DRP Report:
Accelerating Advection Via Approximate Block Exterior Flow Maps

Ryan Bleile
University of Oregon

ABSTRACT

Flow visualization techniques involving extreme advection workloads are becoming increasingly popular. While these techniques often produce insightful images, the execution times to carry out the corresponding computations are lengthy. With this work, we introduce an alternative to traditional advection. Our approach centers around block exterior flow maps (BEFMs). BEFMs can be used to accelerate flow computations by reducing redundant calculations, at the cost of decreased accuracy. Our algorithm uses Lagrangian interpolation, but falls back to Eulerian advection whenever regions of high error are encountered. In our study, we demonstrate that the BEFM-based approach can lead to significant savings in time, with limited loss in accuracy.

1 INTRODUCTION

A myriad of scientific simulations, including those modeling fluid flow, astrophysics, fusion, thermal hydraulics, and others, model phenomena where constituents move through their volume. This movement is captured by a velocity field stored at every point on the mesh. Further, other vector fields, such as force fields for electricity, magnetism, and gravity, also govern movement and interaction. A wide range of flow visualization techniques are used to understand such vector fields. The large majority of these techniques rely on placing particles in the volume and analyzing the trajectories they follow. Traditionally, the particles are displaced through the volume using an advection step, i.e., solving an ordinary differential equation using a Runge-Kutta integrator.

As computational power on modern desktops has increased, flow visualization algorithms have been empowered to consider designs that include more and more particles advecting for longer and longer periods. Techniques such as Line Integral Convolution and Finite-Time Lyapunov Exponents (FTLE) seed particles densely in a volume and examine where these particles end up. For these operations, and many others, only the ending position of the particle is needed, and not the details of the path the particle took to get there.

Despite seemingly abundant computational power, some techniques have excessively long running times. For example, ocean modelers often study the FTLE within an ocean with both high seeding density and very long durations for the particles (years of simulation time) [11, 13]. As another example, fusion scientists are interested in FTLE computations inside a tokamak where particles travel for hundreds of rotations [19]. In both cases, FTLE calculations, even on supercomputers, can take tens of minutes.

With this work, we consider an alternative to traditional Eulerian advection. The key observation that motivates the work is that, in conditions with dense seeding and long durations, particles will tread the same (or very similar) paths over and over. Where the current paradigm carries out the same computation over and over, we consider a new paradigm where a computation can be carried out a single time, and then reused. That said, we find that, while particle trajectories do often travel quite close to each other, they typically follow their own (slightly) unique paths. Therefore, to effectively reuse computations, we consider a method where we interpolate new trajectories from existing ones, effectively trading accuracy for speed.

Our method depends on Block Exterior Flow Maps, or BEFMs. The idea behind BEFMs is, for block-decomposed data, to precompute known trajectories that lie on block boundaries. When a compute-intensive flow visualization algorithm is then calculated, it consults with the BEFMs and does Lagrangian-style interpolation from its known trajectories. While this approach introduces error, it can be considerably faster, since it avoids Eulerian advection steps inside each block.

The contributions of the paper are as follows:

- Introduction of BEFMs as an operator for accelerating dense particle advection calculations;
- A novel method for generating an approximate BEFM that can be used in practice;
- A study that evaluates the approximate BEFM approach, including comparisons with traditional advection.

2 RELATED WORK

McLouglin et al. recently surveyed the state of the art in flow visualization [9], and the large majority of techniques they described incorporate particle advection. Any of these techniques could possibly benefit from the BEFM approach, although the tradeoff in accuracy is only worthwhile for those that have extreme computational costs, e.g., Line Integral Convolution [4], finite-time Lyapunov exponents [6], and Poincare analysis [18].

One solution for dealing with extreme advection workloads is parallelization. A summary of strategies for parallelizing particle advection problems on CPU clusters can be found in [15]. The basic approaches are to parallelize-over-data, parallelize-over-particles, or a hybrid of the two [14]. Recent results using parallelization-over-data demonstrated streamline computation on up to 32,768 processors and eight billion cells [12]. These parallelization approaches are complementary with our own. That is, traditional parallel approaches can be used in the current way, but the phase where they advect particles through a region could be replaced by our BEFM approach.

In terms of precomputation, the most notable related work comes from Nouanesengsy et al. [10]. They precomputed flow patterns within a region and used the resulting statistics to decide which regions to load. While their precomputation and ours have similar elements, we are using the results of the precomputation in different ways: Nouanesengsy et al. for load balancing and ourselves to replace multiple integrations with one interpolation.

In terms of accelerating particle advection through approximation, two works stand out. Brunton et al. [3] also looked at accelerating FTLE calculation, but they considered the unsteady state problem, and used previous calculations to accelerate new ones. While this is a compelling approach, it does not help with the steady state problem we consider. Hlwatsch et al. [7] employ an approach where flow is calculated by following hierarchical lines. This approach is well-suited for their use case, where all data fits within the memory of a GPU, but it is not clear how to build and connect hierarchical lines within a distributed memory parallel setting.

In contrast, our method, by focusing on flow between exteriors of blocks, is well-suited for this type of parallelism.

Bhatia et al. [2] studied edge maps, and the properties of flow across edge maps. While this work clearly has some similar elements to our, their focus was more on topology and accuracy, and less on accelerating particle advection workloads.

Scientific visualization algorithms are increasingly using Lagrangian calculations of flow. Jobard et al. [8] presented a Lagrangian-Eulerian advection scheme which incorporated forward advection with a backward tracing Lagrangian step to more accurately shift textures during animation. Salzbrunn et al. delivered a technique for analyzing circulation and detecting vortex cores given predicates from pre-computed sets of streamlines [17] and pathlines [16]. Agranovsky et al. [1] focused on extracting a basis of Lagrangian flows as an *in situ* compression operator, while Chandler at al. [5] focused on how to interpolate new pathlines from arbitrary existing sets. Of these works, none share our focus on accelerating advection.

3 METHOD

Our method makes use of *block exterior flow maps* (**BEFM**). We begin by defining this mapping, in Section 3.1. We then describe our method, and how it incorporates these maps, in Section 3.2. Finally, in Section 3.3, we present some analysis of the computational complexity of our method, compared to the traditional technique.

3.1 Block Exterior Flow Map

3.1.1 Definition

In scientific computing, parallel simulation codes often partition their spatial volume over their compute nodes. Restated, each compute node will operate on one spatial region, and that compute node will be considered the "owner" of that region. Such a region is frequently referred to as a *block*. For example, a simulation over the spatial region X: [0-1], Y: [0-1], and Z: [0-1] and having N compute nodes would have N blocks, with each block covering a volume of $\frac{1}{N}$.

Consider a point P that lies on the exterior of a block B. If the velocity field points toward the interior of B at point P, then Eulerian advection of a particle originating at P will take the particle through the interior of B until it exits. In this case, the particle will exit B at some location P', where P' is also located on the exterior of B. The BEFM captures this mapping. The BEFM's domain is all spatial locations on the exterior of blocks, and its range is also spatial locations on the exteriors of blocks. Further, for any given P in the BEFM's domain, $BEFM(P,B)$ will produce a location that is on B's exterior. Saying it concisely, the BEFM is the mapping from particles at exteriors of blocks to the locations where those particles will exit the block under Eulerian advection. Figure 1 illustrates an example of a BEFM.

3.1.2 Using BEFMs for Calculating Particle Trajectories

Now consider a particle P that lies on the interior of block B_0. Further, consider the trajectory of P when traveling for T time units. Assume P travels through blocks B_1, B_2, ..., B_{N-1}, before terminating in the interior of block B_N at time T. Consider how BEFMs can be used to calculate P's trajectory:

- Since P lies in the interior of B_0, traditional advection is needed to calculate the path of P until it reaches B_0's exterior.
- The BEFM can then be used to calculate the path of P through B_1, B_2, ..., B_{N-1}.
- P's trajectory into the interior of B_N is then again calculated with traditional advection.

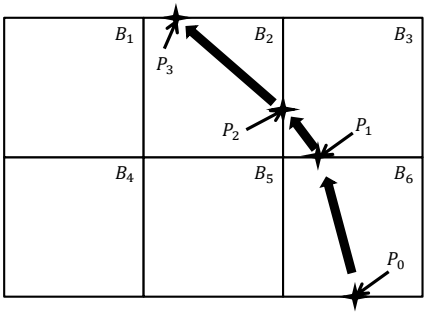

Figure 1: Notional example of a BEFM on a two-dimensional vector field. This example shows the path of a particle moving through a region, with an emphasis on the blocks it travels through. Particle P_0 travels through block B_6 and exits B_6 at location P_1. Thus, $BEFM(P_0,B_6) = P_1$. Similarly, $BEFM(P_1,B_3) = P_2$, and $BEFM(P_2,B_2) = P_3$ etc. In the case of particles placed in an outgoing region of flow, the BEFM returns the particle itself, e.g., $BEFM(P_1,B_6) = P_1$.

Putting it all together, if BEFMs can calculate mappings more quickly than the calculations for advecting a particle through a block, then this method should be faster than traditional advection. Further, the speedup for the BEFM-style calculation is then limited only by the cost for the steps through the initial and final blocks (B_0 and B_N).

3.1.3 Approximate BEFMs

There are many ways to implement a BEFM. For example, a BEFM could respond to each mapping request (i.e., a $BEFM(P,B)$) by going back to the original vector field and employing traditional advection. In this case, the BEFM would have the same performance characteristics as traditional advection, and the abstraction of BEFMs on top of traditional advection would be unnecessarily complicated.

For our research, we are interested in BEFMs where each mapping request can be satisfied much more quickly than the work it takes to advect a particle using traditional advection. For this reason, we consider precomputation, i.e., evaluating the BEFM before the main work begins of calculating particle trajectories. However, it is not obvious how to precompute a perfect BEFM. Our approach to this problem is to precompute an approximate BEFM or **ABEFM**. This ABEFM will know the exact mappings for certain locations on the boundary. We refer to this list of locations as the KnownParticleList.

When an ABEFM is asked to calculate mappings for particles that are not in the KnownParticleList, it will interpolate the exit location from the nearest particles that are in the KnownParticleList.

There are many ways to establish an ABEFM's KnownParticleList. We chose to generate locations uniformly along the exterior of a block at some chosen sample density. With this approach, the accuracy and pre-computation time are in tension. High sample densities will increase accuracy at the cost of pre-computation time. Low sample densities will reduce pre-computation time at the cost of accuracy. We explore this issue more in Section 5.

3.1.4 Conditions Where an ABEFM Cannot Be Used

It is not always possible to interpolate new trajectories from the ABEFM's known trajectories. Through our experiments, we have identified three ways in which interpolation is not possible. They are:

1. If a particle trajectory from the exterior of block B never again reaches the exterior of block B, i.e., if a particle lands in a sink or is caught in a vortex inside the block.

2. If a particle trajectory differs too significantly from its neighbors, i.e., neighboring trajectories separated and exit through different faces of B.
3. If all neighboring trajectories are not uniformly entering the block or uniformly exiting the block, e.g., some neighboring particles get displaced to the interior of the block while others are displaced into neighboring blocks.

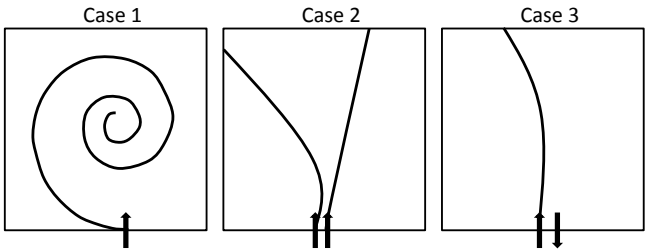

Figure 2: Cases where an ABEFM cannot interpolate a new trajectory. Note that on the right figure one of the particles enters the block while the other particle exits the block.

Fortunately, we can detect each of these three cases, and fall back to traditional advection to determine a particle trajectory. However, it is important that we understand the rate at which these conditions occur. The rate is data set dependent, and we determine these rates experimentally.

3.2 An Approach for Creating and Using an ABEFM

In this section we describe our algorithms for creating an ABEFM and utilizing an ABEFM for advection.

Examples in this outline will follow the assumption that ABEFM's KnownParticleList points are uniformly generated at the mesh resolution, i.e., one particle trajectory for every node in the mesh that lies on the exterior of a block. For example, in a 10×10 two-dimensional mesh with 4 blocks laid in a 2×2 pattern, each block's external edge will consist of 5 cells and therefore 6 points. Additionally, these mapped points are not duplicated across shared faces. Figure 3 illustrates this example.

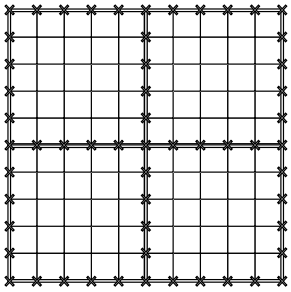

Figure 3: Initial locations for particle trajectories to be mapped during the pre-computation phase of an ABEFM. Depicted is a 10x10 mesh with 2x2 blocks overlaid and the locations of the mappings defined on the block's exteriors

3.2.1 Building an ABEFM

ABEFM construction consists of generating flows for each location in the KnownParticleList. This is done by initializing particles at each location in the KnownParticleList and then advecting those particles across a block. Advection is done using traditional Eulerian methods such as Runge-Kutta. Pseudo code for this method is outline in Algorithm 1.

Algorithm 1 Build Flow Map

```
1: function GET BLOCK ID(Particle P)
2:     Determine the block that P advects through
3:     return BlockID
4: end function
5: function ADVECT ON BLOCK(Particle P, Block B)
6:     Advect P until it exits B (using Eulerian advection)
7:     Stop P on boundary of B
8:     Compute which Face of B that P landed on
9:     return P, FaceID
10: end function
11: for all P in KnownParticleList do
12:     Bid = GET BLOCK ID(P)
13:     NewP, Fid = ADVECT ON BLOCK(P, Bid)
14:     Def: Flow F as the set < P, NewP, Bid, Fid >
15: end for
```

3.2.2 Advecting With an ABEFM

Section 3.1.2 describes how to use a BEFM for particles at arbitrary locations in a volume. For this discussion, we focus on the case of a particle P that lies on the boundary of block B, and calculating where P exits B.

The trajectory for a particle P is calculated as follows. First, the neighboring particles, P_1, P_2, ... , P_n (P_i) from the KnownParticleList are identified. For our study, the KnownParticleList had particles seeded at regular intervals, so n would be four, and we would find the four particles that formed a square around P. Next, we check the P_i for our three conditions where an ABEFM cannot be used (see 3.1.4). If we cannot use the P_i, then we fall back to traditional Eulerian advection using Runge-Kutta solves. If we can use the P_i, then we take the output location to be the weighted average of the exit locations of the P_i. For our construction of four P_is in a square configuration, this entailed bilinear interpolation. We also interpolated the time to advance through the volume from the times of the P_is. If this time was greater than the amount of time remaining for the particle to travel, then we rejected the interpolated result (since it traveled too far), and fell back to Eulerian advection. However, if the interpolated projection was within the time bounds, then we used it and avoid Eulerian advection. Pseudo code for this method is outline in Algorithm 2.

Algorithm 2 Advect with Flow Map

```
1: function ADVECT BLOCK(Particle P, Block B)
2:     Integrate to find P's exit location
3:     Stop P on boundary of B
4:     if P.Time ≥ End Time then
5:         return 0
6:     else
7:         return 1
8:     end if
9: end function
10: function ADVECT VIA FLOW MAP(Particle P, Block B)
11:     Interpolate Output location and time from (P,B)
12:     if Output.Time > End Time then
13:         return ADVECT BLOCK(P,B)
14:     end if
15:     Set P = Output
16:     return 1
17: end function
```

```
18: AdvectionList: List of particles to be advected
19: for all Particles P in AdvectionList do
20:     keepGoing = 1
21:     while P.time < End Time && keepGoing do
22:         Bid = GET BLOCK ID(P)
23:         if Particle on Computable Face then
24:             keepGoing = ADVECT VIA FLOW MAP(P,Bid)
25:         else
26:             keepGoing = ADVECT BLOCK(P, Bid)
27:         end if
28:     end while
29: end for
```

3.3 Computational Analysis

Our method can only be used to accelerate flows under certain conditions, and those conditions are data dependent. Therefore, our discussion of the method in this section can only be presented in terms of probabilities. The Results sections will demonstrate the accuracy we can achieve, and the performance speedups we observe in real-world settings.

Looking closely at each sub-component of the ABEFM algorithm in comparison too the traditional Eulerian approach, helps to clarify the differences between these algorithm's complexities and run times. For this analysis we assume a full mesh resolution advection problem, advecting one particle for every node in the mesh, for integration time T. Also, we assume a cubic mesh with N cells per dimension and cubic blocks with B blocks in each dimension. For simplicity we also assume large N so that we can approximate the N+1 vertices in each dimension as N vertices. In order to make certain simplifying assumptions we will also consider the case where our underlying vector field is pointing in exactly one direction everywhere. This assumption should work for many fields as an average. This analysis is not intended to produce exact answers; simply to provide a general guideline for performance as well as to increase our understanding of the underlying connections between components of our model.

$$\begin{aligned} Mesh &:= N \times N \times N \\ Blocks &:= B \times B \times B \\ Points\ per\ Block &:= \frac{N}{B} \times \frac{N}{B} \times \frac{N}{B} \end{aligned}$$

We will start by breaking the traditional method down into its components. If we are advecting a full mesh resolution number of particles, for some integration time, T, with a step size of S, and the cost of a single Eulerian update step is E, then the total advection time for the traditional method is:

$$\begin{aligned} Time\ to\ Advect\ E &:= N \times N \times N \times \frac{T}{S} \times E \\ &:= N^3 \times \frac{T}{S} \times E \end{aligned}$$

If we do a similar break down of the run times for an ABEFM, we will need to consider both phases and each of the important subcomponents mentioned in Section 3.2.

For the phase where we generate an ABEFM, we will need to consider:

$$[\frac{Faces}{Block}] \times [\#Blocks] \times [\frac{\#Points}{Face}] \times [\#Steps] \times [\frac{Time}{Step * Point}]$$

$$\begin{aligned} Time\ to\ Build\ Flows &:= [6] \times [B^3] \times [\frac{N^2}{B^2}] \times [\frac{N}{B}] \times [E] \\ &:= 6 \times N^3 \times E \end{aligned}$$

This expression over counts the total number of flows. Therefore, we can consider this as an upper bound for this phase's run time. Fixing this would effectively make the constant, 6, a smaller constant.

For the first and last parts of phase two, advecting with an ABEFM, we can estimate that the time to advect all of the particles to a face will be roughly equal to the time to advect all of the particles through their last Eulerian advection steps. Additionally, on a constant flow, the average time to advect a particle to a face, from any point in the mesh, will be the time it takes to advect that particle half way through a block. For all particles this can be expressed as:

$$Time\ to\ find\ Start/End := N^3 \times \frac{1}{2} \times \frac{N}{B} \times E$$

Finally the time to advect all particles with the ABEFM, can be expressed as a probability function. Probability, P, is the probability that we will use the ABEFM to interpolate to a new location instead of falling back to traditional advection. The following equation represents the total time for the ABEFM update steps to run on all of the Particles for time T, given that I is the time to do one interpolation update.

$$\begin{aligned} ABEFM\ Update &:= N^3 \times \left((I \times P) + (\frac{N}{B} \times E \times (1-P)) \right) \times \frac{T/S}{N/B} \\ &:= \left[(N^2 \times \frac{T}{S} \times I) \times P \right] + \left[(N^3 \times \frac{T}{S} \times E) \times (1-P) \right] \end{aligned}$$

Which in words is the total number of points, N^3, times the total number of steps, $\frac{T}{S}$, divided by the number of steps per block, $\frac{N}{B}$, times the sum of the time to do one block interpolation update times the probability to do an update, $(I \times P)$, plus the time to do one Eulerian update times the probability to an Eulerian update times the number of steps in one block, $E \times (1-P) \times \frac{N}{B}$.

Something interesting to note here is that the ABEFM update step, if P = 1, reduces the problem by a whole order of N, which is a significant decrease in number of computations, O(N^3) to O(N^2).

If we wish to compare the total time for each algorithm we can simply add together the portions of the ABEFM method and compare them to the Eulerian method. If we set the equation such that the ABEFM method is faster than the Eulerian method, Eulerian > ABEFM, then we can solve for any interesting parameters.

$$\begin{aligned} Eulerian &> ABEFM \\ N^3 \times \frac{T}{S} \times E &> \left[6 \times N^3 \times E \right] + \\ &\quad \left[2 \times N^3 \times \frac{1}{2} \times \frac{N}{B} \times E \right] + \\ &\quad \left[\left[(N^2 \times \frac{T}{S} \times I) \times P \right] + \left[(N^3 \times \frac{T}{S} \times E) \times (1-P) \right] \right] \\ N^3 \times \frac{T}{S} \times E &> N^4 \frac{E}{B} + N^3 E \left(6 + \frac{T}{S} \right) + N^2 \frac{TP}{S} (I - NE) \end{aligned}$$

This Equation Simplifies too:

$$\frac{T}{S} \times (N \times E - I) \times P > N \times \left(6 + \frac{N}{B} \right) \times E$$

$$ASSERT:$$

$$\begin{array}{cccc} N > 1 & T > 0 & E > 0 & I > 0 \\ 1 < B \leq N & 0 < S \leq 1 & 0 \leq P \leq 1 & \end{array}$$

There are two possible cases for the $(NE-I)$ portion of the equation. And looking at both closely we can see that:

$$\begin{aligned} if: &\quad NE < I \\ then: &\quad (NE-I) = -(I-NE) \\ AND: &\quad P < -\left(\frac{S}{T}\frac{N(6+(N/B)E}{(I-NE)}\right) \end{aligned}$$

Given our assert that P is a probability between 0 and 1, and given the combinations of the the asserts that make the expression on the right hand side positive, this equation is not possible. This means that given this formulation, the ABEFM approach is not faster in the space where $NE < I$. Given that I and E are both roughly small constants this places a bound on the size of N required to make doing an ABEFM useful.

3.4 Discussion

When trying to understand a new method it is important to know when the method will be useful and when it will not. We can gain some basic understanding of this from the algorithmic analysis. The analysis makes certain assumptions however and therefore can only be relied on to help us gain a basic understanding and set up some bounds on the useful range of the method.

Using this analysis we can look at some real world problems and decide if this method is worth applying.

N	B	T	S	P	E	I	$E_{time} > ABEFM$
100	3	10	0.001	.8	0.01	0.1	1
1	3	10	0.001	.8	0.01	0.1	0
10	3	10	0.001	.8	0.01	0.1	0
11	3	10	0.001	.8	0.01	0.1	1
100	3	10	0.1	.8	0.01	0.1	1
100	3	10	0.2	.8	0.01	0.1	0

4 Study Overview

4.1 Data Sets

We considered three data sets. Each had steady state flow (i.e., one time slice) and was defined on a regular mesh. They are:

- **Tokamak**: the magnetic field inside a tokamak. Inside the tokamak, the velocity vector values lead to circulation around the tokamak. Outside the tokamak, the velocity field is all zero vectors. This data set had dimensions 300^3.
- **Astro**: a supernova simulation. The vector field has high variability in its central spherical region, and steadily points out or in when approaching the edges. This data set had dimensions 256^3.
- **TH**: a thermal hydraulics simulation of air mixing in a "fish tank" box with two inlets — one with hot air and one with cold air — and an outlet. This data set had dimensions 500^3.

4.2 Testing Factors

We considered six dimensions of configurations:

- Domain block layout: what are the impacts of having fewer or more blocks?
- Density of known particles: what are the impacts in calculating more or less particles during preprocessing? — time for preprocessing, time for regular execution, and accuracy?
- Integration time: how does performance and accuracy change as particles go for shorter or longer periods?
- Step size: how does step size affect performance and accuracy?
- Number of particles: how does the number of particles to be processed affect overall run times?
- Data set: how does the underlying vector field affect performance and accuracy?

4.3 Testing Methodology

Our methodology consisted of seven phases. The first phase studied our "default" case in detail. Each of the remaining six phases sweep through one dimension of our testing factors, and explores the impact of that factor.

4.3.1 Phase 1: Default Workload

Our default case consists of a workload, and a configuration for ABEFM. The default workload was 20^3 particles integrating for 5 time units with a step size of 0.001 on the vector field from the Tokamak data set. The default ABEFM configuration on the Tokamak was (5x5x5) blocks and 300 particles precomputed in each dimension for the KnownParticleList.

4.3.2 Phase 2: Block Layout

With this phase, we wanted to understand the effects of changing block size. Large blocks cause particles to travel larger distances, but the interpolated path may be less accurate. Small blocks cause particles to travel shorter distances – and so the number of operations needed to go the same distance is greater – but the interpolated path may be more accurate. With this phase, we wanted to understand the magnitude of these effects.

We considered 8 block layouts: (5x5x5), (10x10x10), (15x15x15), (20x20x20), (25x25x25), (30x30x30), (40x40x40), and (50x50x50). Additionally we use the outcome from this phase to focus a more detailed look at a few more layouts.

4.3.3 Phase 3: Density of Known Particles

With this phase, we wanted to understand the effects of changing the number of known particles in the precomputation phase. Increasing this density will increase accuracy and the ability to use an ABEFM, but also increases precomputation costs. Decreasing this density could impact accuracy and decrease the ability to use an ABEFM, but reduces precomputation costs. With this phase, we again wanted to understand the magnitude of these effects.

We considered 5 densities along each dimension of the mesh: 100, 200, 300, 400, and 500.

4.3.4 Phase 4: Integration Time

With this phase, we considered integration time. Short integration times imply that we spend the majority of our time using traditional advection to get to block boundaries, mitigating the opportunity for speedup. Longer integration times, however, create the potential for applying the ABEFM repeatedly, and possibly significant speedups.

We considered 7 integration times: 1, 5, 10, 20, 40, 80, and 100 time units.

4.3.5 Phase 5: Step Size

With this phase, we considered step size. Small steps sizes move more slowly through a volume, while large step sizes move more quickly. However, for the ABEFM method, the step sizes only impact performance for stepping to the boundary, so the principal change is in the comparison with traditional advection.

We considered 9 step sizes: 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, and 0.00001.

4.3.6 Phase 6: Number of Particles

With this phase, we considered the number of particle trajectories to calculate. As this number becomes large, the cost for precomputation is amortized, making ABEFMs more effective.

We consider configurations where the particles were placed in a cubic formation of evenly spaced samples. We considered 3 resolutions: 20^3, 100^3, and 300^3.

Time (seconds)	
ABEFM Build Time	54.883
ABEFM Run Time	26.185
Eulerian Run Time	145.984
Speedup	
ABEFM Run Time over Eulerian Time	5.573
ABEFM Total Time over Eulerian Time	1.801
Usability	
Percent Usable Faces	90.4847%
Percent ABEFM Jumps	84.610%
Error	
Average Percent Error	0.991%
Average Displacement Distance	0.00657

Table 1: Results from Phase 1.

4.3.7 Phase 7: Data Set

The performance of the ABEFM can clearly be affected by the underlying vector field. With this phase, we considered all three data sets. We performed the study from Phase 2 on each of the data sets keeping the total number of Eulerian steps constant. The Tokamak data set values are already listed in phase 2. The Astro data set used an integration time of 5000 and a step size of 1. The TH data set used the same configuration as the Tokamak data set. Each data set used their own native resolution for precomputed particles for the KnownParticleList: 256 for Astro and 500 for FH. Each considered workloads of 20^3 particles.

4.4 Hardware

All studies were performed on a machine with a 2.5 GHz E5-2609 v2 Intel Xeon processor and 64 GB of RAM. This initial study was done in serial, since the serial results will enable direct comparisons between the ABEFM approach and traditional Eulerian integration.

4.5 Measurements

The measurements we took for each experiment were:

- Time: the total run time of the ABEFM approach (meaning both build time and advection time using the ABEFM). We also would run a separate experiment with the traditional, Eulerian approach and measure its time.
- Speedup: the total speed up from using an ABEFM compared to just Eulerian integration.
- Usability Metric: the percentage of time spent interpolating with the ABEFM versus using Eulerian integration and the percentage faces that an ABEFM can be used.
- Error: the average percent error and average displacement distance of an FTLE computed with the ABEFM advected particles with respect to the Eulerian advected particles. Differences in FTLE are used as the error metric since one of the leading motivations for ABEFMs use is the FTLE.

5 Results

5.1 Phase 1: Single Test Analysis

Phase 1 delves into a single test, to set baseline expectations for the ABEFM method, and how the ABEFM method compares with traditional Eulerian advection. Table 1 shows the key results from this phase.

This baseline test demonstrates the viability of the ABEFM approach. Although the precomputation is non-trivial, it is still much smaller than the time to perform Eulerian advection. Additionally, the errors incurred were minimal — only about 1% different from the Eulerian value and 0.006 different in the actual FTLE values.

Figure 4 shows the FTLE computed using both the ABEFM method and traditional Eulerian advection.

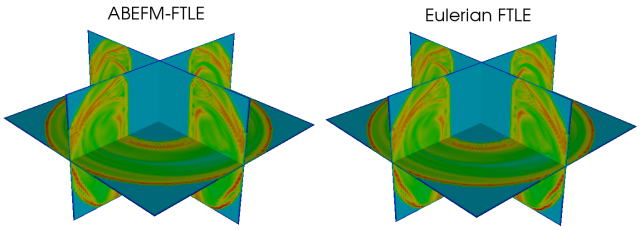

Figure 4: The FTLE field computed using the ABEFM (left) and the Eulerian advection technique (right).

5.2 Phases 2: Varying Domain Block Layouts

This phase studies the effect on ABEFM calculations when dividing the mesh into different numbers of blocks. Figure 5 shows tradeoffs in accuracy and speedup as the number of blocks increases. It shows that with the lowest number of blocks, both speedup and accuracy is quite good. By going to 10^3 or 15^3, the speedup is maintained, but error increases. Ultimately, when going even higher, speedups drop off, but error is also reduced.

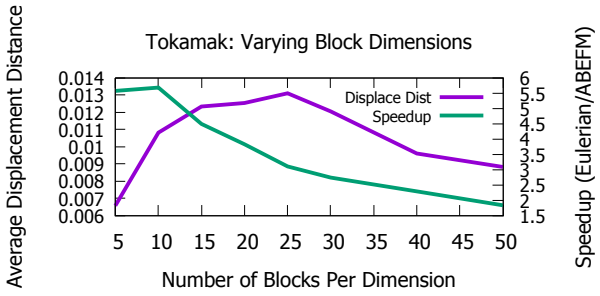

Figure 5: Results from Phase 2: The accuracy and runtime for the Tokamak data set with respect to varying block dimensions.

A subsequent study considered additional block layouts, tailored to the nature of the Tokamak data set, and its circular flow. In this case, blocking occurred along the flow, but there was no vertical blocking, since flow moves horizontally. Figure 6 shows the results of this study.

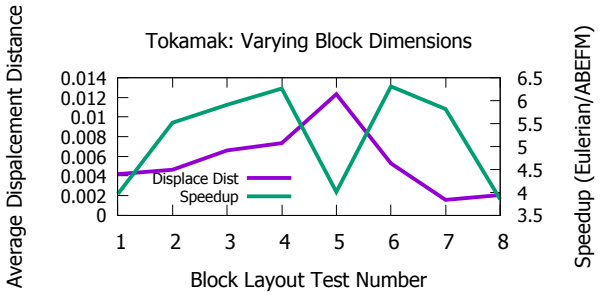

Figure 6: Results from Phase 2: accuracy and runtime of the Tokamak data set with respect to additional block dimensions. Block dimensions are as follows: [1] = (2x2x2), [2] = (3x3x3), [3] = (4x4x4), [4] = (5x5x5), [5] = (20x20x20), [6] = (20x20x1), [7] = (3x3x1), [8] = (2x2x1)

This configuration confirms that fastest run times with the Tokamak data set come from the block layouts with smaller numbers. However, it also shows that, for this data set, it is beneficial to reduce the number of blocks in Z with respect to the number of blocks in X and Y. Before these studies, our initial intuition was that the closer the block jumps are the less error there will be, but this was not true here. Having less block jumps can also decreases the error as there are less interpolations that are approximating the flows locations and/or there is a greater percent of Eulerian updates. This displays a trade-off between the number of times an error is introduced versus the size of the errors introduced.

5.3 Phase 3: Vary KnownParticleList Density

Phase 3 varied the density of the KnownParticleList. Figure 7 shows the tradeoffs between accuracy and build time for the ABEFM as a function of KnownParticleList density. For this test, the percentage error decreased significantly faster when the density increased from below the mesh resolution up to the mesh resolution. Then, as the KnownParticleList density is increased even more, there is an improvement in accuracy, although not as significantly as before. Additionally, the time to build the ABEFM grew slower and slower from 100 to 300 to 500, starting at 8.7s, going to 61.9s, and then to 184s.

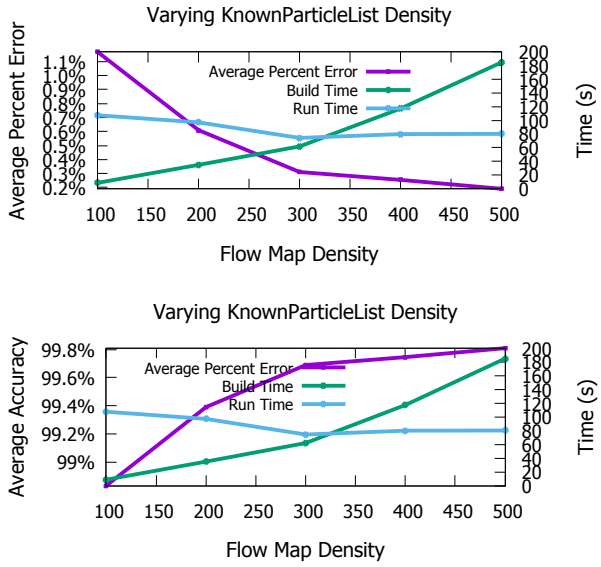

Figure 7: Results from Phase 3: accuracy and runtimes as a function of KnownParticleList density. Run time drops slightly and build time increases steadily as the density of known (precomputed) particles is increased.

5.4 Phase 4: Integration Time

This phase looked at performance and accuracy as particles were allowed to travel for longer and longer distances. Figures 8 and 9 show the results of this study. The take away from these figures is that, as integration time increases, the speedups from the ABEFM method become increasingly higher. While this is expected, the study shows the extent of speedup that is possible, which is ultimately limited by the number of faces along a block that can be used for interpolation (and thus do not have to fall back to Eulerian advection).

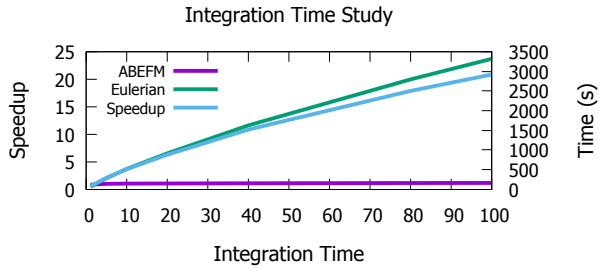

Figure 8: Results from Phase 4: runtimes and speedup for both the ABEFM and Eulerian methods as a function of integration time. Values for run time of the ABEFM method vary from 115 seconds to 159 seconds with a 62 second construction time. The Eulerian run time varies from 64 seconds to 3320 seconds.

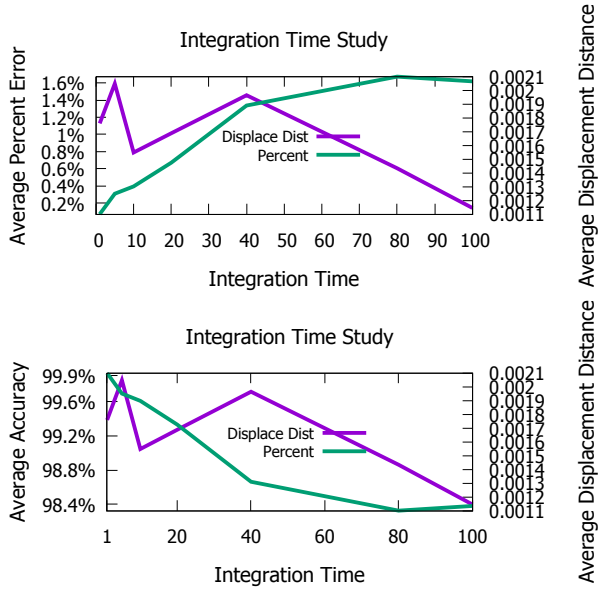

Figure 9: Results from Phase 4: average absolute error and average percent error as a function of integration time.

5.5 Phase 5: Step Size

This phase varied the step size. Figures 10 and 11 show the results of this phase.

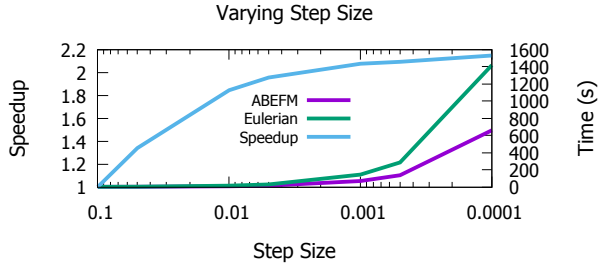

Figure 10: Results from Phase 5: runtimes and speedup for the ABEFM method and Eulerian Method as a function of step size.

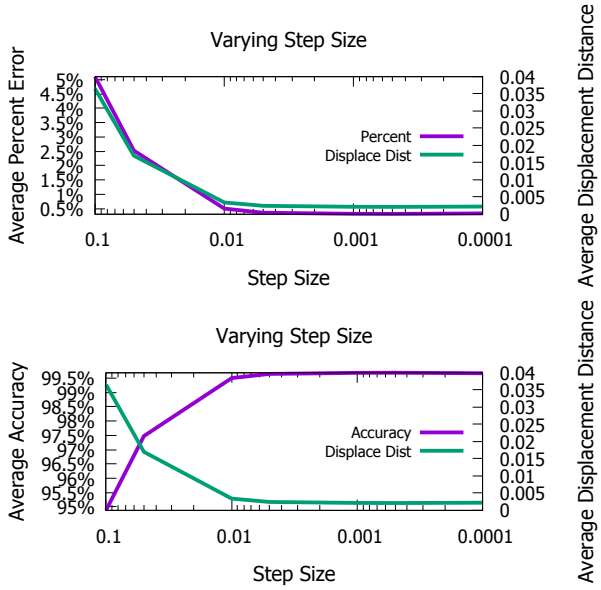

Figure 11: Results from Phase 5: average absolute error and average percent error as a function of step size.

Step size affects both the Eulerian method and the Eulerian portions of the ABEFM preprocessing phase. As step size decreases, the speedup increases, although it appears to be asymptotically bound.

5.6 Phase 6: Varying Number of Particles

This phase shows the effects of varying the number of particles. As the number of particles increases, the precomputation costs for the ABEFM are increasingly amortized. Figure 12 shows that as the number of particles is increased, the speedup also increases.

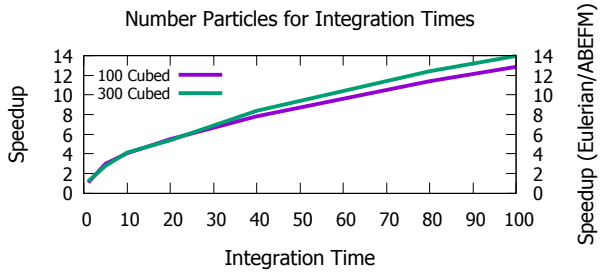

Figure 12: Results from Phase 6: The Speedups of 100 and 300 cubed particles as integration time is increased.

5.7 Phase 7: Varying The Data Set

This study incorporated the remaining two data sets (TH and Astro) to see how well they performed compared to the Tokamak data set. The data sets were studied with a variety of blocks (i.e., the same study that was performed in Phase 2, but for these new data sets). For reference, the Tokamak data set's results for this analysis were listed in Figure 5.

In the Astro data set, there is significant mixing in the center and headed straight out or straight in on the edges. The result of varying block dimension can be seen in Figure 13. It shows an optimal layout for run time at around 20^3 resolution of blocks. The accuracy at this level is also at a local minimum though it is greater then at smaller block sizes. The accuracy also seems to level off at around this level for all of the next tested block sizes.

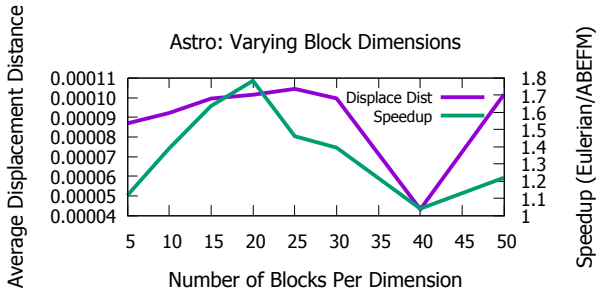

Figure 13: Results from Phase 7: accuracy and runtime for the astro data set as a function of varying block dimensions.

The second data set, TH, captures the mixing of hot and cold air currents. The vector field for this data set has significant mixing throughout its volume. The results of varying block dimension can be seen in Figure 14. The optimal layout for runtime is at around a block resolution of 15^3 to 20^3. Between these two values, 20^3 has lower error. As the number of blocks increases or decreases, the errors go down but so too does the speedup.

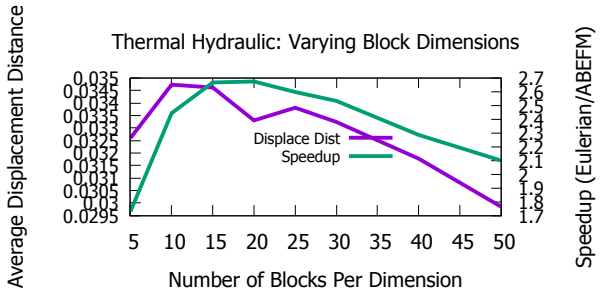

Figure 14: Results from Phase 7: accuracy and runtime for the TH data set as a function of varying block dimensions.

6 Conclusion and Future Work

We introduced Block Exterior Flow Maps (BEFMs) and designed an algorithm for accelerating flow calculations using Approximate BEFMs (ABEFMs). The approach has two significant "knobs" — block layout and density of known particles calculated in the preprocessing phase — and we studied the impacts of these knobs for multiple particle advection workloads. We found that ABEFMs provided significant winnings for extreme particle advection workloads, with one workload completing in 159 seconds where the traditional approach took 3,320 seconds, a speedup of more than 20X and with an average error of less than 2%. Further, as particles are advected for longer and longer distances, our technique has the possibility to show even greater gains.

This technique was developed in response to needs within the fusion community to advect for long periods around a tokamak. While our technique is currently useful for stand-alone *post hoc* analysis, our future work will be to insert the method into their simulation codes for *in situ* processing. While our preprocessing times are currently large, we believe they can be accelerated on the many-core architectures now prevalent on top supercomputers. Further, our block-centric approach lends itself well to distributed memory parallelism. In another branch of future work, we would like to consider constructing the ABEFM adaptively, in an effort

to minimize unneeded calculations, and to increase resolution in complex flow regions.

Acknowledgements

The authors wish to thank Christoph Garth Ph.D of University of Kaiserslautern for his inspiration of the originoal idea, Linda Sugiyama Ph.D of Massachusetts Institute of Technology for providing the underlying problem to solve as well as a data set for testing and validation.

References

[1] A. Agranovsky, D. Camp, C. Garth, E. W. Bethel, K. I. Joy, and H. Childs. Improved Post Hoc Flow Analysis Via Lagrangian Representations. In *Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV)*, pages 67–75, Paris, France, Nov. 2014.

[2] H. Bhatia, S. Jadhav, P. Bremer, G. Chen, J. A. Levine, L. G. Nonato, and V. Pascucci. Flow visualization with quantified spatial and temporal errors using edge maps. *Visualization and Computer Graphics, IEEE Transactions on*, 18(9):1383–1396, 2012.

[3] S. Brunton and C. Rowley. A method for fast computation of ftle fields. In *APS Division of Fluid Dynamics Meeting Abstracts*, volume 1, 2008.

[4] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 263–270, New York, NY, USA, 1993. ACM.

[5] J. Chandler, H. Obermaier, K. Joy, et al. Interpolation-based pathline tracing in particle-based flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 21(1):68–80, 2015.

[6] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248 – 277, 2001.

[7] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *Visualization and Computer Graphics, IEEE Transactions on*, 17(8):1148–1163, Aug 2011.

[8] B. Jobard, G. Erlebacher, and M. Hussaini. Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 8(3):211–222, Jul 2002.

[9] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. In *EuroGraphics 2009 - State of the Art Reports*, pages 73–92, April 2009.

[10] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-Balanced Parallel Streamline Generation on Large Scale Vector Fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011.

[11] T. M. Özgökmen, A. C. Poje, P. F. Fischer, H. Childs, H. Krishnan, C. Garth, A. C. Haza, and E. Ryan. On Multi-Scale Dispersion Under the Influence of Surface Mixed Layer Instabilities. *Ocean Modelling*, 56:16–30, Oct. 2012.

[12] T. Peterka, R. Ross, B. Nouanesengsey, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang. A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields. In *Proceedings of IPDPS 11*, Anchorage AK, 2011.

[13] L. Pratt, I. Rypina, T. Özgökmen, P. Wang, H. Childs, and Y. Bebieva. Chaotic Advection in a Steady, Three-Dimensional, Ekman-Driven Eddy. *Journal of Fluid Mechanics*, 738:143–183, Jan. 2014.

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

[15] D. Pugmire, T. Peterka, and C. Garth. Parallel Integral Curves. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 91–113. Oct. 2012.

[16] T. Salzbrunn, C. Garth, G. Scheuermann, and J. Meyer. Pathline predicates and unsteady flow structures. *The Visual Computer*, 24(12):1039–1051, 2008.

[17] T. Salzbrunn and G. Scheuermann. Streamline predicates. *Visualization and Computer Graphics, IEEE Transactions on*, 12(6):1601–1612, Nov 2006.

[18] A. R. Sanderson, G. Chen, X. Tricoche, D. Pugmire, S. Kruger, and J. Breslau. Analysis of recurrent patterns in toroidal magnetic fields. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1431–1440, 2010.

[19] L. Sugiyama and H. Krishnan. Finite time lyapunov exponents for magnetically confined plasmas. *Bulletin of the American Physical Society*, 57, 2012.