A Parallel Arbitrary-Order Accurate AMR Algorithm for the Scalar Advection-Diffusion Equation

Arash Bakhtiari*, Dhairya Malhotra†, Amir Raoofy*,
Miriam Mehl‡, Hans-Joachim Bungartz* and George Biros†
*Technical University of Munich, Munich, Germany
‡University of Stuttgart, Stuttgart, Germany
†University of Texas at Austin, Austin, TX 78712

***Abstract*—We present a numerical method for solving the scalar advection-diffusion equation using adaptive mesh refinement. Our solver has three unique characteristics: (1) it supports arbitrary-order accuracy in space; (2) it allows different discretizations for the velocity and scalar advected quantity; (3) it combines the method of characteristics with an integral equation formulation; and (4) it supports shared and distributed memory architectures. In particular, our solver is based on a second-order accurate, unconditionally stable, semi-Lagrangian scheme combined with a spatially-adaptive Chebyshev octree for discretization. We study the convergence, single-node performance, strong scaling, and weak scaling of our scheme for several challenging flows that cannot be resolved efficiently without using high-order accurate discretizations. For example, we consider problems for which switching from 4th order to 14th order approximation results in two orders of magnitude speedups for a computation in which we keep the target accuracy in the solution fixed. For our largest run, we solve a problem with one billion unknowns on a tree with maximum depth equal to 10 and using 14th-order elements on 16,384 x86 cores on the "STAMPEDE" system at the Texas Advanced Computing Center.**

I. Introduction

We propose fast algorithms for solving the following scalar advection-diffusion problem for the *concentration* $c(\boldsymbol{x}, t)$:

$$\frac{\partial c(\boldsymbol{x},t)}{\partial t} + \boldsymbol{v}(\boldsymbol{x},t)\cdot\boldsymbol{\nabla} c(\boldsymbol{x},t) - \mathcal{D}\Delta c(\boldsymbol{x},t) = 0,\ \ \boldsymbol{x}\in\Omega, \quad (1)$$

with initial condition $c(\boldsymbol{x}, 0) = c_0(\boldsymbol{x})$ and with either free-space or periodic boundary conditions at the boundary $\partial\Omega$ of $\Omega = [0, 1]^3$. Here $\boldsymbol{v}$ is a given velocity field, and $\mathcal{D} > 0$ is the diffusion coefficient. This problem appears in porous media flows [20], transport phenomena in complex fluids [16], and multiphysics simulations [1], [7].

Equation (1) is a benchmark problem for many parabolic-hyperbolic partial differential equations (PDEs). Although it is a linear PDE, this equation presents several hurdles, both with respect to its discretization and its solution on high performance computing systems. First, depending on the velocity field, the diffusion coefficient and the initial condition, the solution $c(\boldsymbol{x}, t)$ can develop sharp gradients (internal layers) that are hard to resolve. Second, the spatio-temporal scales in $\boldsymbol{v}$ need not be consistent with the scales of the initial condition c_0. Third, special discretization schemes are necessary for the advection scheme. If a conditionally stable scheme is used with q_{th}-order elements and the smallest element size is $h_{\min}$, then the time step δt should be $\mathcal{O}(hq^{-2})$. For large q this can result in an excessive number of time steps. Third, an efficient solver requires the solution of elliptic problems; although the underlying theory is well understood, scaling elliptic solvers to high-order discretization on non-uniform grids is not trivial since one needs appropriate smoothers [22].

a) Contributions: We propose an *Adaptive Mesh Refinement* (AMR) scheme that uses implicit-explicit time stepping. We use an *integral equation solver* for the implicit part (parabolic diffusion with right-hand-side) and a second-order semi-Lagrangian scheme for the explicit part (linear advection) that is unconditionally stable (see §II). In particular, we do the following:

- We use an octree-based scheme with discontinuous q_{th}-order Chebyshev discretization at every octree node. Elliptic problems on this discretization are solved using a volume integral equation formulation.
- We allow for different trees for the velocity and concentration. Working with the two trees can create significant imbalances that can actually exceed memory resources due to load imbalance. We propose a new scheme that addresses this problem.
- We resolve instabilities in the semi-Lagrangian solver by remapping the points to different grids and we optimize its FLOP/s performance.
- We study the convergence along with strong and weak scaling in different scenarios and test our algorithm with time-steps that are orders of magnitude larger than the CFL stability limit. Our largest runs where done with 1 billion unknowns reaching 10 levels of refinement with 14th order elements—this is an effective resolution of nearly 100 billion unknowns with a uniform mesh.

Our integral equation solver and discretization are based on the open-source library `PVFMM` [3], [17]. The new merge

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00 ©2016 IEEE

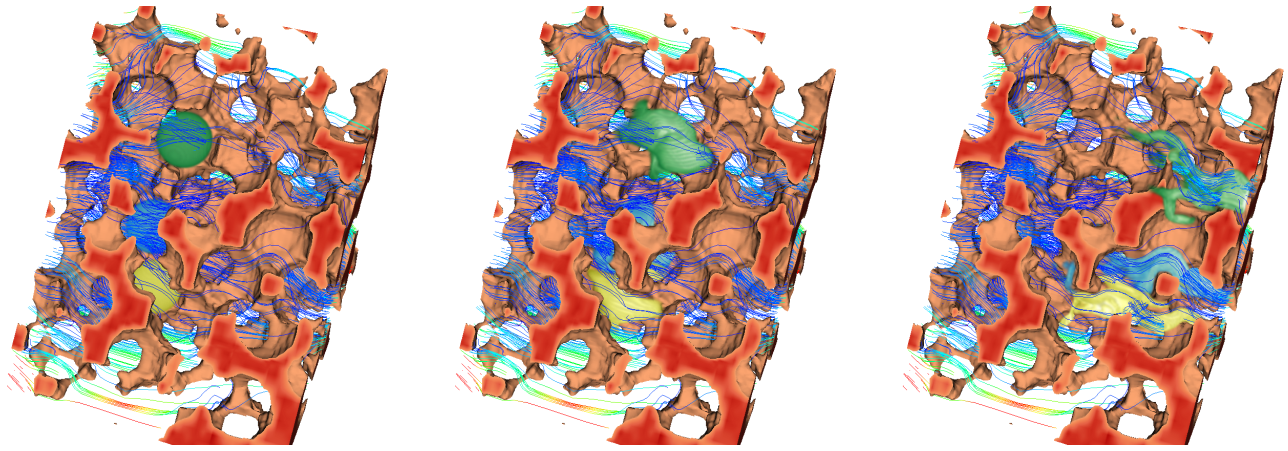


Fig. 1: *Here we illustrate the capabilities of our solver. In these three figures we show an advection diffusion problem in a porous medium. The red and orange areas represent the solid phase. The streamlines visualize the velocity field, which corresponds to a stationary Stokes flow through this porous medium microstructure. We solve* (1) *using this velocity field. The initial condition c_0 is the linear superposition of three Gaussians indicated by light blue, light green, and yellow. The velocity is calculated using a volume integral equation solver, whereas the advection diffusion problem is solved using the scheme we describe in this paper. The initial condition is the left-most image. The other images show different snapshots in time.*

algorithm for multiple octrees, the semi-Lagrangian advection on octrees and the HPC interpolation are all new technologies introduced in this paper. Our implementation uses Intel intrinsics for vectorization, OpenMP for shared memory parallelism, and the Message Passing Interface (MPI) for internode communication.

b) Related work: There is extensive literature on (1). But if we look for methods in 3D that use high-order discretization ($q > 2$), are unconditionally stable, support dynamic non-uniform grids and scale on a large number of cores, the existing work is much more limited. Excellent reviews and state-of-the-art methods for low-order discretization can be found in [1], [5], [7], [18]. Examples of state-of-the-art low-order solvers include [13] (semi-Lagrangian time adaptive, not parallel, complex geometries). In [7] the need for high-order methods is emphasized but, as the authors mention, none of the codes realize high order. Most codes target fifth-order accurate schemes at most.

A third-order scheme is discussed in [12], but it uses regular grids, is only conditionally stable (the diffusion term is treated explicitly in time) and does not support distributed memory parallelism. An 11th-order accurate code was presented in [8] with excellent scalability, but it does not support adaptive mesh refinement. Perhaps the work closest to ours is the one in [6] in which high-order discontinuous Galerkin elements are discussed. A pure advection (no-diffusion) equation was solved using a 3rd-order discretization. An advection-diffusion problem was solved using lower-order discretization. A particle method for scalar advection-diffusion was described in [14]. However, it only supports regular Cartesian grids and no mesh refinement. Finally, multiresolution methods like [2], [4] have not been scaled to distributed memory architectures. Regarding the theoretical work on semi-Lagrangian methods, the time-step we use is described in [25]. Discontinuous Galerkin schemes are discussed in [19], and monotonicity preserving schemes are presented in [24]. Theoretical analysis for conforming finite elements was introduced in [11].

c) Limitations: Our time marching is not adaptive and only second-order accurate in time. A non-adaptive higher-order scheme is possible with multistep methods but time adaptivity requires significantly more work. We only consider problems in the unit cube with either free-space or periodic boundary conditions and stationary velocities (time independent). We are currently working on extending our work to time-dependent velocity fields. Our scheme is designed for smooth velocity fields. If the velocity is not smooth, the interpolation scheme requires significant modifications. Our semi-Lagrangian scheme is pretty basic. More sophisticated schemes preserve additional properties like monotonicity and positivity. Although convergence proofs for continuous and discontinuous Galerkin approximations exist, we don't have a convergence proof for our scheme, only empirical evidence. Our scheme is *not* variable order, it uses the same q everywhere in space.

II. Background

In this section, we summarize basic facts about (1), our assumptions, and the time discretization.

The input to our problem is $\boldsymbol{v}(\boldsymbol{x}, t)$, $c_0(\boldsymbol{x})$, and $\mathcal{D}$. The output is $c(\boldsymbol{x}, t)$. We only consider strong solutions to (1). We assume that $\boldsymbol{v}(\boldsymbol{x}, t) = \boldsymbol{v}(\boldsymbol{x})$ is independent of time and that $\boldsymbol{v} \in \boldsymbol{C}^\infty(\Omega)$ and $c_0 \in C^\infty(\Omega)$. Then for $t \in (0, 1]$ (1) with periodic boundary or free-space conditions has a unique solution [10] and $c(\boldsymbol{x}, t) \in C^\infty(\Omega)$, $\forall t \in [0, 1]$. For free-space boundary conditions, we assume that c_0 has a compact support and that we are only interested in the exact solution in the center of Ω and that Ω is sufficiently large compared to the region of interest.

a) Time discretization: To discretize (1), we follow the time-stepping scheme described in [25]. Given the concentration $c(\boldsymbol{x}, t_k)$, we compute the concentration $c(\boldsymbol{x}, t_{k+1})$, where $t_{k+1} = t_k + \delta t$. A second-order time discretization is

$$\frac{\frac{3}{2}c(\boldsymbol{x}, t_{k+1}) - 2c(\boldsymbol{X}_k, t_k) + \frac{1}{2}c(\boldsymbol{X}_{k-1}, t_{k-1})}{\delta t} - D\Delta c(\boldsymbol{x}, t_{k+1}) = 0, \quad (2)$$

where $\boldsymbol{X}_k$ and $\boldsymbol{X}_{k-1}$ are points in the trajectory of a (virtual) material particle passively moving due to the velocity $\boldsymbol{v}$ and passing through $\boldsymbol{x}$ at time t_{k+1}. These positions are solved by the following equation (characteristics) backward in time

$$\frac{d\boldsymbol{X}(t)}{dt} = -\boldsymbol{v}(\boldsymbol{X}(t), t), \quad \boldsymbol{X}(0) = \boldsymbol{x}. \quad (3)$$

We define the *semi-Lagrangian points* by $\boldsymbol{X}_k = \boldsymbol{X}(\delta t)$ and $\boldsymbol{X}_{k-1} = \boldsymbol{X}(2\delta t)$. Equation (3) can be solved using an explicit time stepping scheme. In our implementation, we use a second-order Runge-Kutta scheme with the same δt as in (2). This way, we construct an optimal upwind scheme for (1) by numerically computing the backward characteristics. Solving (3) requires interpolation for $\boldsymbol{v}(\boldsymbol{X}(t), t)$. Furthermore, once we have the semi-Lagrangian points, we need to interpolate $c(\cdot, t_k)$ and $c(\cdot, t_{k+1})$ at these points. This interpolation is critical for the performance of the scheme since it can introduce artificial diffusion and overshooting. The algorithm for the interpolation is discussed in the next section. Once we have the interpolated values, we solve an elliptic problem defined by

$$\left(\frac{3}{2} - \delta t D\Delta\right) c_{k+1} = 2c_k - \frac{1}{2}c_{k-1}, \quad (4)$$

where we have suppressed the explicit notation in $\boldsymbol{x}$ and t (compare this equation to (2)). To solve for c_{k+1}, we use a volume integral equation formulation, which is described in §III-E.

The analysis of a semi-Lagrangian numerical scheme strongly depends on the spatial discretization. For a high-order conforming finite element discretization a convergence proof can be found in [11]. But, as we will see in the next section, our basis functions do not form a conforming basis, they are essentially discontinuous Galerkin functions. However, we do not do any flux correction other than the pointwise semi-Lagrangian upwinding. The analysis of our scheme is ongoing work. Let us also mention that for conforming elements, the error estimate for (2) is $\mathcal{O}(h^q/dt + dt^2)$ [11]. In the next section we discuss the spatial discretization, the merging of different octrees for the velocity and concentration, and the overall parallel complexity of the scheme.

III. Methods

A. Discretization of Concentration and Velocity Fields

We use piecewise polynomial representations for discretizing the concentration and velocity fields. This data-structure is built into the PVFMM library [3], [17]. For simplicity we just describe the scheme for the concentration. We use a similar scheme for each component of the velocity field. To discretize the concentration c, the computational domain Ω is partitioned using an octree $\mathcal{T}$. Then we construct a polynomial approximation of degree q for the function over each leaf octant $\mathcal{B} \in \mathcal{T}$. To do this, we evaluate c on a $(q+1) \times (q+1) \times (q+1)$ grid of Chebyshev node points in the leaf octant $\mathcal{B}$. We use these $(q+1)^3$ function values to construct the coefficients ($\alpha^{\mathcal{B}}_{ijk}$, $0 \le i, j, k \le q$), which define the interpolation of the data points in terms of Chebyshev polynomials. However, we don't keep all (tensor product) coefficients—for computational efficiency. We keep $(q+1)(q+2)(q+3)/6$ coefficients up to order q to construct the following approximation of c,

$$\hat{c}(x, y, z) = \sum_{i+j+k \le q} \alpha^{\mathcal{B}}_{ijk} T_i(x) T_j(y) T_k(z) \quad (5)$$

where, $T_k(x)$ is the Chebyshev polynomial of degree k in x.

We determine the depth of a leaf *adaptivily* using the norm of the tail of the coefficients. The absolute sum of the highest order coefficients ($\sum_{i+j+k=q} |\alpha^{\mathcal{B}}_{ijk}|$) gives an estimate of the truncation error. For adaptive octrees, the leaf nodes with truncation errors larger than a given tolerance ϵ_{tree} are refined recursively until the required accuracy is achieved. For distributed memory parallelism, the leaf octants in the octree data-structure are sorted by Morton order and repartitioned across MPI tasks.

B. Chebyshev Evaluation

The semi-Lagrangian time-stepping scheme requires the evaluation of the concentration and velocity fields on a large number of arbitrary points in the domain (these points are the backward trajectories of the Chebyshev points as we will see later). So, given the octree-based piecewise polynomial representation of a function, as discussed in §III-A, we need to efficiently evaluate it at a large number of arbitrary points $\{x_1, \cdots, x_n\}$. To assign evaluation points to leaf nodes, we first compute the Morton ID m_i for each evaluation point x_i and then sort the points by their Morton ID to $\{m_{k_1}, \cdots, m_{k_n}\}$. This requires $\mathcal{O}(n \log n)$ work. Then, for each leaf node $\mathcal{B}$, we determine the Morton IDs $M_{\mathcal{B}}$ for $\mathcal{B}$ and $M_{\mathcal{B}'}$ for the next leaf node in the tree $\mathcal{B}'$. In the sorted array of point Morton IDs, we determine the index range $I_{\mathcal{B}}$ such that $M_{\mathcal{B}} \le m_{k_i} < M_{\mathcal{B}'}$ for each $i \in I_{\mathcal{B}}$. This requires just two binary searches in the sorted array of point Morton IDs for each $\mathcal{B}$.

Now, we evaluate the Chebyshev approximation at each point $(x, y, z) \in \{x_{k_i} : i \in I_{\mathcal{B}}\}$,

$$c_{k_i} = \sum_{i \le q} T_i(x) \sum_{i+j \le q} T_j(y) \sum_{i+j+k \le q} T_k(z) \alpha^{\mathcal{B}}_{ijk} \quad (6)$$

Computing the above sum requires $\mathcal{O}(q^3)$ floating point operations. Even for high order approximations ($q = 14$), the coefficients $\alpha^{\mathcal{B}}_{ijk}$ easily fit in the L1 CPU cache. Therefore, for all $n_{\mathcal{B}}$ evaluation points, the coefficients $\alpha^{\mathcal{B}}_{ijk}$ must be read from the main memory only once. The $3(q+1)$ Chebyshev polynomial values $T_i(x), T_j(y), T_k(z)$ are also available in the L1 cache. Therefore, the ratio of floating-point operations to the number of memory accesses to main memory (arithmetic intensity) is high and when carefully implemented achieves high flop-rates. We have also vectorized the above computation for double precision using AVX vector intrinsics. For double-precision AVX vectorization, we vectorize to evaluate for 4 points together. In addition, we have parallelized the loop over the evaluation points using OpenMP. With these optimizations, we are able to achieve about 150GFLOP/s on a single node of Stampede achieving 43% of peak double-precision floating-point performance.

After evaluating the Chebyshev approximation at all evaluation points for each leaf node, we rearrange the values $\{c_{k_1}, \cdots, c_{k_n}\}$ to the original ordering of the points $\{c_1, \cdots, c_n\}$. For shared memory systems, we use an OpenMP merge-sort algorithm for sorting the Morton IDs. For distributed memory parallelism, we use a parallel hypercube sorting algorithm [21]. However, the global sort for assigning semi-Lagrangian points to octants can be quite costly. Since the evaluation points are related to the Chebyshev points, we expect the values of the majority of the semi-Lagrangian points to be locally available. Thus, we reduce the communication by first separating all the local points and performing the global sort only on the remote points.

C. Partitioning Schemes

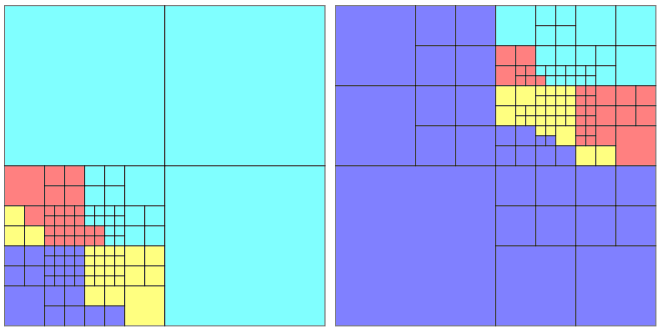
In general $\boldsymbol{v}$ and c are discretized using separate octrees, which are also partitioned independently across MPI tasks. The most expensive task in our semi-Lagrangian solver is evaluating these piecewise polynomial discretizations at points along the backward characteristics. Typically, for reasonable CFL numbers, the displacement of the Lagrangian points in one time-step is small. Therefore, to a large extent, the evaluation points are distributed according to the discretization of the concentration tree. If we allow the velocity tree to be partitioned independently of the concentration tree (Original Approach, Fig. 2a), the following two issues arise:

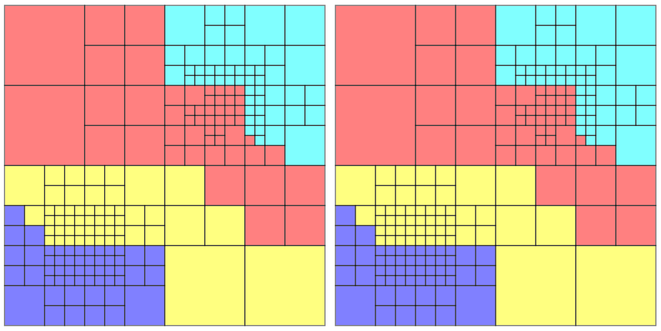
First, the evaluation points may be partitioned completely differently from the partitioning of the velocity tree, requiring very high communication load to send evaluation coordinates to remote velocity tree partitions and bringing back the evaluated velocity data to the concentration tree partition.

Second, for regions with very fine concentration mesh and a coarse velocity mesh, a very large number of evaluation points will be assigned to a single velocity partition causing imbalances in terms of computation and memory. If the velocity and concentration trees are very different, such imbalances can be quite significant to the extent that can exhaust the memory in a hardware node and cause crashes.

Both issues would severely impair the scalability and robustness of our solver. Therefore, a preprocessing step is necessary to address this problem. Fig. 2b and 2c shows two possible solutions, which we discuss below.

Complete Merge (CM). This scheme refines the velocity tree in regions where it is coarser than the concentration tree and similarly refines the concentration tree in regions where it is coarser than the velocity tree. The resulting trees, which now have the same refinement, are partitioned uniformly across the processes so that both trees have the same partitioning.


(a) ***Original Approach (OA):*** *The trees are partitioned independently.*


(b) ***Complete Merge (CM):*** *The trees are merged and then partitioned.*

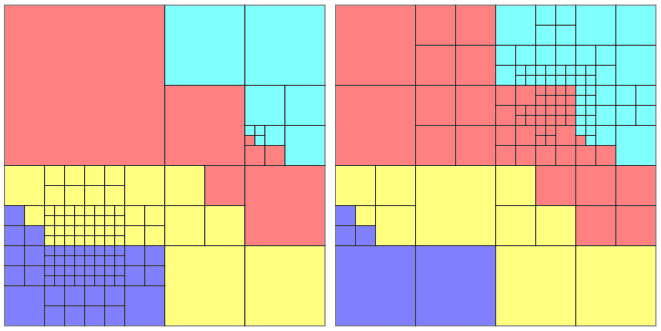

(c) ***Semi-Merge (SM):*** *The trees are partitioned based on the merged tree without actual merge. When necessary, the breaking point is created in the tree.*

Fig. 2: *Illustration of different merging schemes for concentration and velocity trees.*

Semi-Merge (SM). The semi-merge approach enforces the same partitioning of both trees with the help of some additional refinement but allows for different trees. That is, an MPI task is responsible for the same spatial region in both trees but the number of octants in the c tree and $\boldsymbol{v}$ trees in the region can differ. To determine this new partitioning, we merge the Morton IDs of the leaf nodes of both trees and then we sort them and partition them so each partition has the same number of leaves. Then, we impose this partitioning to the original

Algorithm 1 Semi-Merge partitioning scheme

```
Input:
    $\mathcal{T}_c, \mathcal{T}_{\boldsymbol{v}}$: input trees for $c$ and $\boldsymbol{v}$
Output:
    Semi-Merged $\mathcal{T}_c$ and $\mathcal{T}_{\boldsymbol{v}}$

 1: procedure SEMIMERGE($\mathcal{T}_c, \mathcal{T}_{\boldsymbol{v}}$)
 2:     $\mathcal{M} \leftarrow$ MORTONID($\mathcal{T}_{\boldsymbol{v}}$) $\cup$ MORTONID($\mathcal{T}_c$)
 3:     $\mathcal{M} \leftarrow$ PARALLELSORT($\mathcal{M}$)
 4:     $\mathcal{M} \leftarrow$ REDISTRIBUTE($\mathcal{M}$)
 5:     $b \leftarrow \mathcal{M}[0]$                    //New local breakpoint
 6:     if $b \notin \mathcal{T}_{\boldsymbol{v}}$ then
 7:         REFINETREE($\mathcal{T}_{\boldsymbol{v}}, b$)
 8:     end if
 9:     if $b \notin \mathcal{T}_c$ then
10:         REFINETREE($\mathcal{T}_c, b$)
11:     end if
12:     REPARTITIONTREE($\mathcal{T}_{\boldsymbol{v}}, b$)
13:     REPARTITIONTREE($\mathcal{T}_c, b$)
14: end procedure
```

trees. Of course this doesn't ensure optimal communication or work, but in practice it performs reasonably well. The main memory costs are associated with the leaves. Therefore, partitioning the merged leaves helps with memory load balance and makes the overall scheme robust.

For small CFL numbers we can prove that our code will not crash due to insufficient memory caused by a large load imbalance. However, for a semi-Lagrangian scheme, it is hard to balance the memory for arbitrarily large CFL numbers and complex velocity fields since the particle distribution during the interpolation phase can be different from both concentration and velocity fields. A robust scheme is possible but requires modifications beyond the scope of this paper. Also, we would like to mention that one could introduce some kind of weighted partitioning to improve performance but we do not do so in this paper. To ensure that the partition boundaries are resolved in each tree, we have to introduce a few new leaf nodes locally (without further collective communication). This approach adds only a very small number of octants to either tree. The overall scheme results in significantly fewer octants than in the complete merge approach.

In Table I, we present a detailed comparison of the total runtime breakdown for three schemes: the original approach in which we do not attempt to merge the trees (denoted by **OA**), the complete merge scheme (denoted by **CM**) and the semi-merge scheme (denoted by **SM**). A major disadvantage of the complete merge approach is that it requires a larger number of unknowns. Both velocity and concentration have to be represented on a finer mesh than what would be required if both quantities are represented independently on separate trees. Therefore, the complete merge scheme has a larger memory footprint, requires more work and more communication compared to our semi-merge scheme. These drawbacks can be clearly observed from the results presented in Table I, where the complete-merge scheme is over 3x slower than the semi-merge scheme. While the original approach requires a similar number of unknowns, it is over 2x slower than semi-merge scheme due to the high communication overhead. In this test case, both the original and the complete merge approaches failed for 32 processes due to exceeding memory consumption while the semi-merge approach performs well.

D. Semi-Lagrangian Advection

In our semi-Lagrangian scheme, we need to construct the interpolants of the form $c(\boldsymbol{X}_k, t_k)$ by evaluating the concentration solution at time t_k at the semi-Lagrangian point $\boldsymbol{X}_k$. This is used to construct the RHS in equation (4), which is solved to obtain the concentration at time t_{k+1}. To do this, we have to construct a piecewise polynomial approximation of the form discussed in §III-A. We first initialize a new tree $\mathcal{T}_{k+1}$ with the same refinement as the concentration tree $\mathcal{T}_k^c$ at time t_k. Then we select a set of interpolation points at each leaf-node

TABLE I: *Comparison of merging schemes for an advection problem for various numbers of processes with a Hopf field as the velocity field for one time step with $\delta t = 2.5\text{E-}2$ and low-order discretization ($q = 5$). We report the detailed breakdown of the total time (T_{solve}) into tree merging (T_{merge}), sorting of local points (T_{sort}), communicating of points with other processes (T_{comm}), Chebyshev evaluation (T_{eval}) and tree refinement (T_{ref}). In this test case, the semi-merge algorithm is more than $2\times$ faster than the original approach and more than $3\times$ faster than complete merge approach.*

Merging	p	q	N_{dof}	T_{ref}	T_{merge}	T_{comm}	T_{sort}	T_{eval}	T_{solve}
OA			–	–	–	–	–	–	–
CM	32	5	–	–	–	–	–	–	–
SM			4.0E+7	3.9	6.2	1.3	5.2	1.4	21.8
OA			4.0E+7	0.5	–	20.4	0.4	1.5	23.3
CM	64	5	4.3E+8	3.0	5.4	17.4	4.6	1.6	34.4
SM			4.0E+7	2.2	3.1	1.3	2.9	0.7	12.4
OA			4.0E+7	0.5	–	13.7	0.2	0.8	15.6
CM	128	5	4.3E+8	1.7	4.4	13.3	2.7	1.2	24.6
SM			4.0E+7	1.4	1.6	0.9	1.4	0.4	7.0
OA			4.0E+7	0.9	–	8.7	0.1	0.5	11.1
CM	256	5	4.3E+8	1.2	1.8	9.9	1.0	0.6	15.3
SM			4.0E+7	1.6	1.0	0.9	0.8	0.2	6.3
OA			4.0E+7	0.7	–	6.3	0.0	0.3	7.9
CM	512	5	4.3E+8	1.1	1.6	6.8	0.7	0.4	11.5
SM			4.0E+7	0.7	0.9	0.7	0.4	0.1	3.5

in $\mathcal{T}_{k+1}$. We compute the backward characteristics for these points using a second-order Runge-Kutta scheme (requires two evaluations of the velocity tree) and evaluate the concentration tree $\mathcal{T}_k^c$ at these points as discussed in §III-B. From these values at the interpolation points, we compute the coefficients in the Chebyshev approximation, by solving a least-squares problem for each leaf node in $\mathcal{T}_{k+1}$.

For a given set of n interpolation points (x_i, y_i, z_i) and $i \in \{1, \cdots, n\}$, we construct the matrix $M_{ij} = T_j(x_i, y_i, z_i)$. Here, $T_j(x, y, z)$ are the Chebyshev polynomials of the form $T_{j_1}(x)T_{j_2}(y)T_{j_3}(z)$ such that $j_1+j_2+j_3 \leq q$. We precompute the pseudoinverse M^{-1} for the matrix M. Then, from the set of n values $c_i^{\mathcal{B}}$ at the interpolation points for a tree node $\mathcal{B}$, the coefficients for its Chebyshev approximation are given by the matrix-vector product $\alpha^{\mathcal{B}} = M^{-1}c^{\mathcal{B}}$.

Typically, we want to choose the interpolation points in such a way that the matrix M is well-conditioned. If we consider a tree node defined by the box $[-1, 1]^3$ and choose the Chebyshev points (x_i, y_j, z_j) for $i, j, k \in \{1, \cdots, n\}$ with $x_i, y_i, z_i = \cos((2i-1)\pi/(2q))$ as interpolation nodes, then the columns of the matrix M are orthogonal and the matrix is well-conditioned. However, because these node points are strictly in the interior of the box, for sufficiently small time step size or velocity, it results in an unstable advection scheme since we do not get information from the octree nodes in the upwind direction. Therefore, we scale the Chebyshev interpolation node coordinates by $1/\cos(\pi/(2q))$.

Notice that we are using $(q+1)^3$ node points to compute approximately $(q+2)^3/6$ coefficients for the Chebyshev approximation. However, we observed that using fewer interpolation points results in a larger condition number for the matrix M, which leads to a numerically unstable scheme for long time-horizon simulations.

E. Volume Fast Multipole Method

The volume fast multipole method (FMM) computes the convolution of a given density function f with a kernel function G,

$$c(x) = \iiint_y G(x, y)\, f(y) dy, \tag{7}$$

where G is the fundamental solution for an elliptic PDE. The volume potential $c(x)$ computed above gives the solution of the corresponding elliptic PDE [9], [15]. We solve the modified Laplace problem discussed in §II. For the modified Laplace equation: $\alpha c - \Delta c = f$, the fundamental solution with free-space boundary conditions is given by

$$G(x, y) = \frac{1}{4\pi} \frac{1}{|x-y|} e^{-\sqrt{\alpha}|x-y|} \tag{8}$$

In this work, we use the PVFMM library [3], [17] which is a parallel, scalable and highly optimized implementation of the volume FMM in 3D. We briefly summarize the volume FMM and the main features of the PVFMM library. The volume FMM uses the piecewise polynomial discretization on an adaptive octree $\mathcal{T}$, to represent the input density function f and the resulting potential c. We discuss this discretization in greater detail in §III-A since we use the same data-structure in our advection-diffusion algorithm. The integral in equation (7) can be computed as the sum of several smaller integrals over each leaf octant $\mathcal{B} \in \mathcal{T}$.

$$c(x) = \sum_{\mathcal{B} \in \mathcal{T}} \iiint_{y \in \mathcal{B}} G(x-y)\, f(y) dy \tag{9}$$

The above integral is computed at the Chebyshev node points of every leaf octant in the tree. Then, a piecewise polynomial representation is constructed for the potential c.

The integrals over leaf octants $\mathcal{B}$ that are well-separated from the target evaluation point x can be evaluated efficiently using standard Gaussian quadrature. In addition, these interactions are low-rank and can be compressed by constructing a multipole expansion for $\mathcal{B}$ to approximate its far-field. The accuracy of the far-field representation is determined by the order of the multipole expansion m. Typically, for a given accuracy a higher compression can be achieved as the distance between the octant $\mathcal{B}$ and x increases. This allows the fast multipole method to compute these interactions hierarchically, so that interactions with octants which are further away are evaluated at coarser scales. The PVFMM implementation uses the kernel independent FMM [26]. It works well for elliptic PDEs and is therefore applicable to the modified Laplace kernel used in the present work.

For leaf octants $\mathcal{B}$ which contain x or are very close to it, computing the above integral over $\mathcal{B}$ requires special singular and near singular quadratures because of the $1/\|x-y\|$ factor in the kernel function. Computing these integrals on-the-fly would be prohibitively expensive. The volume FMM precomputes the interaction matrices for these near-interaction operators. The interactions can then be computed efficiently using matrix-vector products. To keep the number of interaction matrices small, so that the precomputation is feasible, a 2:1 balance constraint (adjacent leaf octants differ by at most one level) has to be enforced on the octree .

The PVFMM library optimizes near-interaction computations by combining several interactions together into a single matrix-matrix product computed efficiently using BLAS. The far-field interactions are optimized for data locality in cache and vectorized for AVX and SSE instructions. The distributed memory parallelism uses Morton ordering of the leaf nodes to partition data between the processes. Overall, the library achieves high flop-rates and good strong and weak scalability.

IV. Results

A. Numerical Results and Single-Node Performance

We conduct numerical experiments to demonstrate the correctness of our scheme and verify its scalability. All experiments were carried out on the Stampede system at the Texas Advanced Computing Center. Stampede nodes have two 8-core Intel Xeon E5-2680 (2.8GHz) processors and 32GB RAM. Stampede has a theoretical peak performance of 345GFLOP/s per node (excluding the Xeon Phi). We use the PVFMM [17] library for the FMM. Our code is written in C++ and uses OpenMP for shared memory parallelism, Intel

MKL for high performance linear algebra operations, and Intel MPI for distributed memory parallelism. Our semi-Lagrangian interpolation is optimized with SSE2 and AVX. All our runs where done by using ***1 MPI task / node*** and ***16 OpenMP threads***.

All the velocity fields we use for convergence analysis are analytic and scaled so that $\|\boldsymbol{v}\|_\infty \approx 1$. For an octree with maximum depth L and discretization order q, we can estimate the CFL number by

$$\text{CFL} = \frac{\delta t}{\delta x_{\text{cfl}}} \text{ and } \delta x_{\text{cfl}} = 2^{-L} q^{-2}, \tag{10}$$

since the spacing of our points is that of the Chebyshev points [23]. Using this formula, we see that the CFL in most of our runs is quite large ($\mathcal{O}(100)$). In the tables, N_{dof} indicates the number of true degrees of freedom, roughly $q^3/6$ Chebyshev coefficients at each octant. Recall that, when we perform the semi-Lagrangian advection, we use q^3 actual points, so the problem size for the semi-Lagrangian, interpolation, sorting, and communication is six times larger than N_{dof}. Regarding the semi-merging scheme, we can relate the memory costs and show the robustness of the code to the CFL number. More specifically, for a CFL number smaller than q^2, we would only need to move particles by distances smaller than the dimension of the smallest octant. So the extra memory needed would be bounded by the number of ghost octants. For a 2:1 balanced octree this is bounded.

To demonstrate the convergence of the advection solver, we consider the Taylor-Green vortex velocity

$$\boldsymbol{v}\left(x,y,z,t\right) = \begin{pmatrix} \cos\left(2\pi x\right)\sin\left(2\pi y\right)\sin\left(2\pi z\right) \\ \sin\left(2\pi x\right)\cos\left(2\pi y\right)\sin\left(2\pi z\right) \\ \sin\left(2\pi x\right)\sin\left(2\pi y\right)\cos\left(2\pi z\right) \end{pmatrix}. \tag{11}$$

For the initial concentration field, a Gaussian function is used. Since no analytical solution for this problem is available, to compute the error, we advect the concentration field for the second half of the time horizon with the reversed velocity field and compare our solution with the initial condition.

We present convergence results of this test case for uniform and adaptive trees in Table II. For the uniform case, we increase the depth of the tree and the temporal resolution and report the error for various q for a fixed time-horizon ($T = 1.0$). In the case of the adaptive tree, we control the error by reducing the tree refinement tolerance.

For the same accuracy, an adaptive mesh requires significantly fewer degrees-of-freedom and has lower communication and computation costs. For 7 digits of accuracy, using an adaptive mesh is about 10x faster compared to a uniform mesh.

We have not implemented adaptive time-stepping for the advection-diffusion solver but there is nothing in our formulation that prevents it. Adaptive time stepping would improve time-to-solution by reducing the number of time-steps for time-varying velocity fields.

In Table III, we show convergence results for an advection-diffusion problem with a known analytical solution. The initial concentration is given by a set of five Gaussian functions randomly placed at locations $r_i \in (0, 0.2)$ with variance $\sigma_i \in (1\text{E-}2, 3\text{E-}2)$ and amplitude $a_i \in (-0.5, 0.5)$. This concentration is placed in an angular velocity field defined by $\boldsymbol{v}(r) = |r|\hat{\theta}$. For diffusivity $\mathcal{D}$, the analytic solution for the concentration at a point r and time t is given by,

$$c(r,t) = \sum_i a_i \frac{\sigma_i^3(0)}{\sigma_i^3(t)} \exp\left(-\frac{|r - r_i(t)|^2}{2\sigma_i^2(t)}\right) \tag{12}$$

where, $r_i(t) = |r_i|\left(\cos(\theta_i + t)\hat{i} + \sin(\theta_i + t)\hat{j}\right)$, $\sigma_i(t) = \sqrt{\sigma_i^2 + 2\mathcal{D}t}$.

We present results for two different values of the diffusivity ($\mathcal{D} = 1\text{E-}4, 1\text{E-}5$) and varying discretization orders ($q = 4, 8, 14$). In each case, we conduct experiments to show convergence as we reduce the time step size δt while keeping the time horizon for the simulation fixed at $T = 1.6$. We report the relative L^∞ norm of the error at $t = T$. The refinement tolerance for the Chebyshev tree is chosen experimentally to minimize the number of unknowns N_{dof} while keeping the final solution error unchanged. We also report the number of levels of refinement for the adaptive octree, the total time to solution T_{solve} and the overall flop-rate in GFLOP/s. In addition, we report the breakdown of the time spent in different stages of the algorithm: tree refinement (T_{ref}), semi-Lagrangian advection ($T_{sort} + T_{eval}$), and diffusion computation using FMM (T_{fmm}). For the Chebyshev evaluation and FMM computation stages, we also report the flop-rates.

For each choice of diffusivity and discretization order, we show results for three different time step sizes. Starting with $\delta t = 0.1$ and 16 time steps, we achieve about two digits of accuracy. As we reduce the time step size by $4\times$ to $\delta t = 2.5\text{E-}2$ (64 time steps), we observe that the L^∞ error in the solution drops by $16\times$. We observe a further $16\times$ drop in error as we reduce the time step size to $\delta t = 6.25\text{E-}3$ (256 time steps). This confirms the quadratic convergence with δt.

While the discretization order does not affect the accuracy of the solution (for fixed ϵ_{tree}), it can have a significant effect on the cost of the algorithm. In general, a higher discretization order also results in a higher cost per unknown. For example, in the case of Chebyshev evaluation, the cost of each evaluation is $\mathcal{O}(q^3)$. Similarly, for the volume FMM, the cost per unknown depends on the Chebyshev degree q and the multipole order m. However, when approximating smooth functions, a higher discretization order can result in significantly fewer unknowns. In Table III, for $\delta t = 0.1$, we require about 5E+5 unknowns and 7 levels of octree refinement with low-order discretization. For the same case, with high-order discretization, we require about 3E+5 unknowns and 4 levels of octree refinement. For higher accuracy (with $\delta t = 6.25\text{E-}3$), the difference is even more significant with higher order discretization requiring $15\times$ fewer unknowns. We observe that for low accuracy, a moderate discretization order ($q = 8$) works best and for higher accuracy, $q = 14$ gives a faster time to solution.

In Table III, we have also presented a detailed breakdown of the total solve time. T_{ref} is the time spent in refinement

TABLE II: *Convergence of the advection solver for Taylor-Green vortex flow and Gaussian function as the concentration field with a uniform and adaptive trees. T_{solve} is the overall time in seconds on a single node.*

			Uniform Tree				Adaptive Tree				
q	dt	N_{iter}	L	L^2	L^∞	T_{solve}	ϵ_{tree}	L	L^2	L^∞	T_{solve}
8	1.0E-2	100	3	3.5E-2	2.5E-2	7.79	1E-3	4	3.2eE-3	2.7E-3	12.83
8	5.0E-3	200	4	1.3E-3	8.7E-4	93.78	1E-4	4	2.9eE-4	9.9E-5	57.03
8	2.5E-3	400	5	1.1E-5	1.1E-5	1449.34					
14	1.0E-2	100	3	1.7E-3	1.3E-3	38.52	1E-3	3	1.7eE-3	6.0E-4	20.49
14	5.0E-3	200	4	7.5E-6	7.9E-6	525.48	1E-5	3	2.6eE-5	2.3E-5	133.88
14	2.5E-3	400	5	1.6E-7	2.8E-7	8059.58	1E-7	4	3.2eE-7	2.0E-7	809.80

TABLE III: *Convergence and single-node performance results for an advection-diffusion problem with different values of diffusivity ($\mathcal{D}$ = 1E-4, 1E-5) and discretization orders ($q = 4, 8, 14$) for a fixed time-horizon ($T = 1.6s$). In each case, we show convergence in relative L^∞-norm at $t = 1.6$ as we reduce the time step size (δt) and the refinement tolerance (ϵ_{tree}). We report the total time to solution (T_{solve}) and the overall performance in GFLOP/s. We have also presented a breakdown of the cost of various stages in the algorithm.*

$\mathcal{D}$	q	δt	ϵ_{tree}	L	N_{dof}	L^∞_T	T_{solve} (GFLOP/s)	T_{ref}	T_{sort}	T_{eval} (GFLOP/s)	T_{fmm} (GFLOP/s)
1E-4	4	1.0E-1	2.5E-4	7	5.2E+5	6.9E-3	14.1 (44)	2.3	2.0	0.5 (38)	7.9 (76)
1E-4	4	2.5E-2	1.0E-5	8	5.0E+6	3.4E-4	569.0 (44)	43.0	112.7	19.7 (42)	312.4 (78)
1E-4	4	6.3E-3	2.5E-6	8	1.3E+7	3.3E-5	8185.7 (33)	301.9	2160.4	251.7 (35)	4562.4 (57)
1E-4	8	1.0E-1	2.5E-4	5	2.8E+5	5.2E-3	4.1 (41)	0.2	1.0	0.5 (91)	1.6 (74)
1E-4	8	2.5E-2	5.0E-5	5	4.0E+5	3.3E-4	22.6 (43)	0.8	6.4	2.8 (94)	8.4 (82)
1E-4	8	6.3E-3	2.5E-6	6	9.6E+5	2.1E-5	269.5 (38)	5.1	90.1	32.1 (96)	70.7 (97)
1E-4	14	1.0E-1	1.0E-3	4	3.0E+5	5.2E-3	6.2 (58)	0.3	1.2	1.5 (134)	1.8 (81)
1E-4	14	2.5E-2	5.0E-5	4	5.4E+5	3.3E-4	34.0 (71)	1.0	8.3	9.1 (143)	9.3 (108)
1E-4	14	6.3E-3	2.5E-6	5	8.4E+5	2.2E-5	229.1 (72)	4.6	62.3	59.2 (146)	53.5 (136)
1E-5	4	1.0E-1	1.0E-4	8	9.5E+5	9.3E-3	41.8 (43)	5.8	6.9	1.5 (39)	21.7 (80)
1E-5	4	2.5E-2	1.0E-5	9	5.0E+6	5.9E-4	864.0 (45)	54.4	178.6	28.7 (44)	475.5 (78)
1E-5	4	6.3E-3	2.5E-6	9	1.3E+7	3.7E-5	12241.0 (35)	426.5	3401.3	342.1 (40)	6878.3 (60)
1E-5	8	1.0E-1	2.5E-4	6	2.8E+5	9.3E-3	5.9 (43)	0.3	1.5	0.7 (93)	2.3 (81)
1E-5	8	2.5E-2	5.0E-5	6	4.0E+5	6.0E-4	30.6 (44)	1.0	9.2	3.8 (95)	10.9 (87)
1E-5	8	6.3E-3	2.5E-6	7	9.6E+5	3.7E-5	361.8 (37)	6.2	124.7	42.0 (96)	91.8 (98)
1E-5	14	1.0E-1	1.0E-3	4	3.0E+5	9.1E-3	7.7 (60)	0.3	1.6	1.8 (138)	2.2 (88)
1E-5	14	2.5E-2	5.0E-5	5	5.4E+5	6.0E-4	43.4 (76)	1.2	10.7	11.2 (143)	12.5 (125)
1E-5	14	6.3E-3	2.5E-6	5	8.4E+5	3.7E-5	310.4 (72)	5.4	86.6	78.5 (146)	67.6 (147)

and coarsening of the Chebyshev octree. This makes up a small percentage (approximately $2\%-5\%$) of the total runtime. For semi-Lagrangian advection, the most time consuming part of the computation is evaluating the piecewise Chebyshev representation of the velocity and the concentration. It has the following two main components. T_{sort} is the time for sorting the evaluation points by their Morton IDs to determine the octant in which each point resides and rearranging the evaluated values in the original ordering of the points. T_{eval} is the time spent in evaluating the Chebyshev representation of each leaf node at the points belonging to that octant. The Chebyshev evaluation has $\mathcal{O}(q^3)$ cost per evaluation point and therefore, for low-order discretizations it is significantly less expensive than the sorting. However, for $q = 14$ they have comparable cost. For high order discretizations, the Chebyshev evaluation has very high arithmetic intensity and we are able to achieve about 146GFLOP/s or 42% of the theoretical peak performance. This is due to careful vectorization and optimization for data reuse in cache. T_{sort} and T_{eval} together account for $20\% - 30\%$ of the solve time for low order discretization and about $44\%-53\%$ of the solve time for high order discretization. Diffusion is added by using the volume FMM to compute the convolution of the concentration with fundamental solution of the modified Laplace equation. The FMM evaluation time T_{fmm} accounts for $52\% - 56\%$ of the solve time for low order discretizations and $22\% - 29\%$ of the solve time for high order discretizations. For high discretization orders and sufficiently large problem sizes, the volume FMM can achieve high flop rates. However, for low order discretization or small problem sizes, the performance degrades quickly. In the results presented here, we observe a performance in the range of 57GFLOP/s to 147GFLOP/s.

B. Scaling Results

In this section, we present weak and strong scalability results. For some of these problems, we do not have an analytic solution so we only report timing and the breakdown for different parts of the algorithm. In particular:

- In Table IV we report strong scaling results for an advection-diffusion problem with a modest number of unknowns. The initial condition is a linear combination of Gaussians and the velocity field is a rigid rotation; so we know the solution analytically. This problem is solved to 4 digits of accuracy.
- In Figure 4 we report strong scaling for two much larger problems and we scale from 16 nodes up to 1024 nodes. The initial condition is a Gaussian sphere and the velocity field is the Taylor-Green vortex.
- In Figure 5 we report weak scaling from 32 to 1024 nodes for the same problem setup as above.

TABLE IV: *We report strong scaling results for an advection-diffusion problem with diffusivity $\mathcal{D} = 1\text{E-}5$ solved to 4-digits of accuracy. We have used the time step size $\delta t = 6.25\text{E-}3$ and 256 time steps. We show results for moderate and high order discretizations. Here p is the number of compute nodes, q is the degree of the approximation, L is the maximum tree level during the adaptive tree refinement, and N_{dof} is the total number of unknowns. We also present a detailed breakdown of the total runtime T_{solve} into T_{ref} (adaptive refinement, repartitioning and merging of trees), T_{fmm} (volume FMM computation), T_{comm}, T_{sort} and T_{eval} (communication, local sorting and Chebyshev evaluation at leaf nodes). For floating-point intensive parts of the algorithm we also report the performance in GFLOP/s* ***per compute node****. As we scale from 1 compute node to 64 compute nodes, we achieve $17.9\times$ speed up, 28% parallel efficiency for $q = 8$ and $13.4\times$ speed up, 21% parallel efficiency for $q = 14$. The code scales reasonably well up to 16 nodes but then the efficiency drops.*

				Refinement	Semi-Lagrangian				FMM		Total
p	q	L	N_{dof}	T_{ref}	$remote\%$	T_{comm}	T_{sort}	T_{eval} (GFLOP/s)	T_{fsetup}	T_{fcomp} (GFLOP/s)	T_{solve} (GFLOP/s)
1	8	8	2.0E+7	204.3	0.0	0.0	4100.5	923.1 (97.2)	628.6	1080.0 (152)	9094.3 (29)
2	8	8	2.0E+7	687.7	0.2	24.3	1608.2	467.7 (96.0)	332.2	563.6 (145)	4816.3 (27)
4	8	8	2.0E+7	486.3	0.9	40.2	769.6	297.6 (75.4)	193.8	285.0 (144)	2646.9 (25)
8	8	8	2.0E+7	415.1	1.6	49.0	398.7	248.3 (45.2)	218.8	158.4 (130)	1786.8 (19)
16	8	8	2.0E+7	271.2	2.5	61.9	208.6	158.0 (35.5)	206.2	114.9 (90)	1184.4 (14)
32	8	8	2.0E+7	181.2	4.0	63.9	100.6	77.4 (36.2)	153.9	72.9 (71)	739.0 (11)
64	8	8	2.0E+7	129.8	6.0	81.6	52.1	42.5 (33.0)	111.4	53.0 (50)	507.7 (8)
1	14	6	9.4E+6	54.1	0.0	0.0	1607.6	1129.0 (150.1)	54.4	380.3 (253)	4277.3 (65)
2	14	6	9.4E+6	157.3	0.4	20.4	741.8	582.1 (145.6)	51.2	209.2 (230)	2314.1 (61)
4	14	6	9.4E+6	128.5	1.4	32.2	354.6	291.2 (145.5)	49.7	115.2 (209)	1244.5 (56)
8	14	6	9.4E+6	120.9	2.5	40.2	171.9	151.0 (140.4)	64.4	72.2 (167)	764.8 (46)
16	14	6	9.4E+6	115.1	4.1	44.1	82.4	79.1 (133.9)	68.5	49.5 (123)	516.6 (35)
32	14	6	9.4E+6	112.2	7.1	54.5	36.7	42.4 (125.0)	55.8	40.6 (75)	374.9 (24)
64	14	6	9.4E+6	99.4	13.4	68.7	16.9	24.3 (108.8)	52.0	37.2 (42)	319.7 (15)

In all of our runs, we adaptively refine or coarsen the octree ***at every time step***. We discuss our results in detail below.

In Table IV, we present strong scaling results on 64 compute nodes of Stampede. Our test problem is similar to the one discussed in §IV-A. However, we choose 300 randomly distributed Gaussian functions with variance $\sigma_i \in (5\text{E-}3, 1.5\text{E-}2)$. We use the time step size $\delta t = 6.25\text{E-}3$ and 256 time steps. The relative L^∞ norm of error at the end of the simulation ($t = 1.6$) is $5.5\text{E-}5$. We use the semi-merge scheme to partition the concentration and velocity trees across processors. We report results for moderate ($q = 8$) and high ($q = 14$) order discretizations. Overall the high-order scheme delivers nearly $2\times$ speedup for the same accuracy.

For the $q = 8$ case, we require 8 levels of octree refinement with $122K$ leaf octants corresponding to 20 million unknowns. Scaling from 1 compute node to 8 nodes, we achieve $5\times$ speedup or nearly 63% parallel efficiency. The efficiency drops to 48% for 16 nodes. Scaling from 1 compute node to 64 compute nodes, we achieve nearly $18\times$ speedup or 28% parallel efficiency. The time for adaptive refinement, repartitioning and merging of the velocity and concentration trees is reported as T_{ref}. On a single compute node, this accounts for just 2% of the total run time. On two compute nodes, we observe a $3.4\times$ increase in T_{ref} due to the communication between compute nodes. On 64 compute nodes, T_{ref} makes up for nearly 26% of the total time. The computation for the semi-Lagrangian advection is dominated by evaluation of the piecewise polynomial representation of concentration and velocity. We report a breakdown of the run time for the evaluation phase into: T_{comm} is the time for communicating point coordinates and bringing back the evaluated values for points which have to be evaluated on a remote processor, T_{sort} is the time for locally sorting the points by Morton ID to determine the leaf octant on which the points have to be evaluated, and T_{eval} is the time for evaluating the Chebyshev representation of each leaf node at the points belonging to that octant. Among these, the local sorting is the most expensive stage. As we scale from 1 node to 64 compute nodes, we observe a $79\times$ speedup for the local sorting. The Chebyshev evaluation phase is the second most expensive stage in our semi-Lagrangian scheme. For this stage, we achieve about 34% parallel efficiency scaling up to 64 nodes. We also report the percentage ($remote\%$) of points which have to be communicated to a different processor for evaluation. This communication cost is reported in T_{comm}. As we increase the number of compute nodes, the combined available network bandwidth increases. However, we also observe that the percentage of points which need to be communicated increases significantly. Consequently, the communication time T_{comm} increases as we increase the number of compute nodes. For 64 compute nodes, T_{comm} accounts for 16% for the total run time. We report the time for diffusion computation (using the volume FMM) in two parts: the FMM setup time (T_{fsetup}) and the actual FMM computation time T_{fcomp}. The FMM setup stage involves constructing the local essential tree, creating interaction lists and allocating memory buffers for the FMM computation. We need to setup FMM whenever the tree refinement changes. In our case, we have to do this in each time step. Some operations in the setup phase are communication intensive. Therefore, the FMM setup phase scales relatively poorly, giving a $5.6\times$ speed up on 64 compute nodes. On the other hand, the computation phase of FMM is highly optimized and achieves good performance. Overall, the FMM phase ($T_{fsetup} + T_{fcomp}$) of the algorithm accounts for $19\% - 32\%$ of the total time.

In the same table, we have reported similar strong scaling results for $q = 14$. Compared to $q = 8$, we require less

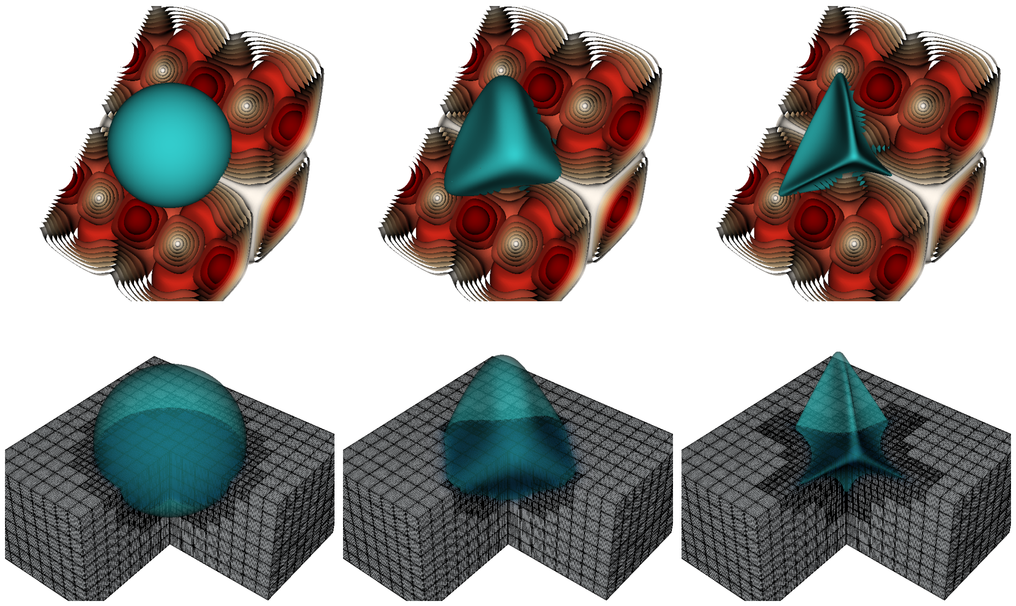


Fig. 3: *In this figure we show the contours of the Taylor-Green vortex flow, which we use for the velocity field, and the Gaussian sphere that we use for the initial condition of the concentration. We also depict a few snapshots from the evolution of c. We can see that it develops sharp gradients that require significant refinement to resolve them accurately. We also observe that the spatial features of c are quite different from the spatial features of $\boldsymbol{v}$. In the second row, we visualize the octree mesh for the different time steps to highlight the dynamic mesh adaptation.*

than half the number of unknowns (9.4 million) and just $14K$ leaf octants. Consequently, the local sort time T_{sort} is smaller. The cost for the Chebyshev evaluation per unknown is higher. However, we also achieve higher flop-rates due to higher arithmetic intensity. The Chebyshev evaluation time T_{eval} shows $46.5\times$ speed up (73% parallel efficiency) when scaling to 64 compute nodes. The FMM computation stage also shows higher flop-rates and scales well up to 16 compute nodes. Because we have such a small number of octants and a relatively large CFL number, the tree refinement (T_{ref}), the ghost point communication (T_{comm}) and the FMM setup (T_{fsetup}) show poor scalability due to increasing communication costs.

In Figure 4, we report strong scaling results for the advection-diffusion solver on up to 1024 compute nodes of Stampede. In this test case, the initial concentration is given by $c = \exp\left(-\left(r/R\right)^{\alpha}\right)$, where R is the radius of a sphere and r is the distance from the evaluation point to the center of this sphere. By increasing α, c develops a sharp gradient around $r = R$. This way, we adjust our problem size only by changing the value of α. Roughly speaking, by doubling α we increase the number of octants by $4\times$. We present results for $\mathcal{D} = 1\text{E-}3$ and $R = 0.1$. The concentration field is placed in a Taylor-green vortex flow, illustrated in Figure 3. We present strong scaling results for two problems with 3.6E+8 and 7.4E+8 unknowns each. For the largest run we have less than 900 octants per compute node. Given the large CFL number, it is not surprising that the volume of communication is that high. For the small number of octants per node and the overall problem size, the scalability is quite good.

In Figure 5, we compare the weak scaling results for the same test case as above for the complete merge and the semi-merge partitioning scheme. As we increase the number of compute nodes from 32 to 1024, the problem size increases from 47 million unknowns to 1.4 billion unknowns, an increase of almost $31\times$. Nevertheless, the overall timings increase only by a factor of two.

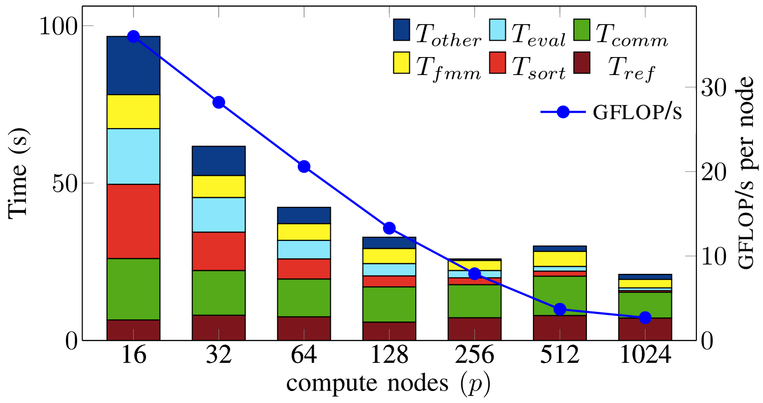
For the largest run, the FMM is less than 10% of the total time and the Chebyshev evaluation and local sorting in semi-Lagrangian takes a similar time. However, the communication cost for the semi-Lagrangian advection is excessive. This is because we have such a large CFL number.

As it can be clearly observed from the results presented in Figure 5, the semi-merge partitioning was quite efficient and drastically reduced the communication and refinement costs, while the original approach failed for most of the runs due to excessive memory consumption caused by severe load imbalance.

V. Conclusion

We presented a new algorithm for the advection-diffusion equation. For cases where the velocity field and the concentration field have different spatial scales and require separate discretization, we proposed a scheme that merges efficiently these two discretizations. We demonstrated convergence and scalability for hard cases with high levels of refinement, and with remeshing and repartitioning at every time step. We constructed simple examples in which a 4th-order discretization is at least two orders of magnitude slower than our 14th-order scheme. We found that it is critical for stability to have points right on the boundary between different octants and to filter

Fig. 4: *Strong scaling results for the advection-diffusion problem with diffusivity $\mathcal{D} = 1\text{E-}3$ with a Taylor-Green vortex flow as the velocity field. We show results for one time step with $dt = 6.25\text{E-}3$ and high order discretization ($q = 14$). We present a detailed breakdown of the total runtime in to T_{ref} (adaptive refinement and repartitioning of trees), T_{fmm} (volume FMM computation), T_{comm}, T_{sort} and T_{eval} (communication, local sorting and Chebyshev evaluation at leaf nodes). We also plot the overall performance of the code* GFLOP/s *per compute node. The left figure shows a problem with 9-levels of refinement and $3.6\text{E+}8$ unknowns. The figure on the right shows results for 10-levels of refinement and $7.4\text{E+}8$ unknowns.*



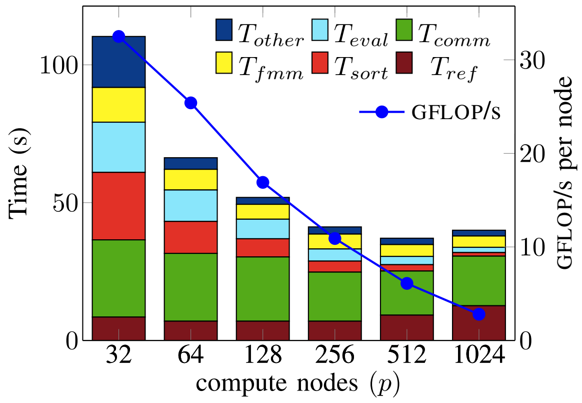
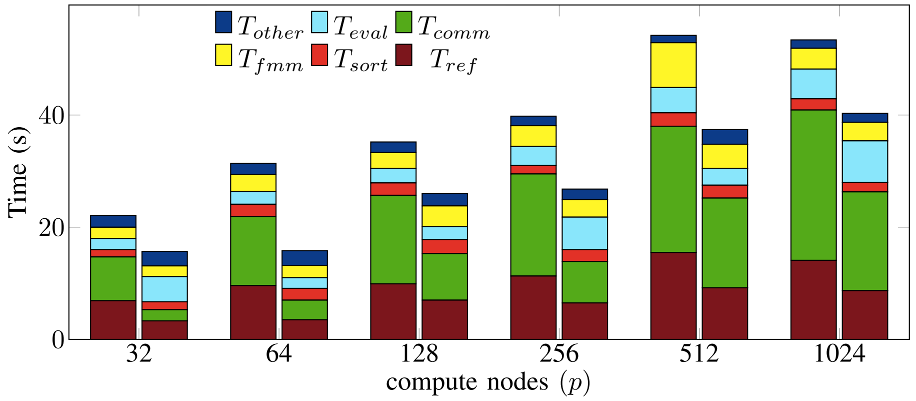


Fig. 5: *Here we present weak scaling results for the same advection-diffusion problem as in Figure 4 for the complete merge and the semi-merge partitioning scheme, respectively. We present a detailed breakdown of the total runtime into T_{ref} (adaptive refinement and repartitioning of trees), T_{fmm} (volume FMM computation), T_{comm}, T_{sort} and T_{eval} (communication, local sorting and Chebyshev evaluation at leaf nodes). We vary the problem size from $4.7\text{E+}7$ unknowns on 32 compute nodes of Stampede to $1.4\text{E+}9$ unknowns on 1024 compute nodes. The problem size per node remains roughly constant to about 2,000 octants per node. Recall that we use 1 MPI task per node and 16 OpenMP threads in all of our runs.*



the Chebyshev coefficients. Once these two modifications are in place the scheme is stable and accurate. The scheme can be extended to non-stationary velocities, nonlinear advection, and problems with variable coefficients. These are ongoing work.

The scheme can be extended in various ways. The first one is to allow for time-varying velocity fields, which are common in complex fluids. The formulation remains exactly the same, but the book-keeping becomes more involved. A second extension is to allow for variable (but smooth) diffusion coefficient. The main difference will be in the elliptic solve, where we would need to solve a volume integral equation instead of simply convolving with the Green's function. A third extension is to apply this scheme to the Navier-Stokes equations. The parabolic solve now becomes a Stokes solve, for which the fundamental solution is known and a similar formulation can be used. However, it is unclear what the stability of the nonlinear convection term will be.

Acknowledgments

This material is based upon work supported by AFOSR grants FA9550-12-10484; by NSF grant CCF-1337393; by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0010518 and DE-SC0009286; by NIH grant 10042242; by DARPA grant W911NF-115-2-0121; and by the Technische Universität München—Institute for Advanced Study, funded by the German Excellence Initiative (and the European Union Seventh Framework Programme under grant agreement 291763). Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the AFOSR, DOE, NIH, DARPA, and NSF. We would also like thank TACC's staff for their support.

References

[1] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: a massively parallel amr code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013.

[2] M. Bergdorf and P. Koumoutsakos, "A lagrangian particle-wavelet method," *Multiscale Modeling & Simulation*, vol. 5, no. 3, pp. 980–995, 2006.

[3] G. Biros and D. Malhotra, "PVFMM: A parallel kernel independent FMM for particle and volume potentials," *Communications in Computational Physics*, vol. 18, no. 3, pp. 808–830, 2015.

[4] O. Bokanowski, J. Garcke, M. Griebel, and I. Klompmaker, "An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations," *Journal of Scientific Computing*, vol. 55, no. 3, pp. 575–605, 2013.

[5] G. L. Bryan, M. L. Norman *et al.*, "Enzo: An adaptive mesh refinement code for astrophysics," *The Astrophysical Journal Supplement Series*, vol. 211, no. 2, p. 19, 2014.

[6] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, and L. Wilcox, "Extreme-scale amr," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.

[7] A. Dubey, A. Almgren *et al.*, "A survey of high level frameworks in block-structured adaptive mesh refinement packages," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3217–3227, 2014.

[8] G. K. El Khoury, P. Schlatter, A. Noorani, P. F. Fischer, G. Brethouwer, and A. V. Johansson, "Direct numerical simulation of turbulent pipe flow at moderately high Reynolds numbers," *Flow, turbulence and combustion*, vol. 91, no. 3, pp. 475–495, 2013.

[9] F. Ethridge and L. Greengard, "A new fast-multipole accelerated poisson solver in two dimensions," *SIAM J. Sci. Comput*, pp. 741–760, 2001.

[10] L. C. Evans, "Partial differential equations," 2010.

[11] M. Falcone and R. Ferretti, "Convergence analysis for a class of high-order semi-lagrangian advection schemes," *SIAM Journal on Numerical Analysis*, vol. 35, no. 3, pp. 909–940, 1998.

[12] E. Germaine, L. Mydlarski, and L. Cortelezzi, "3DFLUX: A high-order fully three-dimensional flux integral solver for the scalar transport equation," *Journal of Computational Physics*, vol. 240, pp. 121–144, 2013.

[13] A. Guittet, M. Theillard, and F. Gibou, "A stable projection method for the incompressible navierstokes equations on arbitrary geometries and adaptive quad/octrees," *Journal of Computational Physics*, vol. 292, pp. 215 – 238, 2015.

[14] J.-B. Lagaert, G. Balarac, and G.-H. Cottet, "Hybrid spectral-particle method for the turbulent transport of a passive scalar," *Journal of Computational Physics*, vol. 260, pp. 127–142, 2014.

[15] H. Langston, L. Greengard, and D. Zorin, "A free-space adaptive fmm-based pde solver in three dimensions," *Communications in Applied Mathematics and Computational Science*, vol. 6, no. 1, pp. 79–122, 2011.

[16] R. Larson, "The structure and rheology of complex fluids," 1999.

[17] D. Malhotra and G. Biros, "PVFMM home page," 2016, http://www.pvfmm.org.

[18] Q. Meng and M. Berzins, "Scalable large-scale fluid–structure interaction solvers in the uintah framework via hybrid task-based parallelism algorithms," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1388–1407, 2014.

[19] M. Restelli, L. Bonaventura, and R. Sacco, "A semi-Lagrangian discontinuous Galerkin method for scalar advection by incompressible flows," *Journal of Computational Physics*, vol. 216, no. 1, pp. 195–215, 2006.

[20] P. K. Smolarkiewicz and C. L. Winter, "Pores resolving simulation of darcy flows," *Journal of Computational Physics*, vol. 229, no. 9, pp. 3121 – 3133, 2010.

[21] H. Sundar, D. Malhotra, and G. Biros, "Hyksort: a new variant of hypercube quicksort on distributed memory architectures," in *Proceedings of the 27th international ACM conference on international conference on supercomputing*. ACM, 2013, pp. 293–302.

[22] H. Sundar, G. Stadler, and G. Biros, "Comparison of multigrid algorithms for high-order continuous finite element discretizations," *Numerical Linear Algebra with Applications*, vol. 22, no. 4, pp. 664–680, 2015.

[23] L. Trefethen, *Spectral methods in MATLAB*. Society for Industrial Mathematics, 2000.

[24] S. Verma, Y. Xuan, and G. Blanquart, "An improved bounded semi-lagrangian scheme for the turbulent transport of passive scalars," *Journal of Computational Physics*, vol. 272, pp. 1 – 22, 2014.

[25] D. Xiu and G. E. Karniadakis, "A semi-Lagrangian high-order method for Navier-Stokes equations," *Journal of Computational Physics*, vol. 172, no. 2, pp. 658–684, 2001.

[26] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *Journal of Computational Physics*, vol. 196, no. 2, pp. 591–626, 2004.