faster and larger capacity SSD cache.

ACKNOWLEDGEMENTS

This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). The authors acknowledge the San Diego Supercomputer Center (SDSC) at The University of California at San Diego for time on the Dash supercomputer that led to the research results reported within this paper. Further, we thank Mahidhar Tatineni for his outstanding support and expertise in running on the machine.

REFERENCES

[1] VISIT – Software that delivers Parallel, Interactive Visualization. `http://visit.llnl.gov/`.

[2] D. Ajwani, A. Beckmann, R. Jacob, U. Meyer, and G. Moruz. On Computational Models for Flash Memory Devices. In J. Vahrenhold, editor, *Experimental Algorithms*, volume 5526 of *Lecture Notes in Computer Science*, pages 16–27. Springer Berlin / Heidelberg, 2009.

[3] R. Bruckschen, F. Kuester, B. Hamann, and K. I. Joy. Real-Time Out-of-Core Visualization of Particle Traces. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, pages 45–50, Piscataway, NJ, USA, 2001. IEEE Press.

[4] B. Cabral and L. C. Leedom. Highly Parallel Vector Visualization Using Line Integral Convolution. In *Proceedings of SIAM PPSC '95*, pages 802–807, 1995.

[5] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy. Streamline integration using MPI-hybrid parallelism on a large multi-core architecture. *IEEE Transactions on Visualization and Computer Graphics*, 2011 (to appear).

[6] C. Y. Cardall, A. O. Razoumov, E. Endeve, E. J. Lentz, and A. Mezzacappa. Toward Five-Dimensional Core-Collapse Supernova Simulations. *Journal of Physics: Conference Series*, 16:390–394, 2005.

[7] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson. Understanding the impact of emerging non-volatile memories on high-performance, io-intensive computing. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[8] L. Chen and I. Fujishiro. Optimizing Parallel Performance of Streamline Visualization for Large Distributed Flow Datasets. In *Proceedings of IEEE VGTC Pacific Visualization Symposium 2008*, pages 87–94, 2008.

[9] H. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. J. Whitlock, and N. Max. A Contract-Based System for Large Data Visualization. In *Proceedings of IEEE Visualization*, pages 190–198, 2005.

[10] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. Weber, and E. W. Bethel. Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, May/June 2010. LBNL-3403E.

[11] D. Ellsworth, B. Green, and P. Moran. Interactive Terascale Particle Visualization. In *Proceedings of IEEE Visualization*, pages 353–360, Washington, DC, USA, 2004. IEEE Computer Society.

[12] E. Endeve, C. Y. Cardall, R. D. Budiardja, and A. Mezzacappa. Generation of Strong Magnetic Fields in Axisymmetry by the Stationary Accretion Shock Instability. *ArXiv e-prints*, Nov. 2008.

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

[14] C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. Efficient Computation and Visualization of Coherent Structures in Fluid Flow Applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1464–1471, 2007.

[15] J. He, A. Jagatheesan, S. Gupta, J. Bennett, and A. Snavely. Dash: a recipe for a flash-based data intensive supercomputer. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[16] Hyperion. Hyperion project – collaboration for an advanced technology cluster testbed. `https://hyperionproject.llnl.gov/`.

[17] H. Krishnan, C. Garth, and K. I. Joy. Time and Streak Surfaces for Flow Visualization in Large Time-Varying Data Sets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1267–1274, 2009.

[18] D. A. Lane. UFAT – A Particle Tracer for Time-Dependent Flow Fields. In *Proceedings of IEEE Visualization '94*, pages 257–264, 1994.

[19] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[20] M. Meswani, P. Cicotti, J. He, and A. Snavely. Predicting disk i/o time of hpc applications on flash drives. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1926 –1929, dec. 2010.

[21] S. Muraki, E. B. Lum, K.-L. Ma, M. Ogata, and X. Liu. A PC Cluster System for Simultaneous Interactive Volumetric Modeling and Visualization. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, page 13, Washington, DC, USA, 2003. IEEE Computer Society.

[22] 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 *To appear in Proceedings of IPDPS 11*, Anchorage AK, 2011.

[23] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. Weber. Scalable Computation of Streamlines on Very Large Datasets. In *Proceedings of Supercomputing*, 2009.

[24] A. Sanderson, G. Chen, X. Tricoche, D. Pugmire, S. Kruger, and J. Breslau. Analysis of recurrent patterns in toroidal magnetic fields. In *Proceedings Visualization / Information Visualization 2010*, volume 16 of *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[25] F. Schmuck and R. Haskin. Gpfs: A shared-disk file system for large computing clusters. In *In Proceedings of the 2002 Conference on File and Storage Technologies (FAST*, pages 231–244, 2002.

[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.*, 195:355, 2004.

[27] D. Sujudi and R. Haimes. Integration of Particles and Streamlines in a Spatially-Decomposed Computation. In *Proceedings of Parallel Computational Fluid Dynamics*, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[28] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White. Low-power amdahl-balanced blades for data intensive computing. *SIGOPS Oper. Syst. Rev.*, 44:71–75, March 2010.

[29] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-Core Streamline Visualization on Large Unstructured Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, 1997.

[30] H. Yu, C. Wang, and K.-L. Ma. Parallel Hierarchical Visualization of Large Time-Varying 3D Vector Fields. In *Proceedings of Supercomputing 2007*, 2007.