Eurographics Symposium on Parallel Graphics and Visualization (2021)
M. Hadwiger, M. Larsen, F. Sadlo (Editors)

HyLiPoD: Parallel Particle Advection Via a Hybrid of Lifeline Scheduling and Parallelization-Over-Data

Roba Binyahib[1], David Pugmire[2], and Hank Childs[3]

[1]National Renewable Energy Laboratory, [2]Oak Ridge National Laboratory, [3]University of Oregon

Abstract

Performance characteristics of parallel particle advection algorithms can vary greatly based on workload. With this short paper, we build a new algorithm based on results from a previous bake-off study which evaluated the performance of four algorithms on a variety of workloads. Our algorithm, called HyLiPoD, is a "meta-algorithm," i.e., it considers the desired workload to choose from existing algorithms to maximize performance. To demonstrate HyliPoD's benefit, we analyze results from 162 tests including concurrencies of up to 8192 cores, meshes as large as 34 billion cells, and particle counts as large as 300 million. Our findings demonstrate that HyLiPoD's adaptive approach allows it to match the best performance of existing algorithms across diverse workloads.

1. Introduction

Particle advection is a foundational operation for many flow visualization techniques. The technique seeds a massless particle at some location, and then calculates the trajectory that the particle follows. For a given vector field, the advection process calculates a trajectory that is tangent to the vector field at all locations. The process is carried out by performing a series of advection steps, each of which calculates a new position from the current one. In practice, each advection step employs a numerical method (typically a 4th order Runge-Kutta) to solve an ordinary differential equation for where the particle travels over a short period. Finally, the particle advection process can be very computationally expensive. When workloads involve many particles and/or many advection steps per particles, then this process can involve solving billions or even trillions of ordinary differential equations.

With this work, we consider parallel particle advection subject to three conditions that are quite common in supercomputing environments. First, the parallel environment is distributed-memory, i.e., each processing element has its own private memory. Second, the vector field data is so large that it cannot fit into the memory of any processing element, and so it must be decomposed into blocks. These first two conditions often make it difficult to achieve efficient performance, as there is overhead to make sure that the advection step for a given particle has access to the necessary block of data from the vector field. The third condition is that the vector field data is steady state, i.e., flow visualization techniques will operate on a single time step. Multiple visualization algorithms have been proposed to achieve efficient performance in this setting, and their efficacy varies, especially based on the workload (seed locations, number of seeds, vector field properties, and duration of advection).

Our previous work [BPYC20] performed an empirical study to understand how different parallelization algorithms perform for different workloads. We compared four parallelization algorithms: Parallelization-Over-Data (POD), Parallelization-Over-Particles (POP), a work requesting algorithm that used the Lifeline Scheduling Method (LSM) [BPNC19], and a Master Worker (MW) algorithm [PCG*09]. The results of the study showed that the two algorithms that had the best performance are POD and LSM. However, each one of them had some workloads where they performed poorly. With this work, we propose an algorithm that chooses between POD and LSM to get the best possible performance. We call this algorithm HyLiPoD — **Hy**brid between **Li**feline and **P**arallelization-**o**ver-**D**ata. In this paper, we compare the performance of our new algorithm with the performance of the four algorithms (POD, POP, LSM, and MW). Our results show that HyLiPoD performs better than or equal to each of the four algorithms for almost all workloads.

2. Related Work

There are three main categories of parallel particle advection: parallelization-over-data (POD), parallelization-over-particles (POP), or a hybrid of the two. For each of POD and POP, there is a general way to carry out the method and then possible optimizations within that method. With POD, data blocks are assigned to different nodes, and each node advects the particles located in its assigned blocks. Extensions of POD focused on balancing the workload between nodes. Peterka et al. [PRN*11] proposed two solutions to balance the workload: the first solution used a round-robin block assignment and the second one used dynamic load balancing. Other solutions have used workload estimation techniques [CF08, YWM07, NLS11] to balance the workload. With POP, particles are distributed equally between nodes and data blocks are loaded as needed. Extensions of POP focused on balancing the workload between nodes, as well as reducing the cost of I/O. Work requesting [MCHG13, LSP14, BPNC19] and dynamic load balancing [MP16, ZGH*18] have been used to balance the workload. Techniques like extended memory hierarchy [CCC*11] and data prefetching [RTBS05, AR13] have been used to reduce the cost of I/O. Finally, hybrids algorithms [LSP14, KWA*11, PCG*09] between POP and POD have aimed to limit the disadvantage that each algorithm have individually. Our own algorithm is technically a hybrid, since it uses both POD and a variant of POP (LSM). That said, previous hybrid algorithms have adapted during run-time, while ours adapts at the beginning of execution.

We compared HyLiPoD to four algorithms: POP, POD, LSM, and MW. For POD and POP, we used straightforward implementations rather than some of the improvements that have been published previously. In particular, some of the POD approaches require pre-processing time, and we envision HyLiPoD being used in scenarios with no pre-processing overhead. One optimization we did not implement was round robin assignments; we felt the extra communication would be problematic for our workloads, but feel further evaluation would be good future work. For POP, some optimizations read a block a single time and then communicate that block to other ranks, rather than reading it again. We experimented with this optimization, and found that it did not enhance performance for our supercomputer and workloads. Other POP optimizations are captured in our LSM implementation. For hybrid algorithms, we feel MW and the work by Kendall et al. [KWA*11] have similar elements and choosing one for evaluation is sufficient. The other noteworthy hybrid algorithm comes from Lu et al. [LSP14]. We do not compare with their algorithm because it has ranks communicate blocks, which, as mentioned before, we found did not enhance performance for our supercomputer and workloads.

3. Algorithm

Our main idea is to create a meta-algorithm. There are two steps for creating this meta-algorithm: (1) data collection and (2) constructing an oracle. The data collection step entails studying the performance of existing algorithms on important workloads. The oracle step involves using the results of the first step to choose the best performing for a workload. In all, when a meta-algorithm encounters a workload, it should be capable of deploying the best existing algorithm. Fortunately, our previous work [BPYC20] already generated the results of the data collection step, which accelerated work for this study.

Our oracle is based on two key observations from the data collection step. First, POD and LSM consistently ranked as the best performers, and so we only needed to consider these two algorithms. Second, the matching between workload and best performing algorithm was straightforward. The POD algorithm has the best performance for workloads with uniform particles distribution, i.e., large seeding box. The LSM algorithm has the best performance for workloads with denser particles distribution, i.e., small seeding box, because there are fewer blocks to load, which leads to less I/O cost.

Algorithm 1 Pseudocode for the HyLiPod algorithm.

```
$ListBlockIdInBox \leftarrow GetBlocksInBox(seedingBox)$
$blocksIncludedPercent \leftarrow ListBlockIdInBox.size()/NumBlocks$
if $blocksIncludedPercent > Threshold$ then
    $algo \leftarrow POD$
else
    $algo \leftarrow LSM$
end if
$numActive \leftarrow TotalNumParticle$
$activeParticles \leftarrow GenerateParticles(ListBlockIdInBox)$
while $numActive > 0$ do
    if $activeParticles.size() > 0$ then
        $WorkerFunction(activeParticles, algo)$
    end if
    $CommunicationFunction(activeParticles, algo)$
end while
```

The remainder of this section describes our HyLiPoD algorithm in more depth. In the beginning of the program, the algorithm calls a function that takes the seeding box and calculates what percentage of the total data domain is included. Then, this information is used to decide which algorithm is better (POD or LSM) by comparing the percentage of included blocks to a given threshold. In Section 5.1, we describe our process for choosing this threshold.

Pseudocode can be found in Algorithm 1, which uses the following building blocks:

- GetBlocksInBox(): a function that returns a list of block ids that are within the seeding box boundaries.
- GenerateParticles(): a function that generates the initial seeds.
- WorkerFunction(): a function that performs I/O operations, advection and process the particles after advection.
- CommunicationFunction(): a function that sends and receives data (particles or messages) between nodes.

4. Experiment Overview

We compare the performance of HyLiPoD with the four algorithms we considered in our previous study: POD, POP, LSM and MW.

4.1. Algorithm Comparison Factors

We consider three factors in our study: Size of Seeding Box (3 options), Total Number of Steps (6 options), and Concurrency (3 options). We considered the cross product of this factors, i.e., 54 configurations (3*6*3). Further, we tested each configuration on all algorithms, meaning 270 tests (54*5).

Size of Seeding Box: The seeding box size represents the distribution of particles across the data domain. This can impact the I/O cost, communication cost, and the load balance. We consider three seeding boxes: large box, mid box, and small box. The large seeding box option has particles distributed across the entire data set (100%), while the mid (50%) and small (10%) seeding boxes have particles distributed in sub-regions of the data.

Total Number of Steps: The total number of steps determine the amount of computations need to be performed. The total number of steps is a product of the number of particles and the duration of each particle.

We consider three options for the number of particles and two options for the duration of each particle, which results in six total options (3×2). The options for the number of particles represented as a ratio of particles per cell, i.e., 1 particle for each C cells. The three options are: for every 100 cells ($P/100C$), for every 1000 cells ($P/1KC$), and for every 10,000 cells ($P/10KC$). With this configuration, as the data size increases, the number of particles increases. For the duration of each particle we consider two options: 1000 steps ($1K$) and 10,000 steps ($10K$).

Concurrency: We test the weak scalability by varying the number of MPI tasks, and then increasing the number of data blocks and the number of particles in proportion. We considered three concurrencies, and for each concurrency we ran 4 MPI tasks per node. Further, as concurrency increased, we increased the number of data blocks, and kept the size of each data block constant (128^3 cells). In all, the three options for concurrency were:

Config. Name	Nodes	MPI tasks	Cores	Blocks
Concurrency1	4	16	128	256
Concurrency2	32	128	1024	2048
Concurrency3	256	1024	8192	16384

4.2. Data Set

We primarily used the "Fishtank" data set for our study, which is a thermal hydraulics simulation by the NEK5000 [FLK08] code. It is a simulation of twin inlets that pump water of different temperatures into a box, the vector field represent the fluid flow. In this data set, the mixing behavior and the temperature of the water are of interest. Finally, the initial round of experiments considers two additional data sets, to ensure that our approach is not specific to a single data set. These additional data sets are "Astro," a simulation by the GenASiS [ECBM10] code of a magnetic field surrounding a solar core collapse, and "Fusion," a simulation by the NIMROD [SGG*04] code of a magnetically confined plasma in a tokamak device.

4.3. Hardware Used

Our experiments were primarily run on Cori, from Lawrence Berkeley National Laboratory's NERSC facility. Cori has 2,388 Intel Xeon "Haswell" processor nodes. Each node has 128GB of memory and two sockets, and each socket has a 2.3 GHz 16-core Haswell processor, where each core supports 2 hyper-threads. The only experiments not run on Cori are plotted in Figure 1. These experiments were run on "Alaska," a cluster at the University of Oregon, also with Xeon Haswell nodes, but running at 3.5GHz. A different machine was used for these experiments because our allocation on Cori expired, but we still wanted to study more data sets. Further, the results for these experiments were highly similar to comparable runs on Cori.

5. Results

This section discusses the results of our study in three phases. Phase 1 discusses results on picking the best threshold to switch between POD and LSM. Phase 2 discusses the performance of HyLiPOD for all workloads. Finally, Phase 3 compares the performance of HyLiPOD with the four algorithms POD, POP, LSM, and MW.

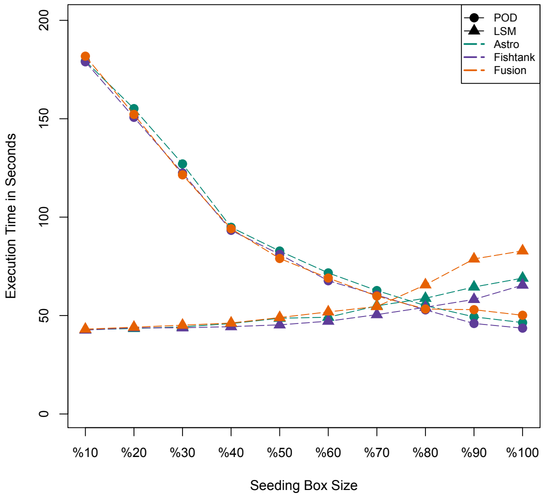

Figure 1: *Comparing the performance of POD and LSM for the workload $P/1KC * 10K$ for 10 seeding box sizes and 3 data sets (Astro, Fishtank, and Fusion) using Concurrency1.*

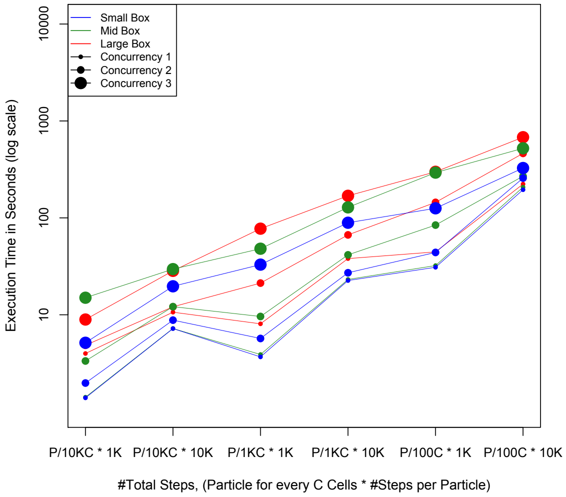

Figure 2: *Execution times for HyLiPoD for the three different seeding box sizes (color) and concurrencies (glyph size).*

5.1. Phase 1: Threshold Selection

Our previous results [BPYC20] demonstrated that meta-algorithm oracle could predict the best algorithm strictly on seeding location (concentrated or spread out). To confirm the stability of this finding, we began by re-running experiments using more seeding locations and more data sets. Specifically, we tested the performance of POD and LSM for seeds that occupied 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% of the volume for each of the Fishtank, Astro, and Fusion data sets. Figure 1 shows that the POD algorithm performs better when the size of the seeding box covers more than 70% of the data domain, while the LSM algorithm performs better when the size of the seeding box covers less or equal than 70%. The

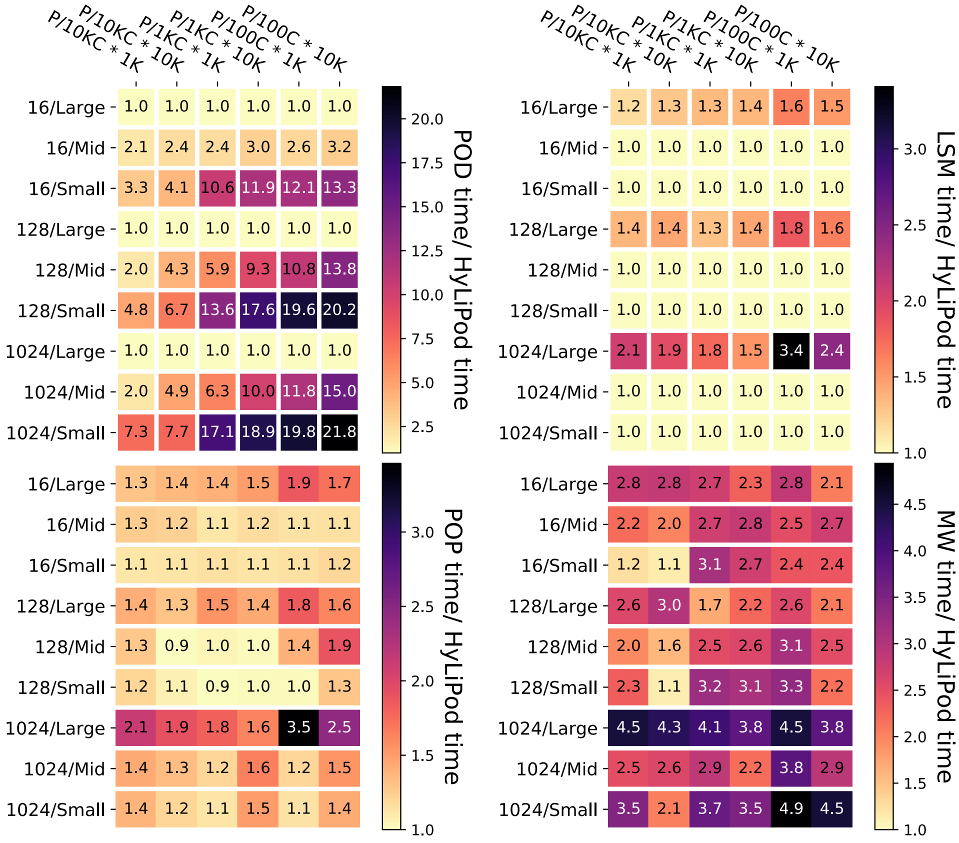

Figure 3: *The difference in execution time between our hybrid parallel particle advection algorithm (HyLiPoD) and the four algorithms. For each workload in the heat map, the number represents the execution time of the algorithm divided by the execution time of HyLiPod.*

results fit previous findings. The POD algorithm has less I/O cost than LSM because each rank is responsible for part of the data only, while in the LSM algorithm, ranks load data on demand. On the other hand, LSM performs better than POD for smaller seeding box sizes because it is better at balancing the workload. This is because in the POD algorithm each rank is responsible for advecting only particles located in its data blocks, which cause some ranks to stay idle.

Importantly, both algorithms show clear monotonic trends — LSM gets slower as the seeding box size increases, while POD gets faster. As a result, we can feel more confident that our straightforward oracle will perform well, i.e., there are not unusual local minima or maxima for algorithm behavior. Therefore, we assign a threshold of 70% for the rest of the study, which corresponds to the crossover point between the two algorithms.

5.2. Phase 2: HyLiPoD Performance

Figure 2 shows the performance of HyLiPoD for the three seeding box sizes at different concurrencies. Since HyLiPoD delivers the best algorithm for a given workload, its overall performance informs future research for our community. In particular, Figure 2 has six particle advection workloads, for each one of the workloads, the number of particle advection steps per MPI rank is the same through different concurrencies. Troublingly, the performance varies by almost a factor of 10X. For example, the "P/10KC * 1K" workload takes about a second for a small box at low concurrency, but takes over 10 seconds for a mid box at high concurrency — even though both tests are doing the same number of advection steps per MPI rank. Concurrency is still clearly an issue, as the "P/10KC * 1K" with a small box at high concurrency is still ~5X slower than the small concurrency test. Other workloads have similar disparities, indicating that many workloads still have no good parallelization approach. Further, for the worst performers, HyLiPoD is actually switching between algorithms — some are with mid box (which uses LSM) and some are with large box (which uses POD) — meaning both perform similarly (and poorly).

5.3. Phase 3: Comparing the Algorithms

Figure 3 shows the performance of HyLiPoD compared to the four algorithms. We feel there are several interesting findings. First, the improvement of HyLiPoD over the four algorithms varies greatly. For each of the four algorithms, we calculated an average speedup, i.e., the average over our 54 configurations. These averages are 1.2X for LSM, 1.4X for POP, 2.8X for MW, and 6.6X for POD. Further, these averages inform the benefit of deploying HyLiPoD over any single algorithm. Second, we see that our choice focusing on only two algorithms was mostly good. POP did beat LSM (and thus HyLiPoD) in a few cases, due to extra communication and sometimes extra I/O. That said, the performance differences were small (0.97 and 0.95), and disappear at higher concurrency. We feel the simplicity of using only two algorithms outweighs this small potential speedup. Finally, we find the patterns for each algorithm to be interesting. While POD's behavior (good for large boxes and poor elsewhere) was expected, MW's behavior is more varied. MW adapts its execution dynamically, and so it avoids doing as poorly as POD for mid and small seeding boxes. That said, other overheads prevent it from actually exceeding the LSM/POP approaches on these boxes, and so it fails to win any workloads. Regardless, its somewhat uniform behavior over workloads may indicate potential going forward, provided overheads can be removed.

6. Conclusions And Future Work

This paper introduced HyLiPoD, and demonstrated that its performance is equal to or better than four prominent existing algorithms. Going forward, we believe the meta-algorithm approach is the best way to deliver performant parallel particle advection to domain scientists. We view HyLiPoD as the start of such an effort. Further, as troublesome particle advection workloads merit the development of new algorithms, we believe that these algorithms should be placed within a meta-algorithm.

From the meta-algorithm perspective, HyLiPoD was fortunate in two key ways: (1) the assessment step for its oracle ran quickly enough to not impact execution time and (2) the oracle was quite easy to devise. In the future, as more workloads and more algorithms are added, these two fortunate properties may no longer hold, and thus building an oracle will likely require additional research. Finally, this work focused on steady state flow, and it will take new investigation to understand which lessons learned will be applicable in an unsteady state setting. In particular, particle positions from one time slice will form the seed positions for the next time slice, and the data-specific ways they advance may lead the fundamentally different types of seeding configurations than the steady state case. We plan to study the impact of different oracles on the performance of our algorithm. As well as adding techniques to adjust the parallelization algorithm used during run time.

Acknowledgment

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research was supported by the Scientific Discovery through Advanced Computing (SciDAC) program of the U.S. Department of Energy. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

References

[AR13] Akande O. O., Rhodes P. J.: Iteration aware prefetching for unstructured grids. In *2013 IEEE International Conference on Big Data* (Oct 2013), pp. 219–227. doi:10.1109/BigData.2013.6691578. 2

[BPNC19] Binyahib R., Pugmire D., Norris B., Childs H.: A lifeline-based approach for work requesting and parallel particle advection. In *2019 IEEE 9th Symposium on Large Data Analysis and Visualization (LDAV)* (Oct 2019), pp. 52–61. doi:10.1109/LDAV48142.2019.8944355. 1, 2

[BPYC20] Binyahib R., Pugmire D., Yenpure A., Childs H.: Parallel Particle Advection Bake-Off for Scientific Visualization Workloads. In *IEEE International Conference on Cluster Computing (CLUSTER)* (Kobe, Japan, Sept. 2020), pp. 381–391. 1, 2, 3

[CCC*11] Camp D., Childs H., Chourasia A., Garth C., Joy K. I.: Evaluating the benefits of an extended memory hierarchy for parallel streamline algorithms. In *2011 IEEE Symposium on Large Data Analysis and Visualization* (Oct 2011), pp. 57–64. doi:10.1109/LDAV.2011.6092318. 2

[CF08] Chen L., Fujishiro I.: Optimizing parallel performance of streamline visualization for large distributed flow datasets. In *2008 IEEE Pacific Visualization Symposium* (March 2008), pp. 87–94. doi:10.1109/PACIFICVIS.2008.4475463. 2

[ECBM10] Endeve E., Cardall C. Y., Budiardja R. D., Mezzacappa A.: Generation of Magnetic Fields By the Stationary Accretion Shock Instability. *The Astrophysical Journal 713*, 2 (2010), 1219–1243. 3

[FLK08] Fischer P. F., Lottes J. W., Kerkemeier S. G.: nek5000 Web page, 2008. http://nek5000.mcs.anl.gov. 3

[KWA*11] Kendall W., Wang J., Allen M., Peterka T., Huang J., Erickson D.: Simplified parallel domain traversal. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 10:1–10:11. URL: http://doi.acm.org/10.1145/2063384.2063397, doi:10.1145/2063384.2063397. 2

[LSP14] Lu K., Shen H., Peterka T.: Scalable computation of stream surfaces on large scale vector fields. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2014), pp. 1008–1019. doi:10.1109/SC.2014.87. 2

[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. doi:10.1109/LDAV.2013.6675152. 2

[MP16] Morozov D., Peterka T.: Efficient delaunay tessellation through k-d tree decomposition. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2016), pp. 728–738. doi:10.1109/SC.2016.61. 2

[NLS11] Nouanesengsy B., Lee T. Y., Shen H. W.: Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (Dec 2011), 1785–1794. doi:10.1109/TVCG.2011.219. 2

[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 Conference on High Performance Computing Networking, Storage and Analysis* (Nov 2009), pp. 1–12. doi:10.1145/1654059.1654076. 1, 2

[PRN*11] Peterka T., Ross R., Nouanesengsy B., Lee T. Y., Shen H. W., Kendall W., Huang J.: A study of parallel particle tracing for steady-state and time-varying flow fields. In *2011 IEEE International Parallel Distributed Processing Symposium* (May 2011), pp. 580–591. doi:10.1109/IPDPS.2011.62. 1

[RTBS05] Rhodes P. J., Tang X., Bergeron R. D., Sparr T. M.: Iteration aware prefetching for large multidimensional datasets. In *Proceedings of the 17th International Conference on Scientific and Statistical Database Management* (Berkeley, CA, US, 2005), SSDBM'2005, Lawrence Berkeley Laboratory, pp. 45–54. URL: http://dl.acm.org/citation.cfm?id=1116877.1116883. 2

[SGG*04] Sovinec C., Glasser A., Gianakon T., Barnes D., NEBEL R., KRUGER S., PLIMPTON S., TARDITI A., CHU M., THE NIMROD TEAM: Nonlinear Magnetohydrodynamics with High-order Finite Elements. *J. Comp. Phys. 195* (2004), 355. 3

[YWM07] YU H., WANG C., MA K.: Parallel hierarchical visualization of large time-varying 3d vector fields. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing* (Nov 2007), pp. 1–12. doi:10.1145/1362622.1362655. 2

[ZGH*18] ZHANG J., GUO H., HONG F., YUAN X., PETERKA T.: Dynamic load balancing based on constrained k-d tree decomposition for parallel particle tracing. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (Jan 2018), 954–963. doi:10.1109/TVCG.2017.2744059. 2