Adaptive Execution of Particle Advection Workloads

Kristi Belcher*, Hank Childs*, Dave Pugmire**, and Dan the Ghost**

*University of Oregon
**Oak Ridge National Lab

Abstract- **Particle Advection is a fundamental flow visualiation calculation with widely varying workloads. Determining the optimal architecture to run these workloads on is a critical consideration. In this study, we investigate the important tradeoffs that must be factored in when deciding how to best schedule the Particle Advection workloads on appropriate resources. Given this insight, our main contribution is a new algorithm which adapts execution: using the GPU to run relevant parts of the problem and then switching the targeted device according to how the workload evolves. This algorithm is motivated by the observation that CPUs are sometimes able to better perform part of the overall computation since (1) this practice avoids latency times to the GPU and (2) CPUs operate at a faster rate when the workload can't take advantage of the threads on a GPU. We evaluate our algorithm by running workloads that vary over data set, number of particles, number of steps taken, and number of GPU nodes. We then compare our algorithm to traditional GPU-only and CPU-only approaches. Our findings show X, Y, and Z. The results of this study will help inform the Scientific Visualiaiton community about those architectures that give optimal performance at certain stages of the simulation run.**

I. Introduction

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, commonly used to compute streamlines, pathlines, stream surfaces, and even Finite-Time Lyapunov Exponents (FTLE). **Describe the PA algorithm a little bit, and give some background on the queues - this will be important for understanding that it is a critical algorithm to get efficient in a large-data environment. The Particle Advection algorithm involves three queues: active, inactive, and terminated. During the simulation, particles are advecting and evolving through the problem domain. As the particles are advecting, they can either terminate, go outside the domain and become inactive, or move into the domain becoming active.** Although Particle Advection is such an important calculation for vector field visualizaiton, 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 for particles is user defined and thus can be very different from one run to the next. Thus, Particle Advection is a data dependent problem that requires variable computational resources to sucessfully trace the particle streams. As simulations require more and more data to produce meaningful results, visualization and analysis is being performed in situ. This means that the data is generated and the 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 - cite?. 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 archtiectures 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. This kind of hybrid computing requires that researchers tailor their codes to two (or more in some cases) architectures at a time. To help ease the transition of efficiently using multiple hardwares, VTK-m was created to provide a hardware agnostic framework that acts as a bridge between several different plateforms [MSU*16]. VTK-m is an extension of VTK, with the addition of several many-core implementations for common visualization algorithms. Figure 1 and 2 shows a pictoral representation of how VTK-m can be used - cite. Note that in (a) a developer must implement several versions of the application code: one for each desired harware 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. Additionally, VTK-m supports general portablility across a wide variety of architectures, thus making portable performance possible.

In this study, we analyze the performance impact of utilizing more than just one architecture during the same run of a VTK-

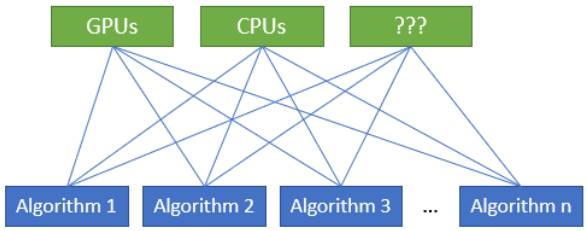

Fig. 1. A picture of how each algorithm must be ported to each architecture.

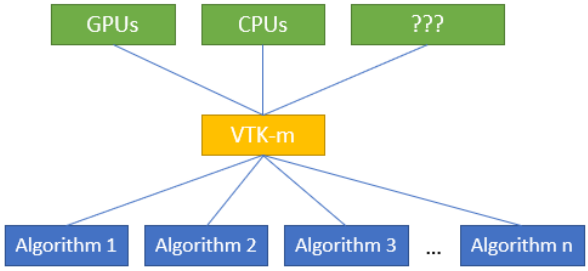

Fig. 2. A picture of how with VTKm, each algorithm only has to be mapped once to VTKm.

m Particle Advection program. We contribute a new Particle Advection algorithm which uses an oracle to determine, at runtime, whether the CPU or the GPU can best handle the current workload, switching to that particular device. As the workload evolves, our algorithm can switch back and forth from CPU to GPU depending on which device's hardware resources can be best utilized. We compare the performance between programs which use either CPU-only or GPU-only to produce results and our program which adaptively selects a device during execution. Our algorithm is motivated by the observation that sometimes the workload can't take full advantage of GPU threads, leading to some sitting idle and wasting resources. This would lead to the speculation that leaving the data on the CPU to be computed would perform better. At other times, the workload may be much more able to fully load the GPU and take advantage of its hardware abilities. In effect, our work explores the fundamental research question: **If faced with a Particle Advection problem, can adaptive execution give better performance than running the entire workload on either the CPU or the GPU?**

Other important contributions of this paper include the insights it provides for visualization scientists who are studying Particle Advection and other flow visualizaiton problems. For example, in implementing an oracle to efficiently determine the best device to run the algorithm, we had to factor in several different characteristics of the current workload. These characteristics included (1) number of particles left in the domain to advect, (2) how many times a particular particle has been switched to a different device, and (3) how many times a particular particle when back and forth from active queue to inactive queue. These considerations were important for scultping an oracle that could best determine the appropriate device for the workload, and thus will give other researchers interesting insight into the factors that affect which architectures will be best. We make the following contributions. We introduce a performance portable, VTK-m Particle Advection algorithm which utilizes both the CPU and the GPU depending on the current state of the workload to achieve a more efficient result. We demonstrate how certain characteristics of the data set can be used as an oracle to determine the appropriate hardware for the current workload. Additionally, we compare our results and findings with other implementations which only use the CPU or only the GPU and provide analysis of this insight. The following sections describe our algorithm and results in more detail. (Could say more about what sections are)

II. Related Work

In this section, we discuss related works grouped into two main areas: Particle Advection and Parallel Visualization Systems.

A. Particle Advection

Because processing integral lines has remained a key computation for Vector field visualization[MLP*10], much work has explored how to make this calculation fast and efficient. Pugmire et. al. discussed two fundamental approaches to calculating the integral lines for vector field visualization.[PCG*09] First *Parallelize-over-Seeds* (POS), involves distributing the particle seeds across nodes to be processed independent of each other. POS often does not scale as well as the other approach because of redundant I/O operations. This leads to the second approach, *Parallelize-over-Blocks* (POB), where each node is assigned a block of the domain to process. The node assigned to a particular block is in charge of calculating traces for any particles that enter its block. However, it is easy to see how load balancing will become a problem for this approach as not all pieces of the domain have a proportional amount of work. Therefore, it is common to adopt some kind of hybrid approach of both of these techniques to increase performance and alleviate the potential downfalls of each technique by itself. Pugmire et. al. [DP12] explore this kind of algorithm in a distributed memory parallel environment.

Camp et. al. [CGC*11] propose a MPI and OpenMP based solution where MPI is used to parallelize across mulitple nodes, and OpenMP is used to parallelize local advection calculations. They propose this scheduling approach to manage how these two APIs interact. Then in [CKP*13], they propose a similar schedule with CUDA implementations substituted for OpenMP. Here they analyze the performance impact between the two schedules.

Other previous work focuses on optimizing the POB technique. Yu et. al. [YWM07] proposes a data-aware POB strategy. Similarly, Nouanesengsy et. al. [SNL11] use a statistical approach to calculate the propagration probability of particles across blocks. They predict the movement of particles to more efficiently partition blocks across nodes. In their work, they explore the POB strategy and find a unique way to optimize it. Additionally, Nouanesengsy et. al. [NLL*12] also

used time intervals as a way of optimizing how blocks get assigned to the different nodes. With this approach they can more efficiently compute the Finite Time Lyapunov Exponent (FTLE) by distributing different time intervals to nodes and generating pathlines accordingly.

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. In this way, most of the load imbalance problem is solved unless there are other imbalance issues in other areas of the problem.

B. Parallel Visualization Systems

In 1990, Blelloch proposed a model where specific operations could be carried out on vectors of size N in O(log(N) time in the worst case [Ble90]. This worked inspired the use of Data Parallel Primitives (DPPs) such as Scan, Scatter, Map, Reduce, and Gather used in libraries like NVIDIA's Thrust [BH12] library. Since The Thurst 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 such as VTK-m [MSU*16]. VTKm 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, VTKm 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. In short, VTKm was created to provide the same functionalities as VTK [SML96], but with the inclusion of achieving portable performance across a variety of many-core and multi-core architectures.

III. Problem and Algorithm Overview

In this section, we discuss the Particle Advection algorithm itself and several key aspects of the algorithm that we took into consideration when implementing our new algorithm. We also discuss our algorithm in detail, explaining our new contributions and the benefits of this implementation.

A. Particle Advection Overview

Particle Advection starts by displacing a particle for a short distance in a velocity field. That particle will be advected from one location to another nearby. This displacement is also known as an advection step. After a series of advection steps, the integral curve can be observed by the total path, or sequence of advection steps from the initial location to the terminal location, that particle has traveled. Because the advection steps must be calculated sequentially, advecting particles is a data dependent problem. In other words, in order to calculate a particle's Nth location, we must know it's (N-1)th location. Additionally, the number of advection steps for any given particle varies, adding more complexity to the problem. For example, the particle could advect into a sink, go outside the problem domain, or meet some other kind of termination criteria. In this study, our Particle Advection algorithm uses a parallelization-over-data approach where the domain is divided into N pieces in which each of the N tasks is assigned to a piece of the total problem. A particular task will execute particles on its piece of the domain until its assigned particles terminate or advect outside of its piece of the domain. Here, a task will fork two threads: a manager thread to communication and keep track of overall progress and a worker thread to advect particles given to it by the manager. If a particle does travel beyond a task's domain to another piece, it is communicated (by the manager thread) to whatever task owns that piece of the domain. This process repeats until all particles have terminated. At this point we can see the paths that each particle has traveled, giving us a map of the particles' trajectories.

B. Algorithm Overview

Our study uses the algorithm introduced in [CKP*13], described in section 2. (fix reference later) Here, we will discuss only the important parts of the previous algorithm that will help readers understand how we've modified it to create our new algorithm. Talk about initialization and advection phases + queues. The particle advection algorithm involves three queues: one for active particles, one for the inactive particles, and the last for terminated particles. The active queue includes the particles that can be immidiately worked on and that are within the task's domain. The inactive queue includes the particles that (have advected into a sink?) have gone outside the domain boundaries and must be communicated to another task. Lastly, the terminated queue includes those particles that have met some termination criteria and are done advecting. In this way, our algorithm mimicks that described in [CKP*13]. However, we've modified how the worker queue interacts with the manage queue in our new algorithm, adding a distinctive CPU worker thread and a GPU worker thread. We want to take advantage of a GPU's hardware resources when the workload calls for it, but then switch to a GPU when the workload no longer can take full advantage of the GPU. Here, we describe how exactly we have accomplished this novel idea (or some kind of transition). First, all particles are initialized on the GPU worker thread's queue for simplicity. After the first round of work, if our oracle determines that the workload is best suited for the GPU, then the manage thread will add work to the GPU worker thread's active queue. Otherwise, the manage thread will insert that work into the CPU worker

thread's active queue. In this way, there is no conflicts within a single worker queue, but instead two separate queues that can be read depending on what the oracle determines about the current state of the workload. This process continues until all the work is done. There will probably need to be another section that talks specifically about the oracle and how it works + why we implemented it the way we did.

IV. Study Overview

To study the effect of adaptively selecting the target device during the Particle Advection execution, we conducted various experiments over a range of carefully tuned parameters. Our experiments varied over 4 main factors:

- **Data Set**: Here we had several different simulation codes that were used to thoroughly test our algorithm. We wanted to thoroghly test how our oracle worked on different codes that behave very differently from eachother. Our simulations included:
  - Fusion: The Fusion data set comes from the NIMROD (cite) simulaiton code. In this particular simulation, a magnetically confined fusion reaction in a tokamak device is shown. During this simulation, the magnetic field travels in a helical fashion around the torus to achieve stable equilibrium. We see that the particles behave in a highly circular way as they traverse the torus-shaped vector field repeatedly.
  - Double Gyre: This data set is an artifical simulation code of blahhhh. (cite)
  - Fish Tank The Fish Tank data set shows water entering a rectangular tank from the top side and splashing around until it reaches a steady state. (cite)
  - Astrophysics: This data set originates from a simulation of a supernova. The data shows the the magnetic field surrounding a solar core collapsing, and the resulting explosion. The data was computed by the GENASIS simulation code (cite).
- **Number of Particles**: We ran our experiments with 1000, 10000, and 100000 particles. This way we could see how frequently one device was called versus another.
- **Number of Steps Taken**: Our experiments used 10000, 20000, and 50000 steps. Each step size was either .010 or .025. This way we could also see how varying our steps would effects the decisions of the oracle.
- **Number of GPU Nodes**: We ran our experiments on Summit and Titan (and NERSC?). (cite something) They have different GPUs like X, Y, and Z. Talk about node architecture!

Cite a bunch of stuff here. Testing - [CCG*12] The experiments were within the VTK-m framework. Mention vtk-h? Testing - [PCG*09] Runtime environment and measurements taken...Testing - [KKPC16]

V. Results

Our results showed that we can get similar or much better performance with our algorithm compared to the traditional approach. In fact, in the best case, we see a performance speedup of up to 1.5x. We did a lot of our analysis in Visit [CBB*05] which allowed us to verify our results and make other conclusions...yatta yatta. Figure 3 shows the best kind of dog. Our major findings were....some cool pictures! Figure 4, 5, and 6 show pictures of the data sets we used. Analysis of major findings, why it's interesting... Our results demonstrate that our oracle was successful in deciding which device to run the current workload on. The most crutial criteria for our oracle is a function of both the number of particles and the

Fig. 3. A Chihuahua.

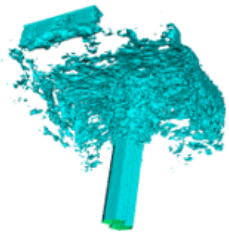

Fig. 4. A picture of the Hydraulics data set.

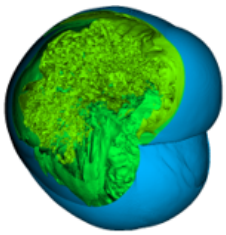

Fig. 5. A picture of the Astrophysics data set.

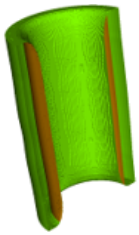

Fig. 6. A picture of the Fusion data set.

number of advection steps. Our oracle was able to determine when the workload started "ping ponging" particles inbetween domain boundaries. At this point, directing the code to be run on the CPU increased performance because the CPU threads could more efficiently process this behavior. (Talk about why) More on why the oracle was good and what it actually did. Something about cool insight that would be interesting to Scientific Visualizaiton researchers. Comparison of results to others? Major comparison to "Particle Advection Performance Over Varied Architectures and Workloads"...

VI. Conclusions and Future Work

We studied the affects of adaptive device selection implementation of a vtk-m Particle Advection program. We have shown that by analyzing the current workload and selecting an appropriate device, we get better performance over the traditional single-device alternatives. Our findings suggest that adaptively selecting an appropriate device can also lead to better energy efficiency, in addition to enhanced performance. We hope to get this paper published and solve world hunger while we're at it. Should we do a summary of findings? Itemize!

My work described in this paper highlights the performance benefits and insights that I accomplished using specifically the Particle Advection algorithm. For our future work, it would be beneficial to see how this idea could be extended to other Scientific Visualizaiton algorithms. For example, can we see similar performance benefits with other similar algorithms? What about algorithms which have more irregular control flows or memory access patterns? There are several interesting research questions that can be explored by extending this idea to other kinds of problems. More on this later.

VII. Acknowledgements

I would like to give a special thanks to the Scientific Visualization group at Oak Ridge National Labratory for their help and computational resources. My work was made possible by many researchers in the group, which I very much appreciate. Other thanks?

References

[BH12] Bell N., Hoberock J.: Thrust: Productivity-oriented library for cuda. *Astrophysics Source Code Library 7* (12 2012), 12014–.

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

[CBB*05] Childs H., Brugger E., Bonnell K., Meredith J., Miller M., Whitlock B., Max N.: A contract based system for large data visualization. In *VIS 05. IEEE Visualization, 2005.* (Oct 2005), pp. 191–198.

[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.

[DP12] D. Pugmire T. Peterka C. G.: Parallel integral curves. In *In High Performance Visualization: Enabling Extreme-Scale Scientific Insight* (2012), Chapman & Hall, CRC Press.

[KKPC16] Kress J., Klasky S., Pugmire D., Childs H.: Visualization and Analysis Requirements for In Situ Processing for a Large-Scale Fusion Simulation Code. In *Proceedings of the Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), held in conjunction with SC16* (Salt Lake City, UT, Nov. 2016).

[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.

[MLP*10] McLoughlin T., Laramee R. S., Peikert R., Post F. H., Chen M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829.

[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.

[NLL*12] Nouanesengsy B., Lee T.-Y., Lu K., Shen H.-W., Peterka T.: Parallel particle advection and ftle computation for time-varying flow fields. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Los Alamitos, CA, USA, 2012), SC '12, IEEE Computer Society Press, pp. 61:1–61:11.

[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).

[SML96] Schroeder W. J., Martin K. M., Lorensen W. E.: The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *Proceedings of Seventh Annual IEEE Visualization '96* (Oct 1996), pp. 93–100.

[SNL11] Shen H., Nouanesengsy B., Lee T.: Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization & Computer Graphics 17*, 12 (2011), 1785–1794.

[YWM07] Yu H., Wang C., Ma K.-L.: Parallel hierarchical visualization of large time-varying 3d vector fields. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2007), SC '07, ACM, pp. 24:1–24:12.