Survey on Heterogeneous Scientific Visualization Algorithms and their Performance

Kristi Belcher

University of Oregon

Abstract- **Heterogeneous computing is increasing in popularity among researchers as the trend toward large data continues. Research scientists have many options when it comes to which architecture to run their codes on. Determining the optimal architecture to run their algorithms on is a critical consideration. Sometimes, it may be best to run an algorithm on a combination of more than one architecture. In particular, Scientific Visualization researchers are faced with the need for more compute power to process their simulation codes with better performance. In this survey, I review several popular heterogeneous implementations and investigate the important trade-offs that must be factored in when deciding to port an algorithm to a heterogeneous implementation. The scope of the survey is limited to the field of Scientific Visualization. Additionally, the main contribution is a literature review of the implementations, advantages, and challenges of heterogeneous computing. The results of this study will help inform the Scientific Visualization community about those architectures that give optimal performance at certain stages of the simulation run.**

I. Introduction

Scientific Visualization is the study of how to accurately represent and understand data from scientific simulations. One particular field within Scientific Visualization focuses on Flow Visualization. This field is dedicated to visualizing how massless particles move through some vector field. For example, through flow analysis, we can understand how air molecules react and move when a high velocity jet flies through them. In particular, Particle Advection is an example of a common flow visualization algorithm. Particle Advection is a fundamental scientific visualization algorithm that calculates the displacement of particles along a velocity field. This algorithm is a key element of flow analysis. Typically, particle advection handles very large amounts of input data in order to obtain an accurate result. Although Particle Advection is such an important calculation for vector field visualization, implementing an efficient large-scale parallel Particle Advection computation remains a challenge. Moreover, particle trajectories can vary greatly depending on different parameters set for the simulation. For example, termination criteria - those rules that dictate when a particle stops moving through the vector field - is user defined and thus can be very different from one run to the next. These rules may involve particles collecting in one particular spot in the field, going out of the global domain, or reaching the maximum allowed advection steps. Additionally, as simulations require more and more data to produce meaningful results, visualization and analysis is being performed in situ. This means that data is generated and visualizations are performed as the simulation is running, even using the same resources. Running large-scale Particle Advection simulations on modern supercomputers has become a requirement for efficient results in a reasonable amount of time.

Modern supercomputers have varied computational capabilities that make tailoring algorithms to those architectures a priority. The supercomputers that researchers must work with have widely differing architectures that range from modest computational power to very high computational power, with several architectures that lie in between. Because of this trend, it is important to take into consideration the kind of architecture that a simulation code will be run on. Sometimes, however, it is not as simple as just picking one of these kinds of architectures for a specific code. In order to more effectively harness the computation power available on a supercomputer, many research scientists are utilizing several different kinds of architectures at one time, during the same run of a simulation code. Using more than one kind of architecture to compute data and solve a problem is referred to as heterogeneous, or hybrid, computation. Heterogeneous computation requires that researchers tailor their codes to two (or more in some cases) architectures at a time. The result of utilizing heterogeneous computations vary depending upon several factors such as problem size, algorithm, and architecture [KK11]. In order to obtain the kind of performance results desired by domain scientists, these factors must be understood and optimally tuned. Previous studies [MV15] have been conducted to better understand heterogeneous computing and those techniques which can give algorithms optimal performance. This survey draws upon previous studies to create a deeper understanding by focusing on heterogeneous computing in the field of Scientific Visualization. The purpose of this survey is in fact to provide a detailed summary of current heterogeneous computing research, and specifically how it is used for Scientific Visualization.

The scope of this survey is limited to scientific visualization with an emphasis on flow visualization. The goals of this survey are listed below.

- To summarize why heterogeneous parallel algorithms are needed and discuss the trend toward large-scale simulations.

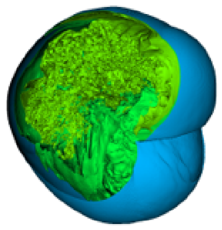

Fig. 1. A picture of both the Astrophysics data set simulation.

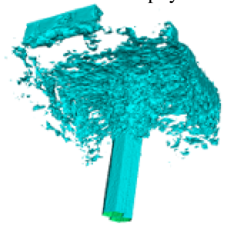

Fig. 2. A picture of both the Hydraulics data set simulation.

- To review previous works that have compared different architectures in a large data setting.
- To analyze different works which have studied various hybrid parallelization techniques in a flow visualization context.
- To make conclusions given the information discussed and provide next steps for future research.

II. Motivation and Background

To provide some motivation for why exploration into heterogeneous Computing is worthwhile, a Top 5 supercomputer (as of 2019 [Jac19]) is introduced and a couple of motivating examples are provided. Supporting the trend toward big data, one of the Top 5 supercomputers, Summit, at Oak Ridge National Laboratory (ORNL) was built with about 4600 compute nodes, each with IBM POWER9 processors and 6 NVIDIA Volta V100 GPUs[sum19]. Like the other Top 5 supercomputers, this top-of-the-line machine was created to process large amounts of data at ever-increasing speeds because of the need for more compute capability.

These supercomputing trends are fueled by scientific research that uses increasingly large amounts of data to get scalable and accurate results that are used to answer important questions. For example, large amounts of data are used to simulate the magnetic field surrounding a solar core collapse which results in a supernova to answer pressing questions in Astrophysics [ECBM08]. Figure 1 shows the visualization of such a simulation. Additionally, another simulation of interest is a thermal hydraulics data set that demonstrates what happens when two equally sized tubes pump water into a box with a temperature difference between the water inserted by each tube; eventually the water exits through an outlet. The mixing behavior and the temperature of the water at the outlet of the box are of interest because it gives insight into the impact of unequal heat exchange resulting in non-optimal mixing [PJDA08]. Figure 2 shows the resulting visualizations from these simulations. Because of these trends, research scientists are studying the performance impact of using accelerators on Supercomputers to do these simulations. When only using accelerators like the GPU are not enough, these researchers turn toward heterogeneous computing in order to utilize more of the system resources available.

The motivation behind this is centered around the idea that while the accelerator is busy doing computation, the CPUs available could also be doing a portion of the total work. Additionally, sometimes the current workload of the simulation is better for the accelerator (i.e. when it can be fully loaded with plenty of data parallel work), while other times the CPU may be better suited (i.e. when there is just a few computations left to do). Perhaps the most important reason why heterogeneous computation is worth exploring in the context of Scientific Visualization is for improving interactive visualization where even just one or two saved seconds really counts. With this in mind, researchers look toward heterogeneous computing to achieve better performance and resource utilization in their scientific visualization simulations.

III. Research Review

In this section, a wide range of hybrid, or heterogeneous, computation research is discussed and reviewed. First, the overall groundwork is laid out with various previous works studying different workloads that are best suited for a target architecture. Then, several hybrid implementations are explained including MPI-Hybrid works, hardware-agnostic works, and those previous works that lie somewhere in between. Next, challenges to hybrid computing are discussed. Lastly, the discussed body of research is summarized and concluded.

A. Architecture Study

Camp et al. [CKP*13] did an extensive study where various workloads were tested on both the CPU and the accelerator (in this case, a GPU). The study was done exclusively with a Particle Advection algorithm where they varied the number of particles computed and the number of steps for each particle. In the study, they show what kinds of workloads result in better CPU performance and which kinds resulted in better GPU performance. They conclude that in those workloads with very large particle counts, the GPU had much better performance. As expected, they also conclude that those workloads with very small particle counts resulted in better performance with the CPU. However, for workloads with a medium-amount of particles, the step size really matters when determining if the CPU or GPU would get better performance. The key to determining which targeted architecture would give better performance is whether or not the accelerator could be fully loaded. If the workload could take advantage of the GPU's bandwidth and computational power, then the GPU would give better results. Accordingly, if the workload could better take advantage of the CPU's latency, then the CPU would give better results. For example, if there were 1000 particles that needed to be advected for 5000 steps, then the model from Camp et al. predicted that the CPU would be better for that workload. If there were 5000 particles that need to be advected for 1000 steps, then the model predicted that the GPU would be better suited for this workload. These results inspire how a heterogeneous implementation of particle advection could be done which would bring both architectures together to work on the problem.

Other works also define the workloads that are best suited for a target architecture. Wu et al. [WTYH18] describe how they tuned their fluid simulation algorithm for the GPU. They go in to detail on how they constructed sparse grids to promote fast incremental updates during computation. This technique was designed to take full advantage of the GPU's hardware in an effort to hide latency. For example, since their fluid simulation algorithm requires neighborhood stencil look-ups, they created a technique that rebuilds the stencil while efficiently utilizing thread blocks and data coherence in order to keep the entire computation on the GPU. The efficient GPU memory management described in their paper removes the need for extra memory transfers, keeping the algorithm GPU-friendly.

Additionally, Chen et al. [CSH16] also describe how they reimplemented a Particle Advection algorithm to better perform on the GPU by taking advantage of asynchronous computation. They also describe how they localized memory accesses on the GPU to overcome cache misses. Furthermore, they described their novel redesign of the Runge-Kutta computation, the main Particle Advection computation that calculates how a particle moves from one location to the next, by splitting it into multiple stages to reduce register utilization. Overall, this previous work lays the groundwork for determining how to best port a single-target architecture implementation to a hybrid one by understanding the optimal workloads for a particular architecture and describing different code optimizations that make code better suited for a particular device.

B. Distributed Memory Heterogeneous Computation

MPI is the Message Passing Interface that enables distributed memory parallel computing. Many researchers have used MPI to obtain the benefits of heterogeneous computing on large parallel systems like supercomputers. By exploring these previous works, it becomes apparent that utilizing distributed memory heterogeneous computation enables very large datasets to be processed, especially those datasets which were previously too large to fit into system memory.

First, to understand workloads suited for distributed memory computations, Camp et al. [CKP*13] also studied those workloads which favored a GPU distributed memory environment compared to those workloads that favored a CPU distributed memory environment. This work is in addition to the work previously described by Camp et al. in the section above. With this study, researchers can better understand the kind of workload characteristics needed in order for good performance. However, note that the scope of this work did not include a study of workloads that favor a heterogeneous distributed memory environment. The experiments done by Camp et al. were either processed by only the GPU or only the CPU (In the GPU case, the CPU was only used for memory transfers, etc. The CPU was not used for any of the computations). Even still, this work laid the foundation for understanding what kinds of workload characteristics are best for CPU or GPU distributed memory environments.

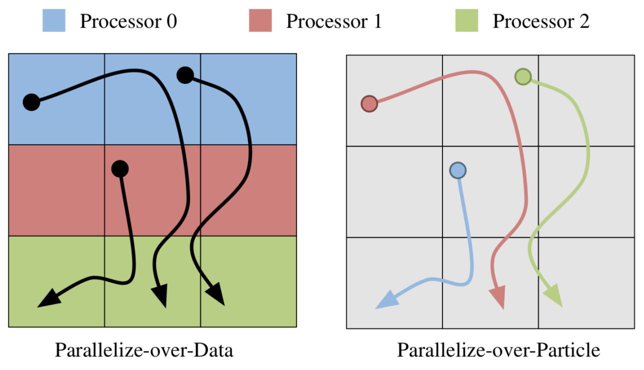

Fig. 3. An example of two parallelization techniques for Stream Surface Computation

Next, a different work by Camp et al. [CCG*12] studied two different types of parallelization techniques for a Stream Surface Computation algorithm. Stream Surface Computation is another very common Flow Visualization algorithm which typical is given very large datasets to process. Thus, Camp et al. studied the performance of two parallelization techniques when implemented in a MPI Parallel environment. Like Particle Advection, the Stream Surface Computation calculates on how particles move through a space. Hence, the two parallelization techniques included *parallelize-over-particles* where each MPI task is assigned a fixed number of particles and performs all the computations related to those particles until they eventually terminate. The other parallelization technique was *parallelize-over-data* where each MPI task is assigned a piece of the domain, and only performs computations on particles that exist within that sub-domain.

Figure 3 gives a visual representation of these two techniques. Camp et al. found that the *parallelize-over-particles* method had the advantage of better load-balance since each MPI task was assigned an equal number of particles, but the disadvantages included the I/O cost since communication of where particles move was done as needed. Moreover, the I/O cost can vary greatly depending on the seeding conditions (where the particles are initially placed) and conditions of the vector field (the values that dictate where the particles go). On the other hand, *parallelize-over-data* had the advantage of performing minimal I/O between MPI tasks, but could be severely impacted by load-imbalance between tasks. Again, the load-imbalance depends on the data layout and is very problem-dependent. Camp et al. summarized their work by pointing out that the biggest factor to distributed memory performance is load-imbalance. They also mentioned further study into caching strategies to help with data transfers between tasks. Overall, by studying the performance advantages and disadvantages of these two implementations, researchers can better understand what kinds of parallelization strategies perform best in a distributed memory environment.

Similarly, Pugmire et al. [PCG*09] also studied Streamline Computations of very large datasets and how well this algorithm scaled. They compare the previous two parallelization strategy to a novel hybrid strategy which aims to achieve optimal load balance while also minimizing I/O cost with overall good scalability. In their MPI-hybrid implementation, they attempt to alleviate load-imbalance by dynamically assigning streamlines and data blocks (i.e. sub-domains) to processors. Their Master-Slave algorithm attempts to keep processors busy as much as possible by communicating streamlines when needed, and performing computations otherwise. In other words, the Master assigns the work as it becomes available and keeps track of progress, while the Slave receives the work and performs necessary computations. As both the Master and the Slave work simultaneously, the Master is making sure that each Slave has roughly equal amounts of work, modifying the work queue for the Slaves accordingly. More details on this complex algorithm is provided in the paper. Overall, this study shows a load-balanced MPI-Hybrid implementation that uses Master-Slave processor parallelization technique to solve the problem. The algorithm introduced in this work leverages both MPI processes and the Master-Slave technique to harness the Hybrid performance advantage.

Additionally, in a different work, Camp et al. [CGC*11] study the performance between an MPI-only implementation and a MPI-Hybrid implementation. This work provides insight in to how and under what circumstances MPI-Hybrid algorithms can provide performance benefits. The difference between this work and the previously described work by Pugmire et al. is that for this MPI-Hybrid implementation, the algorithm uses both MPI and OpenMP so that both distributed memory and shared memory environments are used. Additionally, this work implemented a *parallelize-over-particles* and *parallelize-over-data* MPI-only and MPI-Hybrid implementation. Camp et al. found that both implementations performed better in the MPI-Hybrid setting. In fact, the *parallelize-over-particles* MPI-Hybrid implementation benefited most from improved caching. The improved caching ability lead to better I/O performance. This implementation also had better distribution of faster versus slower streamlines to calculate, thus leading to better load balancing. Moreover, the *parallelize-over-data* MPI-Hybrid implementation benefited most from increased parallel efficiency due to better distribution of the data assigned to MPI tasks. This implementation also benefited from lower communication costs since the OpenMP threads could share data within the data block. Overall, this work illustrates the performance benefits of using MPI-Hybrid approaches.

At the same time, key performance factors like load balancing and data distribution are shown to play a large role in overall performance. This will be explained further in future sections.

C. *Hardware-Agnostic frameworks*

In the field of Scientific Visualization, among others, it became apparent that with large data trends and the need for better performance, multiple architectures would be needed. While some algorithms would be better suited for some kind of CPU parallelization strategy like OpenMP, others might be best suited for an accelerator like the GPU. Still others may be best suited for a distributed network using MPI. Then again, with concerns of portability, Visualization researchers

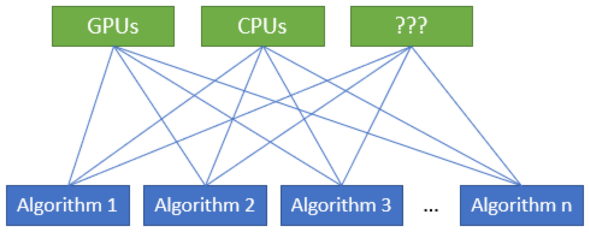

Fig. 4. A picture of how each algorithm must be ported to each architecture.

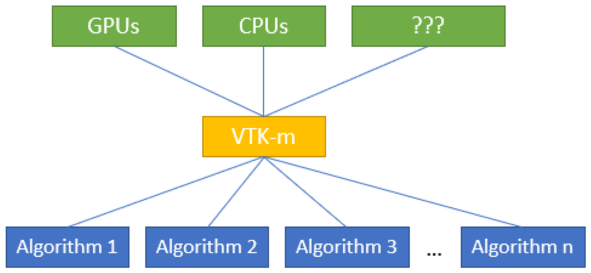

Fig. 5. A picture of how with VTKm, each algorithm only has to be mapped once to VTKm.

worried that there would be a future architecture that would be best for a particular algorithm. For example, Summit currently has NVIDIA GPUs which run with CUDA, yet Frontier, the next generation computer, will have AMD GPUs which do not run on CUDA. Researchers realized that one possible solution would be to port every existing algorithm to several different architectures to test which is best. In an effort not to undergo that kind of extensive development which may or may not pay off, a different solution, a hardware-agnostic solution, was created. To help ease the transition of efficiently porting algorithms to multiple kinds of hardware, several frameworks have been created to provide a hardware agnostic environment that acts as a bridge between several different platforms. These frameworks are described below.

1) VTK-m Framework: VTK-m [MSU*16] is an extension of VTK, the Visualization Tool-Kit, with the addition of several many-core (hence the 'm' in VTK-m) implementations for common visualization algorithms. In short, VTK-m was created to provide the same functionalities as VTK, but with the inclusion of achieving portable performance across a variety of many-core and multi-core architectures.

Figure 4 and 5 shows a visual representation of how VTK-m can be used. Note that in (a) a developer must implement several versions of the application code: one for each desired hardware platform. In (b), the developer only has to implement one VTK-m program to be able to utilize any compatible architecture. VTK-m creates a hardware agnostic environment using Data Parallel Primitives (DPPs). In 1990, Blelloch [Ble90] proposed a model where specific operations could be carried out on vectors of size N in O(log(N) time in the worst case. Blelloch describes DPPs such as Scan, Scatter, Map, Reduce, and Gather used in libraries like NVIDIA's Thrust library. Since The Thrust library uses DPPs, it can be compiled to work across a variety of parallel architectures.

Additionally, DPPs are also used as the basis for different performance-portable libraries like VTK-m. Using DPPs for performance portability requires that developers re-think their algorithms, which can be a difficult task. This remains one of the more difficult hurdles in the process of rewriting algorithms in VTK-m.

VTK-m is a hardware agnostic solution for implementing key visualization algorithms across a wide range of platforms. As modern supercomputers are equipped with a wider range of hardware varieties, from programmable graphics processors (GPUs) and many-core co-processors (Intel Xeon Phi) to large multi-core CPUs, a growing concern for Visualization software developers remains providing optimized software that can employ many different algorithms. As such, VTK-m helps alleviate this problem, providing software developers with the ability to write a single implementation of their code and have it run on a variety of architectures obtaining excellent performance on each distinct platform. VTK-m supports general portability across a wide variety of architectures, thus making portable performance possible.

Works such as Pugmire et al. [PYK*18] use VTK-m to parallelize Particle Advection in order to demonstrate performance portability across various architectures. This work shows that VTK-m is a viable solution because it offers a performant solution that, once parallelized with VTK-m, takes little to no code modification in order to be efficiently ported to a different architecture. Other works from Childs et al. [CBP*14], Lessley et al. [LBMC16], and Larsen et al. [LLN*15] all show that by using VTK-m, they can achieve portable performance in a hardware-agnostic setting. Childs et al. [CBP*14] compared distributed-memory GPU clusters to distributed-memory CPU clusters in order to demonstrate heterogeneous performance portability with Particle Advection. This work provides insight on the types of workloads that are best suited for a target architecture in a distributed memory (and thus heterogeneous) setting. Childs et al. lays the foundation for future work (see section below) to combine a distributed memory environment (provided by MPI) with shared memory (std threads) and an accelerator (the GPU). Both Lessley et al. and Larsen et al. show that VTK-m is a viable solution for portable performance with large datasets. In the field of Scientific Visualization, research has shown that VTK-m is a viable solution for porting algorithms easily from one architecture to the next.

2) Kokkos and RAJA Frameworks: The Kokkos [ET18] and RAJA [HB18] frameworks are also options for hardware-agnostic computation. Although these frameworks are worth mentioning because they offer a performance-portable, hardware-agnostic environment, they are mostly outside the scope of this survey because they are not specific to the Scientific Visualization field. However, when using these frameworks, a developer can implement one Kokkos or RAJA program which can be targeted for any supported architecture, similar to VTK-m development.

D. Challenges for Heterogeneous Computation

From studying and reviewing the related works surrounding heterogeneous computation, there are a number of factors that may limit the performance benefit of Hybrid execution. The purpose of this section is to list those conditions and give some explanation behind each one.

- **Make sure there is sufficient data:** In order for a heterogeneous algorithm to perform well, there must be enough computation available. The more hardware utilized the more parallelism that needs to be exposed and exploited from the data. If an algorithm doesn't have enough work, then there may not be any performance benefit and therefore the entire effort is just wasted.
- **Make sure the architecture is compatible:** Some algorithms are not suitable for certain architectures and thus should be avoided on that particular device. For example, recursion has been shown to not be performant on GPU architectures [KHN12]. Therefore, if a particular algorithm uses recursion, it would be unwise to port that algorithm to a heterogeneous environment where GPUs are involved. The exception to this criteria being if the programmer can cleverly remove the problematic code (in this case, the recursion) in the algorithm before porting it to the target architecture.
- **Make sure performance gain is worth programmer effort:** If the time and effort involved in creating an efficient heterogeneous implementation for a particular algorithm outweighs the performance benefit, then it may be the case that a different parallelization strategy should be used instead. This is largely problem-dependent, but programmers should make sure that the time and cost of implementing a heterogeneous version of the algorithm will pay off by considering factors like available parallelism, hardware resources, etc.
- **Make sure the other factors mentioned in previous works are weighed in**: Previous research mentioned parallelization strategy (i.e. *parallelize-over-particles* versus *parallelize-over-data*), load imbalance, communication, and idle time as factors that should be considered when implementing a heterogeneous algorithm. On the other hand, there are different approaches that use Information Gain and Entropy to predict blocks that will have more data versus those that will have few to no particles. Marsaglia et. al. [MLB*19] created an approach to calculate those blocks that contained more important data, and then used that information to assign blocks to MPI ranks. Müller et. al. [MCHG13] proposed a *work-requesting* scheduling scheme which alleviated much of the load imbalance problem of POB. In their approach, blocks are assigned to nodes and when one node runs out of work to do on its assigned block, it sends a message to a neighboring block requesting half of its remaining work. Previous work from researchers such as Pugmire et al. [PYK*18], Binyahib et al. [BPL*19], and Childs et al. [CBP*14] all describe how they overcame these kinds of setbacks. Overall, these considerations are important for ensuring that the heterogeneous version will perform well

when multiple architectures are used together.

Heterogeneous computation will inevitably have challenges that programmers must overcome. By studying ways to work around these challenges, mitigate them, or otherwise resolve those setbacks, programmers will be able to implement heterogeneous algorithms that can provide more optimal performance benefits.

E. My Research

My current research builds off of the work described in this survey. For my study, a VTK-m Particle Advection algorithm uses a *parallelization-over-data* approach with MPI. In this algorithm, the domain is divided into N pieces in which each of the N MPI tasks is assigned to a piece of the total problem. A particular MPI task will calculate particle trajectories on its piece of the domain until its assigned particles terminate or advect outside of its piece of the domain. Here, a MPI task will fork three shared memory std threads: one manager thread for communication and for keeping track of overall progress and two worker threads to advect particles given to it by the manager. If a particle does travel beyond a MPI task's domain to another piece, it is communicated (by the manager thread) to whatever MPI task owns that piece of the domain. This process repeats until all particles have terminated. Additionally, using a VTK-m feature, one worker thread is dedicated to the CPU while the other is dedicated to the GPU. In this way, the worker thread is only able to advect particles on that specific hardware.

The main idea is that the manager will determine when to use the accelerator (in this case a NVIDIA GPU) and when to use the CPU to process the remaining work, adding that work to the appropriate worker thread's queue. The manage thread does this by asking an "oracle" if the current workload is better suited for a GPU or a CPU. The oracle acts like a black box, telling the manager to use the GPU when appropriate and then telling the manager to switch to the CPU when the workload can no longer take advantage of the GPU's hardware. As before, the workers are relatively simplistic, only advecting particles as work is sent to them. The manager thread simply needs to give the appropriate work to the appropriate worker.

Next, my algorithm's performance is compared to a CPU-only and GPU-only version. Although my research is still in progress, I generally see that my algorithm's runtimes can be summarized as $r \approx \min i(T_c, T_g)$, where r is the runtime for my algorithm, T_c is the runtime for the CPU-only version, and T_g is the runtime for the GPU-only version. In some cases, I see that my algorithm is faster than both of the other versions, but I am still trying to understand why and the conditions in which this most likely will happen. When I have completed my experiments, I will create some kind of model to represent the performance of my algorithm so that the results of my work can be used more broadly by researchers in the Scientific Visualization field.

The goal of my work is to study and model how a heterogeneous environment like the one described above can benefit performance and in what types of circumstances. From results from previous works, it is important that great care go in to creating an oracle that picks a device depending upon the characteristics of the current workload. In effect, our work explores the fundamental research question: **If faced with a Particle Advection problem, can a heterogeneous implementation perform better than an implementation that runs the entire workload on either the CPU or the GPU?**

IV. Conclusions and Future Work

This survey discussed the current body of work surrounding heterogeneous scientific visualization algorithms. By describing the current research in this field, it is apparent that heterogeneous computing is a viable option as simulations handle increasingly large amounts of data. Some researchers have implemented hybrid algorithms which use both distributed and shared memory as a way to try and achieve maximal utilization of system resources to solve this problem. Others have relied on hardware agnostic frameworks like VTK-m to reach a hybrid solution with minimal to no code changes. Yet another approach utilizes both CPUs and available accelerators to make sure that the targeted architecture is best suited for the current workload. Additionally, research focused on using a combination of two or more of these techniques is also discussed. A comparison of results from various different types of hybrid flow visualization implementations demonstrates the current body of research that many scientists rely on as the trend toward larger and larger data simulations continues.

Overall, this survey leads to the conclusion that although many researchers are trying to find performant heterogeneous solutions, more work still needs to be done to complete our understanding of heterogeneous performance. Future works revolve around scalability and portability issues of these heterogeneous algorithms.

References

[Ble90] Blelloch G.:. In *Vector Models for Data-Parallel Computing* (1990), vol. 356, MIT Press.

[BPL*19] Binyahib R., Peterka T., Larsen M., Ma K.-L., Childs H.: A Scalable Hybrid Scheme for Ray-Casting of Unstructured Volume Data. *IEEE Transactions on Visualization and Computer Graphics 25*, 7 (July 2019), 2349–2361.

[CBP*14] Childs H., Biersdorff S., Poliakoff D., Camp D., Malony A. D.: Particle Advection Performance Over Varied Architectures and Workloads. In *IEEE International Conference on High Performance Computing (HiPC)* (Goa, India, Dec. 2014), pp. 1–10.

[CCG*12] Camp D., Childs H., Garth C., Pugmire D., Joy K. I.: Parallel Stream Surface Computation for Large Data Sets. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Seattle, WA, Oct. 2012), pp. 39–47.

[CGC*11] Camp D., Garth C., Childs H., Pugmire D., Joy K.: Streamline integration using mpi-hybrid parallelism on a large multicore architecture. *IEEE transactions on visualization and computer graphics 17* (11 2011), 1702–13.

[CKP*13] Camp D., Krishnan H., Pugmire D., Garth C., Johnson I., Bethel E. W., Joy K. I., Childs H.: 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.

[CSH16] Chen M., Shadden S. C., Hart J. C.: Fast coherent particle advection through time-varying unstructured flow datasets. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS 22*, 8 (Aug 2016).

[ECBM08] Endeve E., Cardall C. Y., Budiardja R. D., Mezzacappa A.: Generation of strong magnetic fields in axisymmetry by the stationary accretion shock instability.

[ET18] Ellingwood N., Trott C. R.: *Kokkos Tutorial*. Tech. rep., Sandia National Lab, 2018.

[HB18] Hornung R., Beckingsale D.: *A Tutorial Introduction to RAJA*. Tech. rep., Lawrence Livermore National Lab, 2018.

[Jac19] Jackson K.: The five fastest supercomputers in the world.

[KHN12] Kim J., Hong S., Nam B.: A performance study of traversing spatial indexing structures in parallel on gpu. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems* (June 2012), pp. 855–860.

[KK11] Kunzman D. M., Kale L. V.: Programming heterogeneous systems. In *IEEE International Parallel and Distributed Processing Symposium* (2011).

[LBMC16] Lessley B., Binyahib R., Maynard R., Childs H.: External facelist calculation with data-parallel primitives. In *Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization* (Goslar Germany, Germany, 2016), EGPGV '16, Eurographics Association, pp. 11–20.

[LLN*15] Larsen M., Labasan S., Navrátil P., Meredith J. S., Childs H.: Volume rendering via data-parallel primitives. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2015), PGV '15, Eurographics Association, pp. 53–62.

[MCHG13] Müller C., Camp D., Hentschel B., Garth C.: Distributed parallel particle advection using work requesting. In *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)* (Oct 2013), pp. 1–6.

[MLB*19] Marsaglia N., Li S., Belcher K., Larsen M., Childs H.: Dynamic I/O Budget Reallocation For In Situ Wavelet Compression. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (Porto, Portugal, June 2019), pp. 1–6.

[MSU*16] Moreland K., Sewell C., Usher W., Lo L., Meredith J., Pugmire D., Kress J., Schroots H., Ma K.-L., Childs H., Larsen M., Chen C.-M., Maynard R., Geveci B.: VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A) 36*, 3 (May/June 2016), 48–58.

[MV15] Mittal S., Vetter J. S.: A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys* (2015).

[PCG*09] Pugmire D., Childs H., Garth C., Ahern S., Weber G. H.: Scalable Computation of Streamlines on Very Large Datasets. In *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC09)* (Portland, OR, Nov. 2009).

[PJDA08] P. F., J. L., D. P., A. S.: Petascale algorithms for reactor hydrodynamics. *Journal of Physics: Conference Series 125* (2008), 1–5.

[PYK*18] Pugmire D., Yenpure A., Kim M., Kress J., Maynard R., Childs H., Hentschel B.: Performance-Portable Particle Advection with VTK-m. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (Brno, Czech Republic, June 2018), pp. 45–55.

[sum19] *Summit User Guide*. Tech. rep., Oak Ridge National Lab Leadership Computing Facility, https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide/, 2019.

[WTYH18] Wu K., Truong N., Yuksel C., Hoetzlein R.: Fast fluid simulations with sparse volumes on the gpu. vol. 37.