Towards Exascale: High Performance Visualization and Analytics – Project Status Report

E. Wes Bethel, David Camp, Hank Childs, Mark Howison, Hari Krishnan, Burlen Loring, Jörg Meyer, Prabhat, Oliver Rübel, Daniela Ushizima, Gunther Weber

University of California, Berkeley, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, 94720

DOE 2012 Exascale Research Conference and Workshop
April 16–18, 2012
Portland, Oregon

ACKNOWLEDGMENT

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).

LEGAL DISCLAIMER

CONTENTS

Abstract—The LBNL Visualization research program focuses on a key set of problems facing our community as we move towards the exascale regime of computing. First, we aim to better understand how to effectively take advantage of an evolving architectural landscape where there are an increasing number of cores per processor by examining algorithmic reformulation and alternative methods of programming these increasingly complex platforms. Second, as the gap widens between computational and I/O capacity, it is increasingly imperative to perform more visual data exploration and analysis operations concurrent with the simulation to avoid increasingly expensive I/O. Our approach is to examine several interrelated issues aimed at minimizing the memory footprint, data movement, and design/implementation issues related to deploying tools that can have broad applicability across many different simulation codes and science domains. Third, increasing computational power produces larger and more complex scientific data, and our research program focuses on new technologies aimed at enabling scientific data understanding with ever-larger data. Here, we explore how multi-resolution data representation techniques can reduce data movement, and study novel approaches that focus visualization and analysis processing on subsets of data that are "scientifically interesting," thereby also minimizing data movement while accelerating scientific knowledge discovery.

Index Terms—hybrid parallelism, parallel volume rendering, parallel streamlines, performance optimization, auto-tuning, extended memory hierarchy

1 Introduction

Many in the community believe that making the transition from the petascale to the exascale regime of computing will require a complete rethinking of all aspects of the computational pipeline [4]. In that spirit, our research focuses on several key challenge areas that are somewhat unique to the fields of visualization, analysis, and analytics.

While the concern that MPI won't scale effectively into the exascale is not unique to visualization and analysis, proving that it is not effective, and potential remedial strategies, including using hybrid parallelism, which blends traditional message-based distributed memory parallelism with shared-memory parallel constructs, is the subject of Section 2. Specifically, we study hybrid parallelism as applied to two staple visualization algorithms, direct volume rendering (Section 2.1) and calculating integral curves (Section 2.2). In both instances, we see significant performance and resource-utilization advantages to using hybrid parallelism as compared to traditional distributed-memory parallelism.

Given that computational platforms are increasing in complexity, the algorithms that run on them are similarly increasingly complex. For example, determining the right granularity of work unit decomposition to achieve optimal performance is not a straightforward proposition. Furthermore, the difference in performance between good and bad choices can be significant. We explore this idea in Section 3 where we apply the principles of auto-tuning to optimize the performance of a staple visualization algorithm, direct volume rendering.

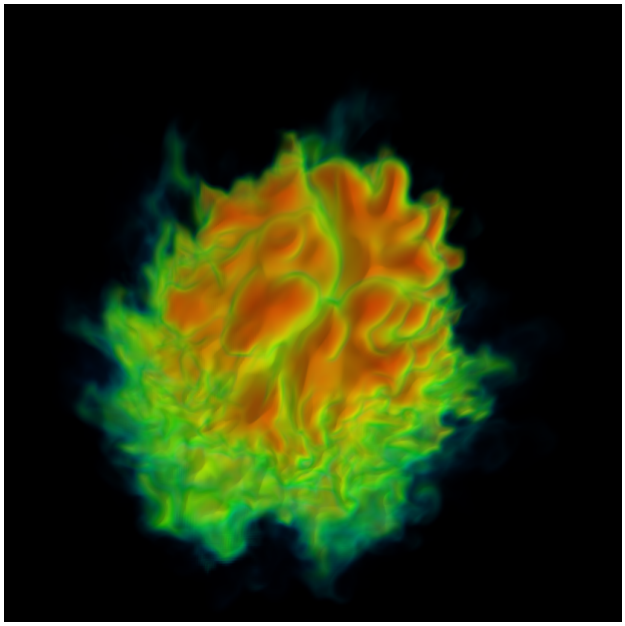

Fig. 1: Our hybrid-parallel volume rendering application produced

Another dimension of increasing platform complexity, specifically a deepening memory hierarchy, raises the question of how algorithms and applications can use alternative architectural configurations to their advantage. We explore this idea in Section 4 to better understand how using a locally attached drive, both traditional and solid-state, can improve performance. We find there are significant performance advantages, particularly for a data intensive integral curve calculation algorithm.

2 Hybrid Parallelism

2.1 Hybrid Parallelism and Extreme-Concurrency Volume Rendering

2.1.1 Introduction

Modern computational platforms are evolving towards using multi-core processors; future generations of machines will be built using processors containing tens or hundreds of cores. There is concern that existing, traditional message-based parallel programming models will not scale well on such platforms. The aim here is to better understand how well hybrid-parallelism, which combines both traditional message-based distributed memory parallel concepts with multi-core, shared-memory parallelism, performs for visualization algorithms, ray-casting volume rendering specifically, as compared to traditional message-based, distributed-memory parallelism.

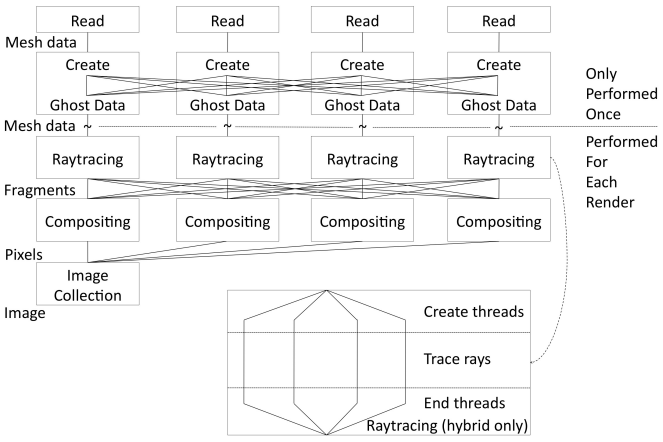

Fig. 2: Hybrid-parallel system architecture.

2.1.2 Implementation and Methodology

The architectural approach we pursue uses traditional MPI-based parallelism for across nodes, and one of a variety of different shared-memory parallel approaches (e.g., POSIX threads, OpenMP, CUDA) within a node. The architecture, shown in Figure 2, illustrates that the raycasting phase of the algorithm runs in shared-memory parallel mode in the hybrid parallel configuration.

Our solution entails conducting performance and scalability tests of traditional and hybrid parallel implementations of raycasting volume rendering, a staple visualization algorithm. Our approach is to compare performance using several different metrics: (1) absolute runtime, (2) memory footprint at various stages of algorithm execution; (3) communication characteristics of the two implementations.

Strong Scaling Study. A strong-scaling study aims to discover how time-to-solution improves as more and more processors are brought to bear on a fixed-size problem. The expectation is that as you bring more processors to bear on a problem, time-to-solution should decrease. The focus of our strong-scaling study was to discover how the relative time-to-solution decrease differs between MPI-only and MPI-hybrid approaches over a range of concurrencies.

In our tests, we used a fixed size dataset of 4608^3 over concurrency levels ranging from 1728 to 216,000-way parallel, the highest level of concurrency ever published (Figure 1). These tests, run on JaguarPF at ORNL, reveal that: (1) for just the initialization phase and before we begin any actual volume rendering work, the MPI-only implementation consumes 12x more memory than the MPI-hybrid implementation due to MPI overhead; (2) the MPI-hybrid implementation requires 40% less memory for ghost data at runtime owing to use of larger-sized data blocks; (3) the MPI-only implementation sends about 6x more communication messages during the compositing phase as compared to the MPI-hybrid implementation; (4) at the highest level of concurrency, the MPI-hybrid implementation is about 3x faster than the MPI-only implementation due primarily to the reduced communication load.

Weak Scaling Study. In contrast to the strong-scaling study, the weak-scaling study aims to increase problem size while adding more processors. The idea is that using more processors, and their associated memory, enables tackling ever-larger problems, which is important from a large-data perspective since we wish to use large, parallel computational resources to enable visual data exploration and analysis. Ideally, time-to-solution remains constant at increasing concurrency for a code that exhibits perfect weak scaling characteristics.

In our weak scaling studies, we use a 384^3 data block per processor at all concurrency levels. At 1728-way parallel, the resulting mesh is 4608^3, and at 216,000-way parallel, the mesh is 23040^3 in size (12.2 trillion cells). We observe that this mesh resolution far exceeds that of current scientific codes.

With raycasting volume rendering, there are actually two dimensions of "problem size." The first is the size of the source data itself. The second is the size of the resulting image. In order to better understand weak-scaling characteristics of our hybrid-parallel application, we explored two types of weak scaling: weak-dataset scaling and weak scaling. With weak-dataset scaling, we use a fixed-size image of 4608^2 pixels at all levels of concurrency. With weak scaling, we also increase the resolution of the image from 4608^2 pixels at 1728-way parallel up to 23040^2 pixels at 216,000-way parallel.

The results of the weak scaling studies confirm the favorable MPI-hybrid characteristics first observed in the strong-scaling studies: (1) at initialization, the MPI-hybrid implementation requires 12x less memory than the MPI-only implementation due to MPI overhead; (2) during the raycasting phase of the algorithm, the MPI-hybrid approach shows better scalability than the MPI-only approach; (3) during compositing, the MPI-hybrid approach requires substantially less communication traffic than the MPI-only approach; (4) the MPI-hybrid implementation consistently runs faster than MPI-only, as much as about 40% faster at 216,000-way parallel on the 12.2 trillion-cell dataset.

Many-core GPU Study. In an effort to better understand how these performance gains would extend to architectures having more than a few cores per processor, we conducted a test run on a multi-GPU system (Longhorn at TACC). Our testing configuration consisted of running an MPI-

hybrid implementation in parallel across 448 GPUs, each of which is capable of concurrently executing 30 CUDA thread blocks, producing an effective concurrency of 13,440-way parallel. The main objective here is to evaluate the relative performance of our MPI-hybrid implementation run a distributed memory GPU cluster as compared to a traditional distributed memory multi-core CPU system. The results indicate that the 448-GPU configuration runs about 30% faster than on a distributed memory multi-core system at similar concurrency.

2.1.3 Impact

The results of this study show that the hybrid-parallel approach offers clear and distinct performance advantages when compared to the traditional approach to parallelism. First, the hybrid-parallel version consumes between one sixth and one twelfth the amount of memory required by the traditional MPI version just for initialization. Second, at high levels of concurrency, the hybrid-parallel implementation runs about three times faster. Third, the hybrid-parallel version requires only about half the communication bandwidth compared to the MPI version.

These early results will help to shape the architecture of future visualization and analysis applications so as to be able to run effectively on current petascale and future exascale platforms: the hybrid-parallel approach allows for tackling larger problems on a given set of resources than is possible using an MPI-only approach.

This work resulted in an award-winning publication at the 2010 Eurographics Symposium on Parallel Graphics and Visualization [5], and an invited follow-on publication to IEEE Transactions on Visualization and Computer Graphics [6], and numerous invited presentations.

2.2 Hybrid Parallelism and Integral Curve Calculation

2.2.1 Introduction

Supercomputers are increasingly relying on nodes that contain multiple cores to achieve FLOP performance while minimizing power consumption. While current supercomputer nodes contain six to twenty-four cores, current trends suggest that future supercomputers will consist of individual nodes with tens to hundreds of cores.

This hardware approach gives rise to an important software question: which parallel programming model can effectively utilize such architectures? The classical method, which uses the Message Passing Interface (MPI), is to assign an MPI task to every core on every node; this approach is often the simplest way to write parallel programs. An increasingly popular approach, however, is to use hybrid parallelism, where fewer MPI tasks are used (typically one per node) and shared-memory parallelism is employed within a node. This approach, although more challenging to implement, can enable significant performance and efficiency gains.

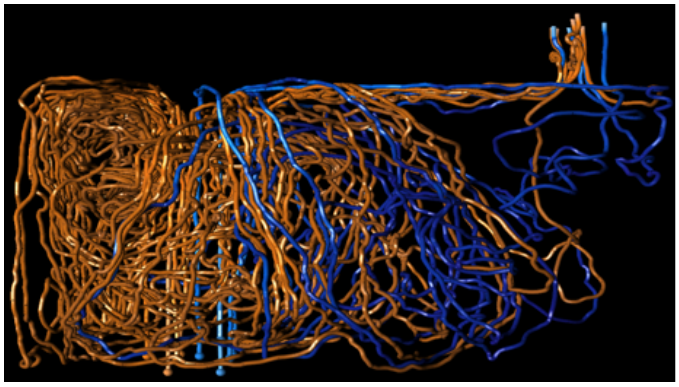

Fig. 3: Visualization of thermal hydraulics simulation output. Twin inlets pump water into a box, where the water from each inlet is of a different temperature. The water moves through the box and eventually exits through an outlet. The mixing behavior and temperature of the water at the outlet are the subjects of scientific study. Non-optimal mixing can be caused by long-lived recirculation zones that effectively isolate certain regions of the domain from heat exchange. This run of the NEK5000 code produces an unstructured grid comprised of 23 million hexahedral elements. Our team applied the hybrid-parallel streamline calculation to this data set, as well as two others, to study its performance characteristics. The hybrid-parallel streamline calculation has been implemented in production form in VisIt, and released to the scientific community.

In early work, we study the difference between a traditional MPI-based implementation and an MPI-hybrid parallel approach applied to the problem of streamline integration in a large vector field data set. That work showed a great improvement in performance on the local node. But this work did not show how the hybrid parallelism scaled. In this work we are looking at how it scales on a shared memory system and on large distributed memory parallel systems.

2.2.2 Implementation and Results

The aim of this work is to explore the scalability of our MPI-Hybrid parallel, distributed streamline algorithms. The hybrid parallel implementation is a blend of traditional message passing between CPUs and shared memory parallelism between cores on a CPU. We investigate the thesis that a hybrid parallel implementation can realize significant improvements in performance via factors like improved efficiency, reduced communication, and reduced I/O costs.

The problem of streamline integration should especially benefit from such an approach since its

runtime complexity and I/O vary greatly with respect to both the data set under investigation and the number and distribution of streamlines to be computed. Based on a wide range of experiments we perform for typical streamline computation scenarios, our findings, presented in Camp et al. 2011 [3], indicate that there is opportunity for significant performance gains under the hybrid approach, ranging from modest to over a 500% increase, depending upon dataset and problem configuration. These gains result from reductions in memory footprint, communication, I/O and improvements in parallel efficiency. Those experiments were conducted with the VisIt visualization tool, and hence, the performance observations we arrive at in that paper directly apply to real-world, production visualization scenarios.

2.2.3 Impact

In our work, we are looking at strong and weak scaling of our MPI-Hybrid parallel particle advection algorithms. We are looking at the limiting factors on scalability and performance. We will also try to optimize the performance on both a large-scale shared memory system and a large scale distributed memory parallel system. In the longer term, the results can help to inform the design and implementation of parallel particle advection algorithms to achieve better performance. Our published results show that the hybrid approach can produce a performance improvement ranging between two- and ten-fold, depending on various data dependent and data independent factors.

3 Autotuning and Performance Optimization for Extreme-Scale Visualization

3.1 Introduction

As computer architectures evolve towards the exascale – through a combination of increased core-count per chip, and deeper and more complex memory hierarchies – it is difficult to achieve optimal algorithmic performance due to increasing architectural complexity. There are many open research questions as we move towards the exascale, including understanding tunable algorithmic parameters and algorithmic optimizations impact performance on evolving computational architectures.

3.2 Implementation and Results

Our approach is to leverage auto-tuning work that has been applied by the computational science research community to numerical solver kernels. In our research, we vary tunable algorithmic settings, along with known algorithmic optimizations and two different memory layouts, and measure performance in terms of absolute runtime, along with L2 memory cache misses.

Our work is a more systematic and thorough study than has ever been performed for a staple visualization algorithm, and the algorithmic optimizations, which have appeared in literature over the years, have not been thoroughly tested on modern processor architectures in conjunction with other tunable algorithmic parameters. Our results, which will appear in Bethel and Howison 2012 [1], indicate there is a wide variation in performance on all platforms, as much as 254% for the tunable parameters we test on multi-core CPUs and 265% on many-core GPUs, and the optimal configurations vary across platforms, often in a non-obvious way.

For example, our results (Figure 4) indicate the optimal configurations on the GPU occur at a crossover point between those that maintain good cache utilization and those that saturate computational throughput. This result is likely to be extremely difficult to predict with an empirical performance model for this particular algorithm because it has an unstructured memory access pattern that varies locally for individual rays and globally for the selected viewpoint.

3.3 Impact

Our research shows that high performance visualization algorithms, and related data-intensive cousins, stand to reap huge performance gains through careful tuning that has worked well in the computational science community. For example, the set of experiment runs in our work in hybrid-parallel volume rendering work (Section 2.1) consumed on the order of 5M CPU hours. Had we not first conducted the performance optimization and autotuning work described in this chapter, the estimate runtime of the untuned code would have been on the order of 15M CPU hours.

Some of the our study's results are unexpected, namely those for the GPU: absolute runtime and memory utilization performance are a function of both blocking factor and device-specific behavior (thread divergence) that is unique to this type of processor and architecture. These benefits will likely become more pronounced in the future as the number of cores per chip and the cost of moving data through the memory hierarchy both increase. Our results presented in the paper raise several new interesting research questions for follow-on work, including use of some form of auto-tuning as part of routine initialization to enable platform-specific optimizations on the part of data-intensive applications.

		Array Order					Z Order				
		1	2	4	8	16	1	2	4	8	16
NoERT	1	3.21	1.85	1.23	1.25	1.49	3.32	1.87	1.17	0.83	0.74
	2	1.94	1.18	0.96	1.21	1.55	1.88	1.12	0.75	0.61	0.69
	4	1.29	0.93	0.93	1.31	1.96	1.18	0.75	0.58	0.62	0.93
	8	1.15	0.99	1.03	1.71	1.75	0.84	0.62	0.63	0.86	0.91
	16	1.31	1.18	1.39	1.42	1.43	0.76	0.71	0.86	0.84	0.89
ERT	1	2.95	1.70	1.12	1.11	1.31	3.06	1.72	1.07	0.75	0.66
	2	1.78	1.08	0.86	1.06	1.35	1.72	1.02	0.68	0.55	0.60
	4	1.18	0.84	0.83	1.14	1.70	1.08	0.68	0.52	0.55	0.81
	8	1.03	0.88	0.92	1.47	1.51	0.76	0.56	0.55	0.74	0.78
	16	1.16	1.04	1.21	1.24	1.25	0.68	0.62	0.74	0.72	0.77

		Array Order					Z Order				
		1	2	4	8	16	1	2	4	8	16
NoERT	1	1E+08	1E+08	1E+08	3E+08	8E+08	7E+07	7E+07	8E+07	1E+08	3E+08
	2	1E+08	1E+08	2E+08	6E+08	9E+08	6E+07	6E+07	7E+07	2E+08	5E+08
	4	2E+08	2E+08	6E+08	8E+08	1E+09	6E+07	6E+07	1E+08	4E+08	1E+09
	8	5E+08	7E+08	9E+08	1E+09	1E+09	9E+07	2E+08	4E+08	9E+08	9E+08
	16	1E+09	1E+09	1E+09	1E+09	1E+09	3E+08	5E+08	9E+08	8E+08	9E+08
ERT	1	8E+07	9E+07	1E+08	3E+08	7E+08	7E+07	7E+07	7E+07	9E+07	3E+08
	2	9E+07	9E+07	2E+08	5E+08	8E+08	5E+07	5E+07	6E+07	2E+08	4E+08
	4	1E+08	2E+08	5E+08	7E+08	1E+09	5E+07	6E+07	1E+08	3E+08	8E+08
	8	4E+08	6E+08	7E+08	1E+09	1E+09	7E+07	2E+08	3E+08	7E+08	8E+08
	16	1E+09	9E+08	1E+09	1E+09	1E+09	2E+08	4E+08	7E+08	7E+08	7E+08

Fig. 4: Runtime (a) and L2 cache misses (b) averaged over 10 views for different thread block sizes on the NVIDIA/Fermi with varying memory layout and early ray termination (ERT). Grey boxes indicate thread blocks with too few threads to fill a warp of execution. The unexpected result here is that better runtime performance does not necessarily correspond to better memory utilization on this particular platform. This result would likely have not been possible to discover using a predictive performance model, and reinforces the argument that finding optimal performance on increasingly complex architectures often times requires a search of tunable algorithm parameter space and algorithmic optimizations.

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

Emerging architectures: can Solid State Drives accelerate visualization and analysis? We have conducted a study aimed at gaining a better understanding of how locally attached solid state drives (SSD's) can accelerate visualization and analysis operations on large problems and on large, distributed-memory parallel machines. The initial findings, reported in Camp et al. 2011 [2], indicate there is the potential for a performance gain ranging modest to approximately a two-fold speedup, depending upon problem configuration.

4.1 Introduction

Computer architects have been experimenting with a new approach to improve the performance of I/O through the use of relatively lower cost components. The approach uses two-stage I/O system: a remote parallel file system coupled with a local solid state drives (SSDs) and/or hard disk drives (HDDs). In a conventional I/O subsystem, the simulation writes data directly to a parallel file system. With the new design, the simulation writes its data to the local drive instead. Then the simulation may immediately resume computation, while concurrently and transparently to the simulation, data is migrated from the local drive to the parallel file system, thereby insulating the simulation performance from potentially much slower parallel file system. Although the local drives introduce a new dollar cost, they lessen the importance of peak I/O bandwidth, allowing for the SSDs to be coupled with a slower (and potentially less expensive) parallel file system, resulting in an overall cost reduction. To applications, this new I/O configuration appears to have two distinct bandwidth characteristics. On write, the bandwidth appears to be good, since it is be accelerated by local drive. On read, however, the bandwidth can appear poor, since the reads are backed by a slower parallel file system and the presence of the local drive cannot directly accelerate this activity. As I/O is often the slowest part of visualization and analysis pipelines, suboptimal I/O read performance will result in poor overall visualization performance. Our research question is to better understand if visualization algorithms can be re-architected to use new I/O access patterns that leverage the local drives and actually improve I/O and visualization performance.

4.2 Implementation and Methodology

We investigate whether local SSDs or HDDs 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 because the data will be located on the remote parallel file system, the local SSDs or HDDs can be

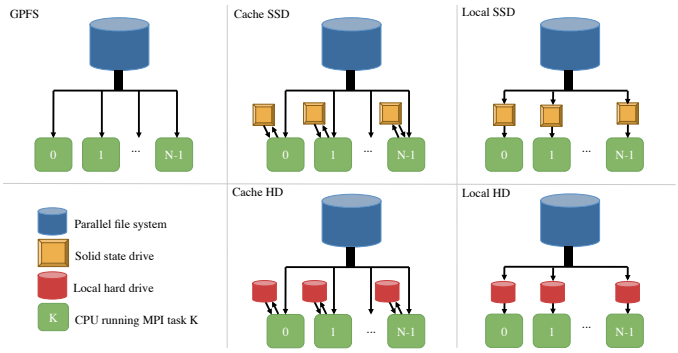

Fig. 5: Test configurations.

used as a cache to store those blocks, considerably accelerating subsequent reads. We also study how these local drives can accelerate I/O and visualization performance. Although many paradigms for processing data do not read blocks of data repeatedly, streamline calculations do. 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. The different test configurations, shown in Figure 5, are detailed below.

In the first variant, which we denote *GPFS*, each MPI task loads data blocks directly from the general parallel file system (GPFS), establishing a baseline for performance without an extended memory hierarchy. In the second variant, *Cache SSD*, each MPI task can load data blocks from either the SSD or from the GPFS. For each load, it starts by checking the SSD, since its load times are considerably faster. If the SSD does not contain the data, then it loads the block from the GPFS and stores the data block back to the SSD, meaning subsequent loads of that 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 GPFS. 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.

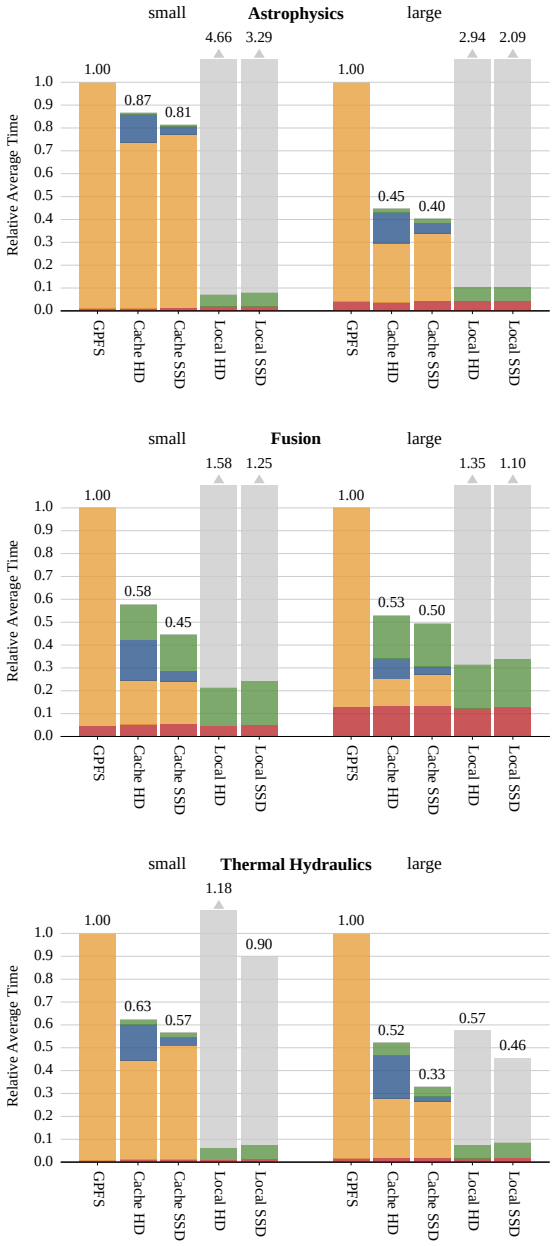

Fig. 6: Using the local storage as a cache (Cache HD, Cache SSD) results in up to about a two-fold improvement in performance compared to baseline performance (GPFS). When the entire dataset can fit into local storage, performance is better yet (Local HD, Local SSD) but requires an expensive one-time data copy (gray).

4.3 Results

We examine five different I/O configurations (see Figure 5). The test results, which measure performance of the algorithm in different configurations, is shown in Figure 6. In Figure 6, baseline performance (GPFS), always going to the filesystem with no use of SSDs or local drives, is always 1.0, and other configurations are scaled to that baseline performance. Values less than 1.0 are faster than baseline performance. These results show that using the SSD or local drive as a local cache speeds performance in all problem configurations. This result is due to the fact that data reloads can be serviced by faster, local storage rather than slower, more distant storage. For problems that fit entirely within local storage (Local HD and Local SSD), these configurations are much faster than the GPFS configuration, but require an expensive one-time data copy, which shows in these charts as gray. The results of this study showed that using a local drive to extend the memory hierarchy created an overall speedup of approximately a factor of two. We developed a equation for a performance model to predict performance when using locally attached storage. This study illuminates possible changes in I/O access patterns that will lead to improved performance on the I/O subsystems predicted to exist on future exascale machines.

REFERENCES

[1] E. Wes Bethel and Mark Howison. Multi-core and Many-core Shared-memory Parallel Raycasting Volume Rendering Optimization and Tuning. *International Journal of High Performance Computing Applications*, (In press), 2012.

[2] David Camp, Hank Childs, Amit Chourasia, Christoph Garth, and Ken Joy. Evaluating the Benefits of an Extended Memory Hierarchy for Parallel Streamline Algorithms. In *LDAV: Large Data Analysis and Visualization symposium at IEEE Visualization 2011*, Providence, RI, USA, October 2011.

[3] David Camp, Christoph Garth, Hank Childs, Dave Pugmire, and Kenneth I. Joy. Streamline Integration using MPI-Hybrid Parallelism on a Large Multi-Core Architecture. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1702–1713, 2011.

[4] Sonia Sachs (ed.). Tools for Exascale Computing: Challenges and Strategies – Report of the 2011 ASCR Exascale Tools Workshop. Technical report, U. S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Bethesda, MD, USA, October 2011.

[5] Mark Howison, E. Wes Bethel, and Hank Childs. MPI-hybrid Parallelism for Volume Rendering on Large, Multi-core Systems. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, Norrköping, Sweden, May 2010. LBNL-3297E, Best Paper Award.

[6] Mark Howison, E. Wes Bethel, and Hank Childs. Hybrid Parallelism for Volume Rendering on Large, Multi, and Many-core Systems. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):17–29, January 2012. LBNL-4370E.