Solving PDEs in Space-Time: 4D Tree-Based Adaptivity, Mesh-Free and Matrix-Free Approaches

Masado Ishii
University of Utah
Salt Lake City, Utah

Milinda Fernando
University of Utah
Salt Lake City, Utah

Kumar Saurabh
Iowa State University
Ames, Iowa

Biswajit Khara
Iowa State University
Ames, Iowa

Baskar Ganapathysubramanian
Iowa State University
Ames, Iowa

Hari Sundar
University of Utah
Salt Lake City, Utah

ABSTRACT

Numerically solving partial differential equations (PDEs) remains a compelling application of supercomputing resources. The next generation of computing resources – exhibiting increased parallelism and deep memory hierarchies– provide an opportunity to rethink how to solve PDEs, especially time dependent PDEs. Here, we consider time as an additional dimension and simultaneously solve for the unknown in large blocks of time (i.e. in 4D space-time), instead of the standard approach of sequential time-stepping. We discretize the 4D space-time domain using a mesh-free *kD* tree construction that enables good parallel performance as well as on-the-fly construction of adaptive 4D meshes. To best use the 4D space-time mesh adaptivity, we invoke concepts from PDE analysis to establish rigorous *a posteriori* error estimates for a general class of PDEs. We solve canonical linear as well as non-linear PDEs (heat diffusion, advection-diffusion, and Allen-Cahn) in space-time, and illustrate the following advantages: (a) sustained scaling behavior across a larger processor count compared to sequential time-stepping approaches, (b) the ability to capture "localized" behavior in space and time using the adaptive space-time mesh, and (c) removal of any time-stepping constraints like the Courant-Friedrichs-Lewy (CFL) condition, as well as the ability to utilize *spatially varying time-steps*. We believe that the algorithmic and mathematical developments along with efficient deployment on modern architectures shown in this work constitute an important step towards improving the scalability of PDE solvers on the next generation of supercomputers.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; **Distributed algorithms**; • **Applied computing** → Engineering.

KEYWORDS

4D, space-time adaptive, sedectree, mesh-free, matrix-free, finite element method, distributed memory parallel

ACM Reference Format:
Masado Ishii, Milinda Fernando, Kumar Saurabh, Biswajit Khara, Baskar Ganapathysubramanian, and Hari Sundar. 2019. Solving PDEs in Space-Time: 4D Tree-Based Adaptivity, Mesh-Free and Matrix-Free Approaches. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3295500.3356198

1 INTRODUCTION

We present *kD* tree-based parallel algorithms for solving a general class of partial differential equations (PDEs) using a *space-time*-adaptive approach. This approach is primarily motivated by the necessity of designing computational methodologies that leverage the availability of very large computing clusters (exascale and beyond). For evolution problems, the standard approach of decomposing the spatial domain is a powerful paradigm of parallelization. However, for a fixed spatial discretization, the efficiency of purely spatial domain decomposition degrades substantially beyond a threshold – usually tens of thousands of processors – which makes this approach unsuitable on next generation machines. To overcome this barrier, a natural approach is to consider the time domain as an additional dimension and simultaneously solve for blocks of time, instead of the standard approach of sequential time-stepping.

This approach of solving for large blocks of space-time is one of several promising approaches to time-parallel integration approaches that have been developed over the past century, but which are gaining increasing attention due to the availability of appropriate computing resources [17]. Broadly one can consider three types of parallelization approaches to solving a space-time problem. The first type of methods explicitly parallelizes only over time, and leaves spatial parallelism (and spatial adaptivity) undefined [10, 25]. These methods may also be considered as shooting methods [17]. The second type of methods explicitly parallelizes over space, and leaves temporal parallelism (and temporal adaptivity) undefined. These methods include wave form relaxation methods that attempt to reconcile the solution at spatial boundaries between space-time blocks [17]. The third type of methods explicitly targets parallelism (and adaptivity) in space and time. The current work seeks to advance type three methods. In the more narrow context of finite

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356198

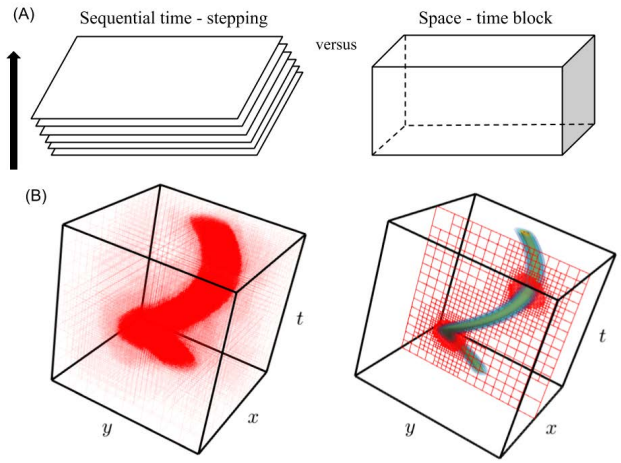

Figure 1: (A) Conventional approaches to computationally solving evolution equations rely on marching sequentially in time. This limits parallel scalability to the spatial domain. In contrast, we present an approach for simultaneously solving for large blocks of space-time. This approach not only exhibits good scalability, but also provides improved temporal convergence (proved via *a priori* estimates) as well as localized refinement in space and time based on *a posteriori* error estimates. (B) Illustrative results in $2D$ `space`$+1D$ `time` dimensions (i.e. 3D space-time) for a rotating heat source problem. Notice that the mesh is refined in space-time only in regions of interest. This not only guarantees accurate solutions in space and time, but is also substantially more efficient than the strategy of taking very small time steps necessary to guarantee temporal accuracy in the sequential approach.

element methods, early work on type three methods was considered by Hughes and coworkers [20, 22], Tezduyar et al. [35], and Potanza and Reddy [28], while variations on this theme have recently been explored by several groups [5, 6, 26, 27, 29, 37].

Our preliminary work [9] using the Finite Element Method (FEM) indicates that the third approach is particularly promising (see Fig. 1) as it provides a natural way to integrate mathematical analysis including construction of finite element error estimates (both *a priori* as well as *a posteriori*), and numerical stabilization of FEM solutions to advection-dominated equations, with developments in scalable, parallel algorithms. The finite element method (FEM) is a widely popular numerical approach to solving partial differential equations, with multiple billions/year spent in CAD and CAM (computer aided design and manufacturing) based FEM software alone [8]. Its popularity arises from a compelling set of properties including (a) the ability to model arbitrary geometries, (b) the ability to seamlessly change order of representation (linear, quadratic and higher order), (c) the ability to utilize variational arguments that guarantee monotonic convergence to the solution with improved discretization, and (d) the ability to seamlessly utilize *a posteriori* error estimates to adapt the mesh.

Solving for blocks of space-time provides a natural approach for effective usage of exascale computing resources [19], as well as leveraging novel architectures that exhibit extreme parallelism and deep memory hierarchies. In addition to this obvious advantage, solving in space-time provides the following advantages:

- Allows natural incorporation of *a posteriori* error estimates for space-time mesh adaptivity. This has several additional tangible benefits in the context of computational overhead. For evolution problems – including wave equations, and problems involving moving interfaces like bubbles and shocks – that exhibit "localized" behavior in space and time, solving in blocks of space-time that are locally refined to match the local behavior provides substantial computational gain [7]. An illustration of this advantage (in $2D$ space $+1D$ time for illustration purposes) is shown in Fig. 1(B), where the refined mesh is localized along the regions of high gradient of the solution.
- Provides easy access to the full time history of the solution which is essential for the solution of design/inverse problems involving adjoints [15, 16]
- Removes any time-stepping constraints (like the CFL condition) on the numerical approach as the solution across all time-steps are simultaneously solved. This concept is illustrated in Fig. 1(B). The temporal discretization far away from the oscillating source is 2^5 times larger than the smallest temporal discretization. This large variation in time step has no impact on numerical stability, which is an advantage over conventional sequential time stepping approaches.
- Finally, considering time as an additional dimension allows us to harness the benefits of using higher order basis functions to get high order accurate temporal schemes for no additional implementation cost (since these basis functions are already available for the spatial discretization). We derive theoretical *a priori* estimates of convergence with basis order and mesh size, and show that temporal error scales as a power of the mesh size in space time $|u - u_e|_2 \leq Ch^{\alpha}$, where α = order of basis function + 1. Fig. 2 computationally illustrates this point. We solve a 3D transient problem in space-time on a 4D mesh. Using cubic basis functions is equivalent to a multi-step fourth order Runge-Kutta scheme for the sequential time stepping problem.

Additionally, instead of using a traditional mesh-based abstraction for performing finite element (FEM) computations, we design a new mesh-free abstraction on adaptive space-time meshes with the objective of achieving high performance and scalability on current and future architectures with extreme levels of parallelism and deep memory hierarchies. The mesh-free abstraction is combined with matrix-free abstractions since these are important for large-scale parallelism.

Our contributions in this paper are as follows: (a) we develop efficient $4D$ adaptive mesh construction and 2:1 balancing algorithms, (b) we develop mesh-free matrix-free algorithms for finite element computations using a space-time formulation, (c) we derive both *a priori* error estimates, as well as residual-based *a posteriori* error estimates for a canonical problem – the linear time dependent advection diffusion equation, and numerically illustrate the theoretically determined (improved) convergence behavior of the space-time solution approach, and (d) we compare the scaling behavior of sequential time stepping method with the space-time approach.

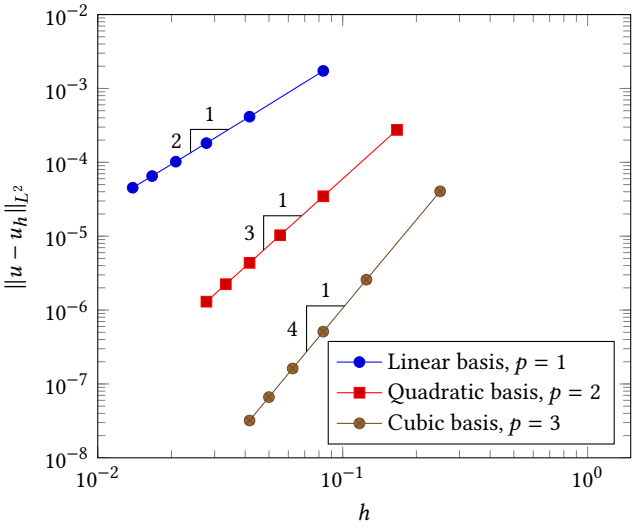

Figure 2: Considering time to be another dimension allows easy use of higher order basis functions to get high order temporal accuracy. The figure illustrates how switching from linear to quadratic to cubic basis functions (for the problem solved in Fig. 1B) in space-time is equivalent to multi-step Runge Kutta methods of order 2, 3 and 4, respectively.

The rest of the paper is organized as follows: the finite element formulation of PDEs in space-time and the related error estimates are presented in Sec. 2; the methodology for implementing the *kD*-tree based *comparison-free*, *matrix-free*, *mesh-free* algorithms are outlined in Sec. 3; numerical results are presented and discussed in Sec. 4 and concluding remarks are made in Sec. 5. Some of the important notations pertaining to the algorithms in Sec. 3 are summarized in Table 1.

A	input array (distributed)
comm	MPI communicator
p	number of MPI tasks in *comm*
N	global number of keys in A
$n = N/p$	local number of keys in A
$P(e)$	parent of tree node e
$N(e)$	neighbours of tree node e
s	local splitter indices
p_r	the task id (its MPI Rank)
$\|B\|$	number elements in array B

Table 1: *Notations used in paper.*

2 BACKGROUND: SPACE-TIME SETTING AND ERROR ESTIMATES

We present here a brief mathematical formulation of a time dependent PDE in the space-time setting. We define our spatial domain as $U \subset \mathbb{R}^n$, where U is an open and bounded domain, with $n = 1, 2,$ or 3. We consider the time horizon $I_T = (0, T) \subset \mathbb{R}^+$, which is a bounded time interval. Then we define the space-time domain as the Cartesian product of these two domains, $\Omega = U \times I_T = U \times (0, T)$. Denote the overall boundary of this space-time domain as $\Gamma = \partial\Omega$. In particular, the spatial domain boundary is defined as $\Gamma_U = \partial U \times (0, T)$ and the time boundaries are defined as $\Gamma_0 = \bar{U} \times \{0\}$ and $\Gamma_T = \bar{U} \times \{T\}$ which are the initial and final time boundaries respectively. Note that $\Gamma = \Gamma_U \cup \Gamma_0 \cup \Gamma_T$. Finally we also define the closure of Ω as $\bar{\Omega} = \Omega \cup \Gamma$.

The canonical equation we are interested is the following equation for the scalar space-time field $u : \Omega \to \mathbb{R}$:

$$\partial_t u + \mathbf{a} \cdot \nabla u - \nabla \cdot (\kappa(u) \nabla u) = f(u) \quad \text{in} \quad \Omega \tag{1a}$$

$$u = 0 \quad \text{on} \quad \Gamma_U \tag{1b}$$

$$u(\cdot, t_0) = u_0 \quad \text{on} \quad \Gamma_0 \tag{1c}$$

where, $\nabla \equiv (\partial_x, \partial_y, \partial_z)$ denotes the usual spatial gradient operator and $\mathbf{a} : \Omega \to \mathbb{R}^n$ denotes the advection velocity. In this work, $\mathbf{a}$ is assumed to be linear and divergence free. The forcing $f : \Omega \to \mathbb{R}$ and the diffusivity $\kappa : \Omega \to \mathbb{R}$, ($\kappa > 0$ in Ω) are smooth functions and, in general, nonlinear. We assume that Dirichlet conditions are imposed on the boundary Γ_U.

Next we define suitable function spaces for the trial and test functions used in the variational formulation. We define the space of the trial functions as $V_u = \left\{ u \in H^1(\Omega) : u|_{\Gamma_U} = 0, u|_{\Gamma_0} = u_0 \right\}$ and the space of the test functions as $V_v = \left\{ v \in H^1(\Omega) : v|_{\Gamma_U \cup \Gamma_0} = 0 \right\}$. Let us first assume that both κ and f in equation 1 are linear functions. Then the continuous variational problem is posed as follows: find $u \in V_u$ such that,

$$a(u, v) = l(v) \quad \forall v \in V_v \tag{2}$$

where,

$$a(u, v) = (\partial_t u, v) + (\mathbf{a} \cdot \nabla u, v) + (\kappa \nabla u, \nabla v) \tag{3}$$

$$l(v) = (f, v) \tag{4}$$

Here $(\cdot, \cdot)$ denotes the usual inner product in $L^2(\Omega)$. Note that the inner product is over the space-time domain, $(a, b) = \int_0^T \int_U ab dx dt$. The finite dimensional approximation of these spaces are denoted as V_u^h and V_v^h, and

$$V_u^h := \{u^h \mid u^h \in H^1(\Omega), \quad u^h \in P(\Omega^K) \quad \forall K, \quad u^h|_{\Gamma_U} = 0 \quad \text{and} \quad u^h|_{\Gamma_0} = u_0\} \tag{5}$$

$$V_v^h := \{v^h \mid v^h \in H^1(\Omega), \quad v^h \in P(\Omega^K) \quad \forall K, \quad v^h|_{\Gamma_0 \cup \Gamma_U} = 0\} \tag{6}$$

where $P(\Omega^K)$ being the space of the standard polynomial finite element shape functions on element Ω^K.

Numerical solutions to Equation 1 can exhibit numerical instabilities because the bilinear form $a(u, v)$ in Equation 2 is not strongly coercive in V_u. Therefore, following [24], we choose the basis function in the form of $v^h + \delta \partial_t v^h$, where δ is a mesh dependent parameter proportional to the mesh size h. This is one of the many ways to "stabilize" the original discrete problem [14, 21]. The discrete variational problem along with this stabilization is then given by,

$$a^h(u^h, v^h) = l^h(v^h) \quad \forall v^h \in V_v^h \tag{7}$$

$$a^h(u^h, v^h) = (\partial_t u^h, v^h) + (\mathbf{a} \cdot \nabla u^h, v^h) + (\kappa \nabla u^h, \nabla v^h) + \delta(\partial_t u^h, \partial_t v^h) + \delta(\mathbf{a} \cdot \nabla u^h, \partial_t v^h) + \delta(\kappa \nabla u^h, \nabla(\partial_t v^h)) \tag{8}$$

$$l^h(v^h) = (f, v^h) + \delta(f, \partial_t v^h) \tag{9}$$

This discrete bilinear form is bounded and also coercive on V_v^h with respect to the norm

$$\|u^h\|_{V_u} = \left[\kappa\|\nabla u^h\|^2_{L^2(\Omega)} + \delta\|\partial_t u^h\|^2_{L^2(\Omega)} + \frac{1}{2}\|u^h\|^2_{L^2(\Gamma_T)}\right]^{\frac{1}{2}} \tag{10}$$

Since the linear form given by Equation 9 is also bounded, the generalized Lax-Milgram Lemma guarantees a unique solution.

Denoting the solution to Equation 2 by u and the solution to Equation 7 by u^h, we derive the *a priori* error estimate as

$$\|u - u^h\|_{V_u^h} \leq Ch^p\|u\|_{H^{p+1}(\Omega)} \tag{11}$$

where p is the highest degree of the polynomial space to which u^h belongs, h is the size of the space-time element, and some constant C. Note that this estimate guarantees high order temporal accuracy (essentially, equivalent to a $p+1$-th order time stepper) when using a p^{th} order basis function. We numerically illustrate this powerful result in Fig. 2, where we solve a problem with a rotating heat source with an analytically known solution. We plot the L_2 error (over the space-time domain) between the analytical solution and the computed solution using different order of basis functions. The mesh convergence plots clearly illustrate the expected change in slope from linear to quadratic to cubic basis functions. This is equivalent to a sequential time stepper, specifically a multi-step Runge Kutta methods of order 2, 3 and 4, respectively.

Finally, based on the calculated solution u^h, we can formulate an *a posteriori* error estimate. We use the residual based approach [2, 36] that associates an error estimate with each space-time element in the 4D tree. This error estimate will inform adaptive space-time refinement of the domain. The residual based *a posteriori* error estimator for a space-time element K is given by:

$$\eta_K = \left\{ h_K^2 \left\| \bar{f}_K + \nabla \cdot \kappa \nabla u_h - \partial_t u_h \right\|_K^2 + \frac{1}{2} \sum_{E \in \mathcal{E}_K} h_E \left\| \mathbb{J}_E(\mathbf{n}_E \cdot \nabla u_h) \right\|_E^2 \right\}^{\frac{1}{2}} \tag{12}$$

where $\mathcal{E}$ denotes the set of all edges of element K, $\mathbb{J}$ denotes the jump in the gradient of the solution u^h across those edges and $\bar{f}_K$ denotes the average force value in the element K. This *a posteriori* error is calculated elementwise and a suitable refinement strategy is adopted (see next section) for space-time mesh refinement.

We emphasize that while the conceptual formulation of a PDE solution strategy in space-time blocks is not entirely new, the rigorous formulation of both *a priori* and *a posteriori* error estimates is a novel contribution. In particular, the *a priori* error estimates provide theoretical guarantees of enhanced convergence which serve as rigorous tests of our numerical implementation.

The foregoing presentation of the mathematical formulation is based on a linear equation. We defer detailed derivations for the non-linear case due to space limitations, but sketch out the basic approach (which is analogous to the one detailed above). In case of non-linearity (i.e., when either f or κ or both are nonlinear functions depending on u), we first linearize the equations (*a la* Newton-Raphson) to get a linear incremental equation. The rest of the procedure then remains similar: the variational problem is framed on the linear incremental equation, with the discrete variational problem following in an analogous way. Note that in a sequential time-marching method for a nonlinear problem, the initial condition acts as a starting value of the Newton-Raphson iteration scheme. But in the space-time setting, the initial condition only pertains to the $t = 0$ boundary and thus an initial guess over the entire space-time domain has to be provided. We use an extrusion of the initial condition over the time domain as our initial guess. This can admittedly be improved.

In this work, the finite element mesh is generated through a kD-tree based locally regular octant (for 3D) or sedecant (for 4D) objects (the "leaves" in the kD-tree as explained in Sec. 3). This mesh is not explicitly saved (see Sec. 3), and the mesh information is deduced on-the-fly. Such a mesh-free approach is particularly attractive for adaptive mesh refinement, where the mesh changes frequently and one does not want to incur the high cost of building maps (meshing). This is also a very efficient way of implementing the continuous Galerkin method through a locally structured mesh. Each of the leaves in the kD-tree embodies a finite element which can then be projected onto an isoparametric space through an affine transformation. All calculations for the domain integration are performed in this isoparametric space and then transformed back to the physical space.

Numerically speaking, the discrete problem (7) results in a matrix equation of the form of $\mathbf{Au} = \mathbf{b}$. Here $\mathbf{u} \in \mathbb{R}^{\text{NDOF}}$ is the vector of nodal unknowns whereas NDOF denotes the total number of nodal degrees of freedom in the FEM discretization. $\mathbf{A}$ is a square matrix and $\mathbf{b}$ a vector of matching size. A typical $3D$ problem, when solved through space-time method, will involve a $4D$ mesh which can be very large. The resulting $\mathbf{A}$ matrix, although sparse, will also be typically large. Therefore Krylov-type iterative solvers that rely on repeated matrix-vector multiplication (MATVEC) of $\mathbf{A}$ and $\mathbf{b}$ become extremely important for solving such matrix-vector systems. This also means that the matrix $\mathbf{A}$ need not be saved explicitly. Rather, the elemental submatrices resulting from the left hand side of Eq. 7 can be performed in each iteration of the Krylov method and multiplied to the elemental vector resulting from the right hand side (embodied through MATVEC). The next section provides a detailed description of these ideas and their implementations.

3 METHODOLOGY

In this section, we present our *matrix-free*, *mesh-free* algorithms for performing FEM computations on kD trees. Several state-of-the-art approaches [1, 3, 12, 31, 32, 34] use a comparison operator induced by a Space Filling Curve (SFC) to sort and partition the nodes of the tree[13]. The matrix-free FEM computations are widely used in the HPC community[1, 3, 30, 34], due to better scalability compared to the matrix-based approaches. Here matrix-free FEM approaches refer to FEM computations without the explicit assembly of matrices, instead providing a matrix-vector product (MATVEC). These are also particularly efficient for non-linear operators, which would require repeated assembly. For performing matrix assembly or a MATVEC we need neighborhood data structures such as *element to nodal mapping* or *nodal to nodal mapping*. We refer to such data structures as the mesh (connectivity information).

There are several major drawbacks of mesh-based numerical computations on adaptive trees. Firstly, since the elements are ordered based on the SFC, it is possible to use binary search to find neighbors among the list of elements. However, for large numbers of elements (as in the case of 4D), this can result in memory accesses forcing cache misses at least for the initial few accesses. Secondly, despite the locality of the SFC, neighbors are not contiguous in memory. During an assembly/MATVEC operation, the separation between the indices of neighbours will cause inefficient memory access. More importantly, the use of a mesh causes indirect memory access of the variable (`u[mesh[elem,node]]` instead of `u[node++]`), making it difficult for compilers to prefetch data. As the dimensionality k increases, building neighborhood data structures becomes extremely complex, increases storage, and makes the algorithm computationally expensive. The last factor is particularly important for problems requiring frequent mesh-refinements.

In order to overcome the above challenges we propose a mesh-free MATVEC that avoids these issues by computing the neighborhood information on the fly, similar to how matrix-free methods work without explicitly storing the matrix entries.

3.1 Comparison-free SFC based tree partitioning

Given that we are interested in generating a space-traversal, more so of application specific coordinates or regions, it is efficient to consider this problem as a traversal of the points in the SFC order. Specifically, we construct the tree in a *top-down* fashion, one level at a time, arranging the nodes at a given level based on the recurrence rules for the specific SFC (see Figure 3). We call this algorithm TREESORT (Algorithm 1). There are two main advantages for this approach: 1). The comparison-free property makes the structure of TREESORT independent of the SFC being used[11]. 2). The top-down traversal of the tree in breadth-first fashion, which is performed in the distributed TREESORT, gives us explicit control over the load-imbalance, which can be tuned as specified in [13]. All subsequent operations, such as tree construction, enforcing 2:1 balancing, and the MATVEC operation in FEM computations, are performed using top-down and bottom-up traversals with slight variations of the sequential TREESORT algorithm.

Algorithm 1 TREESORT

Require: A list of points or regions W, the starting level l_1 and the ending level l_2
Ensure: W is reordered according to the SFC.

```
counts[] ← 0                                   ▷ |counts| = 2^d, 16 for 4D
for w ∈ W do
    increment counts[child_num(w)]
counts[] ← R_h(counts)                         ▷ Permute counts using SFC ordering
offsets [] ← scan(counts)
for w ∈ W do
    i ← child_num(w)
    append w to W_i at offsets[i]
    increment offset[i]
if l_1 > l_2 then
    for i := 1 : 2^d do
        TreeSort(W_i, l_1 − 1, l_2)            ▷ local sort
return W
```

Algorithm 2 DISTTREESORT: Distributed TREESORT

Require: A distributed list of points or regions W_r, $comm$, p, p_r of current task in $comm$, tol load flexibility,
Ensure: globally sorted array W

```
counts_local ← [ ], counts_global ← [ ]
s[] ← TreeSort(W_r, l − log(p), l)             ▷ initial splitter computation
while |s_r − P_r N / p| > tol do
    counts[] ← 0                               ▷ |counts| = 2^d, 16 for 4D
    for w ∈ W do
        increment counts[child_num(w)]
    counts_local[] ← push(counts)
    counts[] ← R_h(counts)                     ▷ Permute counts using SFC ordering
    offsets [] ← scan(counts)
    for w ∈ W do
        i ← child_num(w)
        append w to W_i at offsets[i]
        increment offset[i]
    MPI_ReduceAll(counts_local, counts_global, comm)
    s[] ← select(s, counts_global)
MPI_AlltoAllv(A,splitters,comm)                ▷ Staged All2all
TreeSort(W_r, 0, l)                            ▷ local sort
return W_r
```

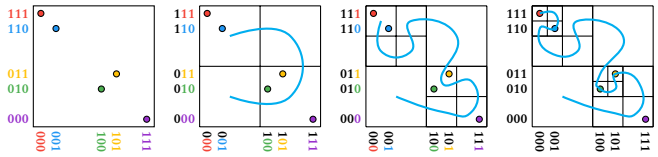

Figure 3: Bucketing each point and reordering the buckets based on the SFC ordering at each level l with top-down traversal. Each color-coded point is represented by its x and y coordinates. From the MSD-Radix perspective, we start with the most-significant bit for both the x and y coordinates and progressively bucket (order) the points based on these. The bits are colored based on the points and turn black as they get used to (partially) order the points.

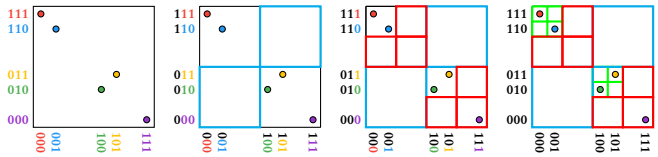

Figure 4: Equivalence of the MSD Radix sort with top-down quadtree construction when ordered according to space filling curves. Each color-coded point is represented by its x and y coordinates. From the MSD-Radix perspective, we start with the most-significant bit for both the x and y coordinates and progressively bucket (order) the points based on these. The bits are colored based on the points and turn black as they get used to (partially) order the points.Note that (■) denotes octants added at level 1, (■) denotes octants added at level 2, and (■) denotes octants added at level 3.

3.2 kD-Tree Construction

In this section, we describe how we can extend sequential and distributed approach of TREESORT to perform comparison-free construction. Contrasting with traditional SFC-based AMR algorithms,

Algorithm 3 TreeConstruction : Octree construction

Require: A list of points or regions W, the starting level l_1 and the ending level l_2, K_{oct}
Ensure: τ_c ordered complete octree based on W
1: $counts[] \leftarrow 0$
2: $\tau_c \leftarrow null$ ▷ $|counts| = 2^d$, 8 for $3D$
3: **for** $w \in W$ **do**
4: increment $counts[child\_num(w)]$
5: $counts[] \leftarrow R_h(counts)$ ▷ Permute counts using SFC ordering
6: offsets $[] \leftarrow scan(counts)$
7: **for** $w \in W$ **do**
8: $i \leftarrow child\_num(w)$
9: append w to W_i at offsets$[i]$
10: increment offset$[i]$
11: **if** $l_1 > l_2$ **then**
12: **for** $i := 1 : 2^d$ **do**
13: **if** $|W_i| > K_{oct}$ **then**
14: TreeConstruction(W_i, $l_1 - 1$, l_2)
15: **else**
16: $\tau_c.push(oct_i)$
17: **return** τ_c

TreeSort does not use any binary searches, which are inherently comparison-based. Instead of searches, we can perform a fixed number of streaming passes over the input data in a highly localized manner. The comparison-free approach of tree construction will reduce the random memory accesses and cache misses, leading to better memory performances. The key idea in TreeSort is to perform MSD radix sort, except that the ordering of digits is permuted at every level according to the SFC recurrence rules. A top-down traversal is equivalent to generating quadtrees (when $k = 2$) & octrees (when $k = 3$) as shown in figure 4. To construct trees, we perform top-down traversal with bucketing (see Figure 4), with the added constraint that any octant may contain at most K_{oct} of the input points; otherwise, it must be subdivided (see Algorithm 3). The distributed tree construction can be done using partitioning of the input points using TreeSort , then local construction of the tree, followed by elimination of duplicates across processors.

3.3 2 : 1 Balancing

In many applications involving adaptivity it is desirable to impose a restriction on the relative sizes of adjacent leaf nodes[32, 33]. There can be various reasons to enforce balancing constraints on the underlying tree, such as for better conditioning in the stiffness matrix and enforce a gradual change of refinement over the tree. The 2 : 1 balancing constraint enforces that two neighboring leaf nodes may differ by at most one level. The existing balancing approaches involve the use of a comparison operator in binary searches while the balancing constraint is enforced in several stages [32, 33]. The proposed approach computes a minimal necessary set of auxiliary nodes T_{aux}, which are are added to the distributed tree $\mathcal{T}$. After a second construction pass, the resulting tree is 2 : 1 balanced (i.e. see Figure 5).

3.4 Computing the unique node coordinates

In the mesh-free abstraction, the only pertinent information are the node-coordinates, i.e., the location of the nodal basis function. Once the sedectree has been constructed, nodes can be assigned to each sedecant (element) based on the order of the basis functions [23]. In order to generate a continuous Galerkin basis, we need to determine a unique set of node coordinates globally, i.e., on shared element faces, edges etc., we have duplicated node coordinates, and these duplicates need to be removed to have the node coordinate associated with a unique element. Additionally, when we have hanging nodes, only the node coordinates corresponding to the larger face/edge will exist, so this removal of node coordinates needs to be done as well. This is done in a single bottom-up traversal of the elements as illustrated in Figure 6. As shown, at each level as we go up the tree, we only need to resolve duplicates between the shared faces/edges of siblings, and this is a local operation. All interior nodes can be marked as unique, and so can the interior face nodes once the duplicates including testing for hanging nodes is resolved. Note that hanging nodes will be resolved at the level of the coarser element, so the decision is straightforward to make. At the end of the bottom-up traversal, we have a set of unique node coordinates that is all the information we will store and use for FEM computations.

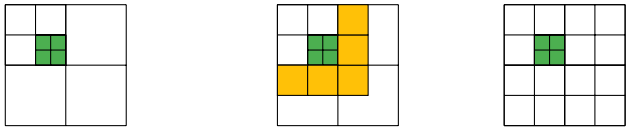

Figure 5: Left most figure shows an octree which violates the 2:1 balanced constraint, where the octants that cause the violation is showed in (■). In the middle figure auxiliary balanced octants are showed in (■), in other words these are the octants needed to remove the balance constraint violation in (■). Right most figure shows the constructed octree with auxiliary balanced octants which satisfies the 2:1 balance constraint.

Algorithm 4 AuxiliaryOctants : The set of auxiliary nodes to balance τ_c

Require: τ_c tree on domain Ω
Ensure: t_{aux} unique set of auxiliary nodes needed to balance τ_c
1: $t_{aux} \leftarrow \tau_c$
2: **for do** e in t_{aux}
3: $t_{aux}.add\_unique(N(P(e)))$
4: **return** t_{aux}

Algorithm 5 TreeBalancing: 2 : 1 tree balancing

Require: An tree τ_c on domain Ω, starting level l_1 and the ending level l_2.
Ensure: τ_b ordered 2 : 1 balanced tree
1: $K_{oct} \leftarrow 1$
2: $t_{aux} \leftarrow$ AuxiliaryOctants(t_{cons})
3: $\tau_b \leftarrow$ TreeConstruction(t_{aux}, l_1, l_2, K_{oct})
4: **return** τ_b

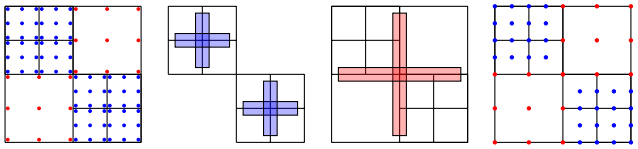

Figure 6: A simple example of how the cell nodes are placed for quadratic elements in a quadtree. The leftmost figure shows the *locally shared nodes*, which contain duplicates that need to be resolved in order to get *unique shared nodes* (the rightmost figure). We perform a single bottom-up pass of the tree resolving node conflicts, at the corresponding interior region indicated by the same color.

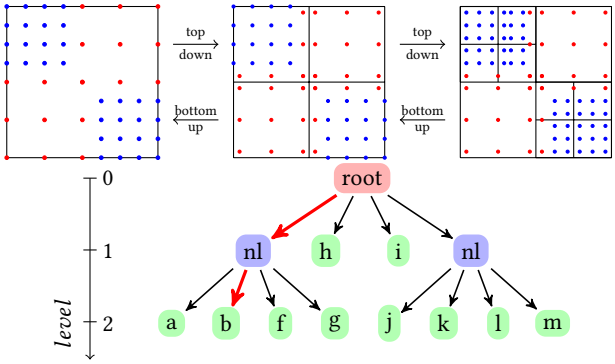

Figure 7: Illustration of top-down & bottom-up tree traversals for a $2D$ tree with quadratic element order. The leftmost figure depicts the unique shared nodes (nodes are color-coded based on level), as we perform top-down traversal nodes shared across children of the parent get duplicated for each bucket recursively, once leaf node is reached it might be missing elemental local nodes, which can be interpolated from immediate parent (see the rightmost figure). After elemental local node computations, bottom-up traversal performed while merging the nodes duplicated in during the top-down traversal.

3.5 Efficient SFC-based kD-tree traversal

In this section, we present an extension of TreeSort algorithm to perform efficient traversals on kD trees in order to perform FEM computations. The state-of-the-art approaches[4, 30, 32, 34] use lookup-tables such as element-to-node mappings, to perform matrix/vector assembly and matvec. In this section, we describe the top-down and bottom-up traversal of the kD-tree that are needed for performing these operations without the use of any lookup-tables.

top-down: In the top-down traversal for a given non-leaf tree node e, we bucket the corresponding dof (degree of freedom) coordinates to the children of e, duplicating any coordinates that are incident on multiple children (see Figure 7). This process is repeated recursively until we reach a leaf tree node. A leaf node is detected by counting the number of dof coordinates in the bucket. Some buckets might have fewer than the prescribed number of dofs for an element of the specified order, if there are hanging nodes. Since the nodes of the parent are available at this level, we obtain the missing hanging nodes by interpolation. We recurse in a depth-first fashion so as to expose memory locality suited for deep memory hierarchies.

bottom-up: Once we reach a leaf node, we can perform elemental operations (*e.g.* elemental matvec), and if the computation requires an accumulation, nodal values are merged in the reverse direction of the duplication in top-down approach (see 7).

3.6 Matrix-free implementation on $4D$ meshes

As mentioned in the previous section, we will only store the $4D$ coordinates and not any maps (such as element-to-node maps). There are two major reasons for this. Firstly, these maps are very expensive to construct and will have a large memory footprint for $4D$ meshes. Secondly, using maps, the variables of interest during FEM computations must be accessed and updated via indirect memory accesses. For large $4D$ meshes in particular, such indirect memory access is likely to be extremely inefficient on modern architectures with high levels of parallelism and deep memory hierarchies. As a remedy, we propose a *mesh-free* approach that makes use of the quasi-structured nature of sedectrees and enables direct access to data. We explain this approach in detail and provide evidence for the efficacy of this approach.

We will illustrate a matvec with the transient diffusion operator $(\frac{\partial}{\partial t} + \nabla^2)$, given in a discrete form as K, i.e., we will compute $v = Ku$. Here u is the scalar unknown defined over our space-time domain, i.e., there is one unknown per node (coordinate point). Therefore the input to our matvec will be the `real` vector u and another vector of points $\mathbf{p} = (x, y, z, t)$ (4× `unsigned int`). The output will be the vector v, the same size as u such that $v = Ku$. Unlike conventional FEM codes, we will evaluate v without requiring indirect memory accesses to u, v or $\mathbf{p}$. Note that our approach becomes significantly more effective for systems with multiple dofs per spatio-temporal point, as these all will use the same coordinate information.

Since we do not have a mesh, we will have to extract the required information on the fly. Similar to the sedectree construction, (Figure 4), we proceed in a top-down fashion using the radix sort. This is particularly efficient since we have the x, y, z and t coordinates as `unsigned ints`. Also, since the coordinates and the unknowns are arranged using space filling curves, there is high locality. In the most significant digit (MSD) radix, we examine the bits, from most to least significant, to bucket the data. In our case, at each level we use one bit each from the x, y, z and t coordinates to bucket the points ($\mathbf{p}$) and unknowns u. We then recurse within each bucket. This happens in a depth-first fashion that in combination with the locality of space filling curves, make the overall data-access amenable to modern deep memory hierarchies. Bucketing within radix sort involves only direct memory in a streaming fashion and requires one cache-line for accessing the input and one for each bucket.

Bucketing for the matvec is a bit more involved, so we first illustrate in $2D$ in Figure 7. Here we need to bucket the interior points and replicate the interior shared faces and the interior corner as well. In $4D$, we need to bucket the volumes (faces in 3D) and the interior faces, edges and volumes as these dofs need to be replicated across octants. Once replicated, the sedecants are

Algorithm 6 `matvec` : Mesh-free 4*D* FEM MATVEC

Require: x unique coordinates, u, v input and output vectors and level l.
Ensure: $v = Ku$, where K denotes FEM discrete operator.

```
v ← 0
(U, X, V) ← scatter_to_buckets(u, x, v, l)
for (u_i, x_i, v_i) in (U, X, V) do
    if length(x_i) == n then                    ▷ reached leaf tree node
        v_i ← k_e u_i                           ▷ K_e is elemental matrix
    else
        matevc(u_i, x_i, v_i, l − 1)            ▷ recurse, top-down
gather_from_buckets(v_i, x, v, l)               ▷ bottom-up
return
```

Algorithm 7 `scatter_to_buckets` : Mesh-free 4*D* FEM MATVEC

Require: x unique coordinates, u, v input and output vectors and level l.
Ensure: x, u, v get scattered to the children buckets (X, U, V) with duplication as needed.

```
cnt[] ← zeros(nb + 1)                           ▷ nb number of buckets
for x_i ∈ x do
    cnt[x_i & (1 << l) + 1] + +                 ▷ Figure cout the child bucket sizes
for (x_i, u_i) ∈ (x, u) do
    idx ← cnt[x_i & (1 << l)] + +
    U[idx], X[idx], V[idx] ← u_i, x_i, v_i      ▷ perform scatter
return X, U, V
```

Algorithm 8 `gather_from_buckets` : Mesh-free 4*D* FEM MATVEC

Require: x unique coordinates, v_i scattered input, v vector corresponding to x and level l.
Ensure: scattered input v_i is accumulated to the vector v

```
cnt[] ← zeros(nb + 1)                           ▷ nb number of buckets
for i ∈ [0, length(x)) do
    idx = cnt[x[i] & (1 << l)] + +
    v[i]+ = v_i[idx]
return
```

independent of each other and can recurse independently. This expresses a very fine-grained parallelism not possible with traditional FEM MATVECs. As previously explained, we identify reaching the leaf node based on when all dofs correspond to the nodes of a single element, potentially with interpolation in case of hanging nodes. Having reached the leaf node, we apply the elemental operator to compute $v_e = K_e u_e$. On the return, the results in v are accumulated from all children. This is the opposite of the duplication of u prior to the recursion. The simplified pseudocode for the MATVEC is presented in Algorithms 6,7 & 8.

The mesh-free MATVEC approaches the computation in a data-first fashion and is structured based on the data dependencies and the associated data movement. Note that in the distributed memory setting, we follow a similar principle and exchange ghost or halo regions, albeit using additional lookup tables. Since the mesh-free approach exposes such parallelism in a hierarchical fashion (due to the tree structure), the same basic algorithm holds for the distributed memory cases as well, except that the bucketing at the inter-process level will require MPI communication. Again, unlike traditional codes, this can be done without any additional lookup tables (scatter maps). Also note that the resulting code does not have any indirect memory accesses to the large data arrays u and v. This makes implementations simple and easy enough for modern compilers to optimize (such as prefetching, vectorization, etc.) without special architecture specific tuning of the code.

4 RESULTS

We present a thorough evaluation of our 4*D* adaptive framework in this section. We start by giving a brief overview of the hardware and software setup used for this evaluation. Additional information can be found in the appendices. We can demonstrate the improved parallel scalability that can be obtained by using a 4*D* formulation instead of the traditional $3D + t$ formulation in §4.2. We then demonstrate the ability to handle different classes of PDEs and the use of *a posteriori* error estimates to facilitate the refinement in spacetime in §4.3. Finally, in §4.4, we conduct weak and strong scalability experiments for the tree construction, balancing and for performing a MATVEC; the fundamental building blocks for all our test applications presented in §4.3.

4.1 Experimental Setup:

The large scalability experiments reported in this paper were performed on Titan and Stampede2. Titan is a Cray XK7 supercomputer at Oak Ridge National Laboratory (ORNL) with a total of 18,688 nodes, each consisting of a single 16-core AMD Opteron 6200 series processor, with a total of 299,008 cores. Each node has 32GB of memory. It has a Gemini interconnect and 600TB of memory across all nodes. Stampede2 at the Texas Advanced Computing Center (TACC), University of Texas at Austin has 4, 200 Intel Xeon Phi 7250 (KNL) compute nodes, each with 96GB DDR4 RAM and 16GB of MCDRAM and 1, 736 Intel Xeon Platinum 8160 (SKX) compute nodes with 2×24 cores and 192GB of RAM per node. Stampede2 has a 100Gb/sec Intel Omni-Path (OPA) interconnect in a fat tree topology. We used the SKX nodes for the experiments reported in this work.

4.2 Space-time solution with heavy parallelism

As mentioned in section 2, equation 1 represents a wide class of partial differential equations. More recognizable PDEs can be obtained by enforcing certain assumptions on $\mathbf{a}$, κ and f. For example, the linear heat equation is obtained by setting $\mathbf{a} = (0, 0, 0)$, $\kappa = \kappa(x, y, z)$ and $f = f(x, y, z)$. Then setting $\mathbf{a} = \mathbf{a}(x, y, z)$ renders it an advection-diffusion equation. And gradually, many other such combinations will lead to some other specific equation, for which the formulation presented in section 2 will remain the same.

First, to demonstrate the central theme of parallelism in space-time coupled analysis, we choose the simple linear heat equation. As mentioned before, this is done by setting $\mathbf{a} = (0, 0, 0)$, $\kappa = 1$ in equation 1. The force f on the right hand side is manufactured by assuming the solution u to be

$$u(x, y, z, t) = e^t \sin(\pi x) \sin(\pi y) \sin(\pi z) \tag{13}$$

Once we have the corresponding discrete bilinear form, the problem is then translated into a linear algebra problem using the 4*D* formulation described in §2 and §3. The final linear algebra problem is solved using PETSc 3.7 via the `MATSHELL` interface to expose the mesh-free matrix-free interface. The convergence results for this problem on a series of uniformly refined meshes is plotted in figure 2.

Figure 8 shows the solve time comparison between a time stepping and a space-time problem on Titan. The time stepping problem

solves the PDE posed on a 3D spatial domain for a prescribed number of time steps through the well known Crank-Nicholson scheme. The space-time counterpart of this problem is posed in a 4D space-time domain and is solved using the formulation presented in §2. The linear systems in these cases were solved with GMRES with a block Jacobi preconditioner. The space-time problem performs badly when the number of cores used is low and in this region the time-stepping method is much more efficient. But as the number of cores increase we notice a linear decrease in the solve time for the space-time problem whereas the performance of the time-stepping method deteriorates slowly in the beginning and rapidly when the number of cores go above 5000. This shows concretely the potential of the space-time method with heavy parallelism. By adding the time dimension into the formulation, we are making the problem more complex from the point of view of computation, but this increased complexity in turn allows us to leverage the benefits of high parallelism. Indeed, when number of cores 8000 and above then space-time formulation beats the sequential time stepping solution time.

4.3 Space-time adaptive refinement applied to different classes of PDE

The other important theme of the space-time analysis is the adaptive refinement strategy applied to diverse PDEs of interest. These PDEs produce solutions that exhibit a high degree of localization in both space and time. This idea is clear by looking at Figure 10. This figure presents the adaptive solutions to three different classes of PDEs. The first one is the linear heat equation shown in Figures 10a, 10d (i.e., the first column of Figure 10), where a Gaussian heat pulse localized in space diffuses in time. Figure 10a shows the contour plot of the non-zero solution. The solution is zero in most of the space-time domain. This problem is a perfect candidate where space-time analysis with adaptive refinement can be applied. Figure 10d reveals an important feature about the space-time refinement. It shows a 2D slice of the whole mesh; the $y-t$ plane, in particular. In this slice, we observe that different regions in space (points with different y-values at a particular time level) experience different "time-step" sizes. Such spatially varying time-stepping approaches are extremely non-trivial to implement and execute in a sequential setting.

The next two columns in Figure 10 illustrate the same concept, but using two increasingly challenging PDEs. The second column shows the solution to the linear **advection-diffusion equation** (Figures 10b and 10e) where a discontinuous pulse moves in a rotating field with a very small amount of diffusivity (i.e. very large Peclet number). This translates to a spiral in the space-time domain. Notice that the space-time region around the pulse is highly refined in space-time (as shown in the second image in the bottom row). The third column illustrates a slice of the **nonlinear Allen-Cahn equation** (Figures 10c, 10f, the third column) which models the movement of a solidification front (starting from a sphere, and shrinking in size). The $y-t$ slice cleanly exhibits the advantage of the space-time approach in tracking the solidification front. The adaptivity in the case of linear heat and linear advection diffusion equation is facilitated by the *a posteriori* error estimate derived in 12. In the case of the Allen-Cahn equation, we simply refine the interface in space-time, and hence the the refinement in space and time ensures that the interface is resolved accurately.

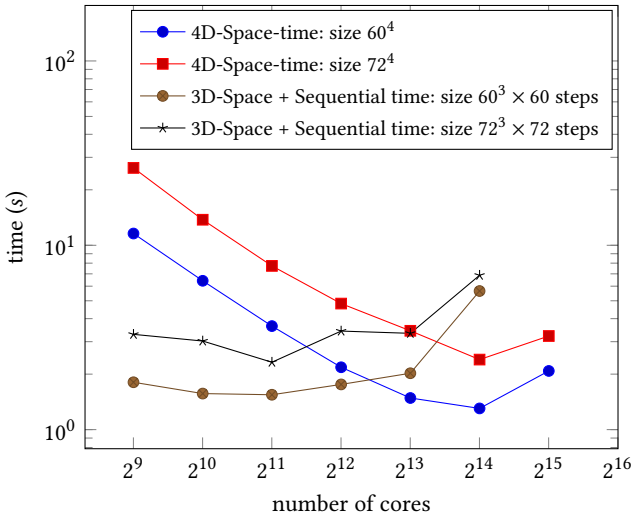

Figure 8: Comparison of scaling of Crank Nicholson time stepping on Titan for linear diffusion problem with coupled space-time formulation. The coupled space-time formulation scales up to 16384 processors whereas the time stepping scales only up to 2048 processors.

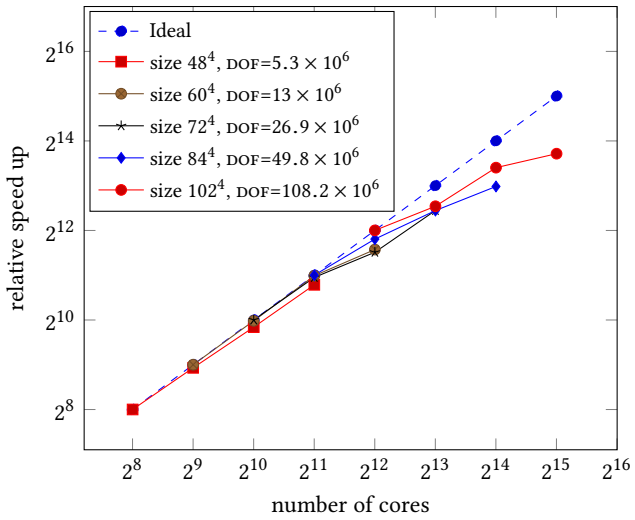

Figure 9: Relative speedup of 4D space - time formulation on Titan. The number of degree of freedom was varied from 5.3 million to 108.2 million across 16384 processors.

4.4 Scalability

Adaptive 4*D*-**trees**: We present scalability results for building adaptive 4D-trees in order to perform space-time FEM computations. The adaptive trees are produced by randomly generated points

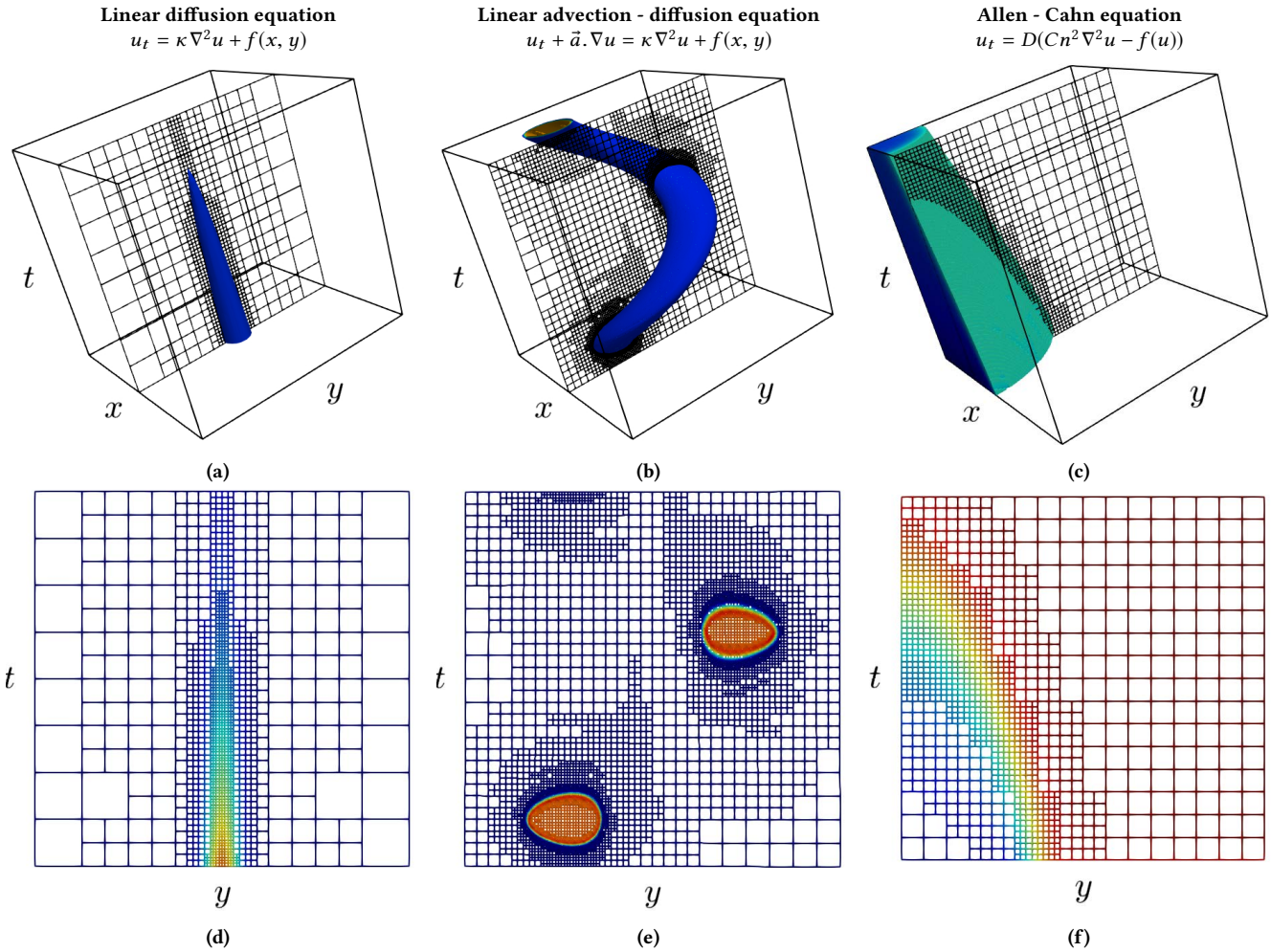

Figure 10: Examples illustrating space-time adaptive refinement for three different PDEs. The top row shows the solution in space-time, while the bottom row shows $y - t$ slices across interesting regions in space-time. The first column (Figs a,d) shows the diffusion of a heat source. With increasing time, the source diffuses out across the domain needing less resolved elements. The second column (Figs b,e) shows the action of a rotating flow field on an initial concentration source, when diffusivity is very low. The source is advected around in the domain, with minimal change in peak concentration due to the low diffusivity. This is clearly seen in Fig (b). Fig(e) shows a $y - t$ slice illustrating the highly resolved space-time elements close to the source. The third column (Figs c,f) shows a shrinking solidification front modeled via the non-linear Allen-Cahn equations. Fig (f) shows that the interface between the two phases is well refined in space-time.

based on the normal distribution. Randomly generated distributed point set does not obey the SFC based partitioning. Therefore we use, DistTreeSort to perform a re-partitioning, respecting the SFC, which is followed by TreeConstruction and TreeBalancing to ensure that the generated tree is complete and $2:1$ balanced. To perform FEM computations, we need to compute the shared unique nodes (see §3.4) and the communication map, which is required to perform halo/ghost node exchange during matvec computation. The Figure 11 presents the weak scalability results across $6,144$ cores on Stampede2 for complete adaptive $4D$-tree construction. Note that even though we have used 500 and 250 random points per core, after construction and balancing, the resulting trees have roughly $5K$ elements per core and after unique node computation $3K$ & $50K$ unknowns per core for linear and quadratic cases respectively.

FEM matvec : The core computational kernel for all problems considered in this work is the matvec. This is the basic building block that determines the overall parallel performance and scalability. In this section we present scalability results for performing a $4D$ matvec on an adaptive mesh. Note that for typical cases, the overall performance and scalability will be dominated by the matvec and not the cost of remeshing. In Fig.13 we present weak scaling results for the matvec on Stampede2 using both linear and quadratic basis functions. The breakdown of the total time is given in terms of the

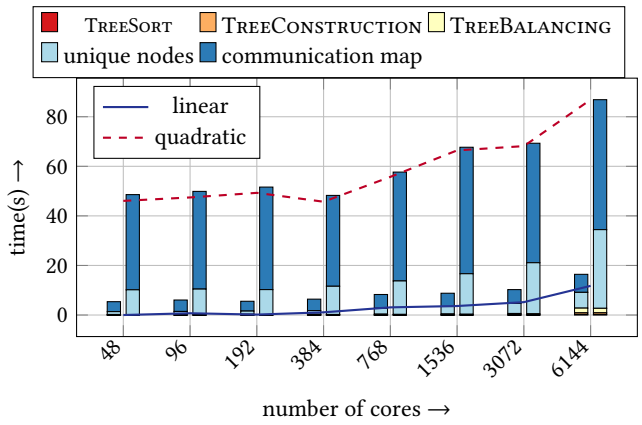

Figure 11: Weak scaling results for distributed sorting and partitioning, $4D$ adaptive tree construction, $2:1$ balancing, computing unique nodes and building the communication scatter map for randomly generated coordinates (using normal distribution) with 500 &250, points per core for linear and quadratic element order across 6, 144 cores in TACC's Stampede2.

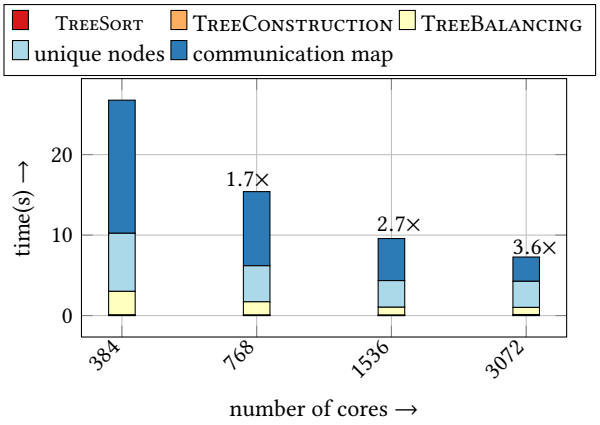

Figure 12: Strong scaling results for distributed sorting and partitioning, $4D$ adaptive tree construction, $2:1$ balancing, computing unique nodes and building the communication scatter map for a problem size of $3M$ randomly generated coordinates (using normal distribution) for linear order across 3, 072 cores in TACC's Stampede2.

major steps, including communication, the top-down and bottom-up traversals, and the cost of the elemental MATVEC. One can see that the elemental MATVEC and the communication dominate the overall time, for both the linear and quadratic discretizations. These costs would be encountered in any distributed memory FEM implementation. The bottom-up and top-down traversals account for the overhead of adaptivity and performing the MATVEC in a matrix-free and mesh-free fashion. The top-down traversal involves data duplication and is therefore more expensive than the bottom-up traversal. There is room for improvement here by using more efficient memory allocation and layout to reduce the cost of the top-down traversal. Note that the overhead from the top-down traversal is comparable to the overhead one would obtain from a mesh-based approach (*e.g.* [12] reports $20-25\%$ overhead for 3D octree-refined case).

A major motivation for adopting a 4D formulation is to expose parallelism in time and reduce time to solution by increasing number of processes. In other words, strong scaling is extremely important. In Fig. 14 we demonstrate excellent strong scalability for the $4D$ matvec on an adaptive mesh. Again as in the weak-scaling experiments, the communication and elemental MATVEC dominate the overall time and scale well. Going from 384 to 6144 cores on Stampede (16×), we get speedups of 10.5× for linear and 9.9× for quadratic discretizations.

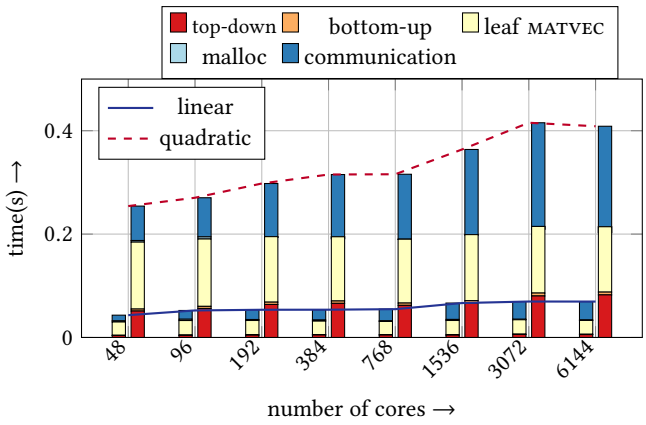

Figure 13: Weak scaling results *mesh-free, matrix-free* FEM MATVEC on an adaptive $4D$ tree on TACC's Stampede2 across 6,144 cores with linear and quadratic basis functions. For the largest problem sizes for linear and quadratic cases with 16M and 340M degrees of freedoms respectively. Execution times reported are averaged over 10 MATVEC operations, where the average number of unknowns per core were 3K and 50K for linear and quadratic cases, respectively.

Figure 9 shows the relative speedup of the 4D space-time problem on Titan by varying the number of cores from 256 to 32768. The case with 102 elements in each direction (108.2 Million degree of freedom) scales up to 32768 cores. This is a strong scaling result and shows considerably good performance. It must be noted that this shows a very high potential to exploit the parallelism on the current scale of machines, which is not possible with the time stepping method.

5 CONCLUSION & FUTURE WORK

In this work we have presented an innovative $4D$ formulation for solving time-dependent PDE problems. We combined this within an adaptive setting and with variable order discretizations to realize a powerful framework. Our choice of algorithms specifically targets modern architectures with deep-memory hierachies and high levels of parallelism. We demonstrated improved strong and weak scaling as a result of the $4D$ formulation that is able to utilize parallel resources more efficiently compared to the traditional $3D$+time approach, where time is treated sequentially. Additionally, we derived both *a priori*, as well as residual-based *a posteriori* error estimates

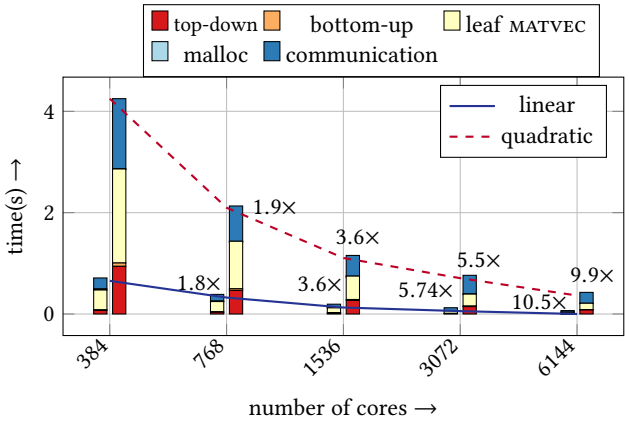

Figure 14: Strong scaling results for fixed problem sizes of 16M (linear) and 340M (quadratic) unknowns for *mesh-free, matrix-free* FEM MATVEC on an adaptive $4D$ tree on TACC's Stampede2 across 6,144 cores with linear and quadratic basis functions. Execution times reported are averaged over 10 MATVEC operations.

for the linear time dependent heat diffusion equation, and numerically illustrated improved convergence behavior of the space-time solution approach. By developing sophisticated numerical methods backed by theoretical guarantees of improved convergence, as well as innovative mesh- and matrix-free algorithms to enable efficient mapping to modern architectures, we have demonstrated impressive scalability results on current supercomputers. We believe our methods are a first step towards improving the scalability of PDE solvers to the next generation of supercomputers.

6 ACKNOWLEDGMENTS

This work was funded by National Science Foundation grants OAC-1808652 and PHY-1912930. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC05-00OR22725 and the Extreme Science and Engineering Discovery Environment (XSEDE) allocation TG-PHY180002.

REFERENCES

[1] [n. d.]. MFEM: Modular Finite Element Methods Library. mfem.org. https://doi.org/10.11578/dc.20171025.1248
[2] Mark Ainsworth and J. Tinsley Oden. 2000. *A Posteriori Error Estimation in Finite Element Analysis.* John Wiley & Sons, New York.
[3] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. 2018. The `deal.II` Library, Version 9.0. *Journal of Numerical Mathematics* 26, 4 (2018), 173–183. https://doi.org/10.1515/jnma-2018-0054
[4] David A. Bader and Guojing Cong. 2005. A fast, parallel spanning tree algorithm for symmetric multiprocessors (SMPs). *J. Parallel and Distrib. Comput.* 65, 9 (2005), 994 – 1006. https://doi.org/10.1016/j.jpdc.2005.03.011
[5] Marek Behr. 2008. Simplex space–time meshes in finite element simulations. *International journal for numerical methods in fluids* 57, 9 (2008), 1421–1434.
[6] V Carey, D Estep, August Johansson, M Larson, and S Tavener. 2010. Blockwise adaptivity for time dependent problems based on coarse scale adjoint solutions. *SIAM Journal on Scientific Computing* 32, 4 (2010), 2121–2145.
[7] V. Carey, D. Estep, A. Johansson, M. Larson, and S. Tavener. 2010. Blockwise Adaptivity for Time Dependent Problems Based on Coarse Scale Adjoint Solutions. *SIAM J. Sci. Comput.* 32, 4 (2010), 2121–2145. https://doi.org/10.1137/090753826
[8] J Austin Cottrell, Thomas JR Hughes, and Yuri Bazilevs. 2009. *Isogeometric analysis: toward integration of CAD and FEA.* John Wiley & Sons.
[9] Robert Dyja, Baskar Ganapathysubramanian, and Kristoffer G van der Zee. 2018. Parallel-In-Space-Time, Adaptive Finite Element Framework for Nonlinear Parabolic Equations. *SIAM Journal on Scientific Computing* 40, 3 (2018), C283–C304.
[10] Robert D Falgout, Stephanie Friedhoff, Tz V Kolev, Scott P MacLachlan, and Jacob B Schroder. 2014. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing* 36, 6 (2014), C635–C661.
[11] Milinda Fernando, Dmitry Duplyakin, and Hari Sundar. 2017. Machine and Application Aware Partitioning for Adaptive Mesh Refinement Applications. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '17).* ACM, New York, NY, USA, 231–242. https://doi.org/10.1145/3078597.3078610
[12] M. Fernando, D. Neilsen, H. Lim, E. Hirschmann, and H. Sundar. 2019. Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals. *SIAM Journal on Scientific Computing* 41, 2 (2019), C97–C138. https://doi.org/10.1137/18M1196972 arXiv:https://doi.org/10.1137/18M1196972
[13] Milinda Fernando and Hari Sundar. 2016. Fast Algorithms for Distributed Ordering and Partitioning using Hilbert Curves. (2016). in preparation.
[14] Leopoldo P Franca, G Hauke, and A Masud. 2004. Stabilized finite element methods. International Center for Numerical Methods in Engineering (CIMNE), Barcelona âĂę.
[15] Baskar Ganapathysubramanian and Nicholas Zabaras. 2005. Control of solidification of non-conducting materials using tailored magnetic fields. *J. Cryst. Growth* 276, 1–2 (2005), 299–316. https://doi.org/10.1016/j.jcrysgro.2004.11.336
[16] Baskar Ganapathysubramanian and Nicholas Zabaras. 2005. On the control of solidification using magnetic fields and magnetic field gradients. *Int. J. Heat Mass Transfer* 48, 19–20 (2005), 4174–4189. https://doi.org/10.1016/j.ijheatmasstransfer.2005.04.027
[17] Martin J. Gander. 2015. 50 Years of Time Parallel Time Integration. In *Multiple Shooting and Time Domain Decomposition Methods: MuS-TDD, Heidelberg, May 6-8, 2013*, Thomas Carraro, Michael Geiger, Stefan Körkel, and Rolf Rannacher (Eds.). Springer International Publishing, Cham, 69–113. https://doi.org/10.1007/978-3-319-23321-5_3
[18] Herman J. Haverkort. 2012. Harmonious Hilbert curves and other extradimensional space-filling curves. *CoRR* abs/1211.0175 (2012). arXiv:1211.0175 http://arxiv.org/abs/1211.0175
[19] Jeffrey Hittinger, Sven Leyffer, and Jack Dongarra. 2013. Models and Algorithms for Exascale Computing Pose Challenges for Applied Mathematicians. SIAM News.
[20] Thomas JR Hughes and Gregory M Hulbert. 1988. Space-time finite element methods for elastodynamics: formulations and error estimates. *Computer methods in applied mechanics and engineering* 66, 3 (1988), 339–363.
[21] Thomas JR Hughes, Guglielmo Scovazzi, and Leopoldo P Franca. 2018. Multiscale and stabilized methods. *Encyclopedia of Computational Mechanics Second Edition* (2018), 1–64.
[22] Thomas JR Hughes and James R Stewart. 1996. A space-time formulation for multiscale phenomena. *J. Comput. Appl. Math.* 74, 1-2 (1996), 217–229.
[23] David A Kopriva. 2009. *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers.* Springer Science & Business Media.
[24] Ulrich Langer, Stephen E Moore, and Martin Neumüller. 2016. Space–time isogeometric analysis of parabolic evolution problems. *Computer methods in applied mechanics and engineering* 306 (2016), 342–363.
[25] J. Lions, Yvon Maday, and Gabriel Turinici. 2001. A"parareal"in time discretization of PDE's. *C. R. Acad. Sci. Series I Mathematics* 332, 7 (2001), 661–668.
[26] Robert B Lowrie, Philip L Roe, and Bram Van Leer. 1998. Space-time methods for hyperbolic conservation laws. In *Barriers and Challenges in Computational Fluid Dynamics.* Springer, 79–98.
[27] Karthik Mani and Dimitri Mavriplis. 2011. Efficient solutions of the euler equations in a time-adaptive space-time framework. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition.* 774.
[28] JP Pontaza and JN Reddy. 2003. Spectral/hp least-squares finite element formulation for the Navier–Stokes equations. *J. Comput. Phys.* 190, 2 (2003), 523–549.
[29] Thomas CS Rendall, Christian B Allen, and Edward DC Power. 2012. Conservative unsteady aerodynamic simulation of arbitrary boundary motion using structured and unstructured meshes in time. *International Journal for Numerical Methods in Fluids* 70, 12 (2012), 1518–1542.
[30] Johann Rudi, A. Cristiano I. Malossi, Tobin Isaac, Georg Stadler, Michael Gurnis, Peter W. J. Staar, Yves Ineichen, Costas Bekas, Alessandro Curioni, and Omar Ghattas. 2015. An Extreme-scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth's Mantle. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15).* ACM, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/2807591.2807675
[31] Hari Sundar, Dhairya Malhotra, and George Biros. 2013. HykSort: A New Variant of Hypercube Quicksort on Distributed Memory Architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS '13).* ACM, New York, NY, USA, 293–302. https://doi.org/10.1145/2464996.2465442

[32] Hari Sundar, Rahul Sampath, and George Biros. 2008. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing* 30, 5 (2008), 2675–2708. https://doi.org/10.1137/070681727
[33] Hari Sundar, Rahul S. Sampath, Santi S. Adavani, Christos Davatzikos, and George Biros. 2007. Low-constant Parallel Algorithms for Finite Element Simulations using Linear Octrees. In *SC'07: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. ACM/IEEE.
[34] Hari Sundar, Rahul S Sampath, and George Biros. 2008. Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel. *SIAM J. Sci. Comput.* 30, 5 (2008), 2675–2708. https://doi.org/10.1137/070681727
[35] Tayfun E Tezduyar, Sunil Sathe, Ryan Keedy, and Keith Stein. 2006. Space–time finite element techniques for computation of fluid–structure interactions. *Computer methods in applied mechanics and engineering* 195, 17-18 (2006), 2002–2027.
[36] Rüdiger Verfürth. 2008. *A Posteriori Error Estimation Techniques for Finite Element Methods*. Oxford University Press, Oxford.
[37] Luming Wang and Per-Olof Persson. 2015. A high-order discontinuous Galerkin method with unstructured space–time meshes for two-dimensional compressible flows on domains with large deformations. *Computers & Fluids* 118 (2015), 53–68.

7 ARTIFACT DESCRIPTION

7.1 Getting and Compiling Dendro-KT

The Dendro-KT simulation code is freely available at GitHub (link will be disclosed after review process) under the GNU General Public License (GPL). The latest version of the code can be obtained by cloning the repository

```
$ git clone https://github.com/paralab/Dendro-KT.git
```

The following dependencies are required to compile Dendro-KT

- C/C++ compilers with C++11 standards and OpenMP support
- MPI implementation (e.g. openmpi, mvapich2)
- ZLib compression library (used to write `.vtu` files in binary format with compression enabled)
- BLAS and LAPACK are optional and not needed for current version of Dendro-KT
- CMake 2.8 or higher version

After configuring Dendro-KT, generate the Makefile (use `c` to configure and `g` to generate). Then execute `make all` to build all the targets. Please follow the instruction page on the repository for more details on installation.

7.2 Experiments

All the experiments reported using Stampede2 are executed in the following module environment.

```
Currently loaded modules :
1) intel/18.0.0 2) impi/18.0.0 3) cmake/3.10.2
4) git/2.9.0 5) gsl/2.3 6) autotools/1.1
```

Experiments performed in Titan are executed in the following module environment.

```
Currently Loaded Modulefiles:
  1) modules/3.2.6.6        4) petsc/3.8.4
  2) PrgEnv-intel           5) cmake/3.10.2
  3) cray-mpich2/6.1.1.
```

matvec and TreeSort experiments: Both strong and weak scalaing results were executed using, `matvecBenchAdaptive` and `tsortBench` which requires following parameters.

- `pts_per_core` : Number of random generated points per core.
- `maxdepth` : Maximum refinment level.
- `order`: Order of the FEM basis functions.
- `iter`: Number of iterations to be averaged over when benchmarking run times.

In order to run the benchmark, please use the following command.

```
ibrun -np <mpi tasks> ./<benchmark>
    pts_per_core maxdepth order iter
```

Space-time adaptivity linear and non-linear PDEs: The adaptive refinement experiments mentioned in §4.3 additionally requires `PETSC` library for solving the result in linear system by space-time FEM. The Dendrite simulation code can be obtained from cloning the repository

```
$ git clone https://bitbucket.org/baskargroup/ad_dendrite.git
```

The codes for all the experiments are provided in the examples folder and can be run by the following command:

```
$ cd examples/XXX/io*;
$ bash run_case.sh
```

7.3 Stampede2 compute node configuration

```
TACC_SYSTEM=stampede2
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
CPU(s):                96
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:Intel(R)Xeon(R)
Platinum 8160 CPU @ 2.10GHz
CPU MHz:               2100.000
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              33792K

MemTotal:       196438176 kB
MemFree:        191589692 kB
MemAvailable:   190917852 kB
```

7.4 Titan compute node configuration

```
Architecture:          Cray XK7
Processor:             16-Core AMD
Memory/node:           32GB
```

7.5 4D-Hilbert curve

The comparison-free structure of TreeSort allows us to use any SFC. In our work we are interested in the Hilbert curve.

The Hilbert curve is an SFC with the property that adjacent segments in the curve are mapped to adjacent regions in space. Previous works have exploited this property to better preserve spatial locality in linearized octrees, thus curbing the communication cost of partitioning a spatial domain [13].

Extending the Hilbert curve to any dimension beyond 2D is nontrivial. Although a known three-dimensional extension was used previously [13], certainly the situation is more complicated in four dimensions. Not only does it become tedious to write (and reason about) a permutation table of recurrence rules, but also the Hilbert curve does not have a unique high-dimensional extension. One must choose a definition among the many possible SFCs that satisfy the above locality property.

Haverkort [18] provides

- an additional constraint ("interdimensional consistency") that uniquely characterizes the so-called *harmonious* Hilbert curve in any dimension;
- a systematic constructive definition for the curve in any dimension.

(The idea of interdimensional consistency is that the restriction of the curve to a face should be the same curve, but a lower-dimensional version. Potential benefits include hardware optimization and/or better locality [18].)

Based on Haverkort's abstract recurrence rules, we can programmatically generate SFC rotation tables for the Hilbert curve in four dimensions, and higher.

Implementation. A recursive definition of the Hilbert curve specifies a traversal of the unit hypercube in terms of sub-traversals of each child orthant.

The recurrence rule must do two things. It must specify the order in which children are traversed; and it must also specify the orientations of the children's coordinate frames relative to the parent. Depending on these two specifications, various SFCs may be produced. Some of them can be considered valid extensions of the Hilbert curve. Exactly one of them, according to Haverkort, produces an interdimensionally consistent family of extensions to the Hilbert curve.

To set the stage, assume a coordinate system for the unit K-cube, with axes numbered $i \in \{0, \ldots, K-1\}$ and origin at the center of the K-cube. Magnitudes are irrelevant; we are concerned with the signs of coordinates only. We represent a coordinate tuple as a bit string, $x \in \mathbb{B}^K$, where $\mathbb{B} \equiv \{0, 1\}$, '1' meaning '+' and '0' meaning '-'. Each child orthant has a unique coordinate string, relative to the parent frame, that, as an integer, is precisely the child number in lexicographic order: $c = \sum_i x_i 2^i$.

As for the traversal order, the Hilbert curve follows the "reflected Gray code" [18]. In our implementation, the r^{th} visited child is

$$c \leftarrow (r >> 1) \text{ XOR } r$$

where ">>" is the bit-wise right shift operator and "XOR" is the bit-wise XOR operator.

The orientation of the r^{th} visited child is described by a permutation of, followed by reflections of, the parent coordinate axes. The axis permutation depends on the parity of r and the coloring of axes as r is read as a bit string. Starting from a reverse ordering of axes, if r is even, then even-colored axes are collected in the back, but if r is odd then odd-colored axes are collected in the back. The reflection m of axes (a bit string, '1' meaning reflect) can be defined in terms of c and r:

$$m \leftarrow ((r-1) >> 1) \text{ XOR } (r-1);$$

$$m \leftarrow (m \text{ AND } -2) \text{ OR } ((\text{ NOT } c) \text{ AND } 1);$$

The above recurrence rule characterizes the SFC relative to a local coordinate frame. The final lookup table must describe the various orientations of the recurrence rule in terms of an absolute coordinate frame.

To generate such a table we define a multiplication operator that transforms the recurrence rule to another coordinate frame, and then we fill out the group closure until all recurrence rules are defined in terms of previously computed recurrence rules. The base case is to take the absolute frame as the local frame, that is, the unmodified definition. The multiplication operator is realized

through the semi-direct product:

$$(MA)(ma) = MAm(A^{-1})(A)a = M(AmA^{-1})(Aa)$$

where a and A are axis permutations and m, M, and AmA^{-1} are axis reflections.

8 ARTIFACT EVALUATION

8.1 Scalability study on $4D$-trees & MATVEC

Prior to performing FEM computations, discretization and partitioning the space-time domain is performed using $4D$-trees. We have used TreeSort, TreeConstruction, TreeBalancing algorithms to perform basic 2:1 balanced construction of $4D$-trees. Once the basic topological tree is established we perform the computation of the unique shared nodes(see §3.4), and the scatter map computation to perform ghost/halo nodal value exchange. Note that the topological tree is independent of the FEM element order, where it is introduced at the unique node computation phase.

The main benchmark program used in the study is `tsortBench` which can be executed as mentioned in §7. The main sketch of the benchmark includes generating the random points in $\mathbb{R}^4$ space, using the normal distribution for the user specified grain size. These points are used to perform the adaptive tree construction and 2:1 balancing. Once the complete 2:1 balanced tree is constructed we unique node and communication map computation. The presented execution times are averaged of 10 different runs.

Once the randomly adaptive trees are generated and the corresponding unique nodes are computed we perform, 10 MATVEC operations, profiling each phase of the matvec. These phases include top-down traversal, elemental matvec operation, bottom up traversal and the communication cost.

8.2 Space-time adaptivity for linear & non-linear PDEs

PETSc was used to solve all the linear algebra problems. In particular, GMRES solver was used in conjunction with a BLOCK JACOBI preconditioner. Both the relative residual tolerance and the absolute residual tolerance are set to 10^{-12} in all numercal results. The GMRES breakdown tolerance is kept at the default value of 10^{-30}. The NEWTONLS class by PETSc, that implements a Newton Line Search method, was used for the nonlinear problems. Each of the iterations within NEWTONLS is solved by GMRES with BLOCK JACOBI preconditioner.

The correctness of each of the examples provided in the artifacts described in §7 are ensured by first solving the examples with a known analytical solution for increasingly refined meshes (controlled by `-refine_lvl` in the execution script). In each case, after the solution is computed, the error is then calculated using the known analytical solution. The convergence the linear algebra solution in each case and the convergence of the norm of the analytical error in the solution together guarantees the trustworthiness of the FEM results.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We perform scalability experiments in Stampede2 and Titan supercomputers for the parallel algorithms presented in the paper. We have included the instructions on how to run the code in the above-mentioned machines with details in artifact description in the paper.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

List of URLs and/or DOIs where artifacts are available:

`https://github.com/paralab/Dendro4`
`https://github.com/paralab/Dendro-KT`
`https://bitbucket.org/baskargroup/ad\_dendrite/src/master/`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: SKX, Stampede2, Titan Cray supercomputers

Operating systems and versions: Linux-3.10.0-957.5.1.el7.x86_64

Compilers and versions: intel/18.0.0 , impi/18.0.0

Libraries and versions: 1) intel/18.0.0 2) impi/18.0.0 3) cmake/3.10.2 4) git/2.9.0 5) gsl/2.3 6) autotools/1.1

Paper Modifications: The large scalability experiments reported in this paper were performed on Titan and Stampede2. Titan is a Cray XK7 supercomputer at Oak Ridge National Laboratory (ORNL) with a total of 18,688 nodes, each consisting of a single 16-core AMD Opteron 6200 series processor, with a total of 299,008 cores. Each node has 32GB of memory. It has a Gemini interconnect and 600TB of memory across all nodes. Stampede2 is the flagship supercomputer at the Texas Advanced Computing Center (TACC), the University of Texas at Austin. It has $1,736$ Intel Xeon Platinum 8160 (SKX) compute nodes with 48 cores and 192GB of RAM per node. Stampede2 has a 100Gb/sec Intel Omni-Path (OPA) interconnect in a fat tree topology. We used the SKX nodes for the experiments reported in this work.

Loaded Modules for Stampede2 experiements: 1) intel/18.0.0 2) impi/18.0.0 3) cmake/3.10.2 4) git/2.9.0 5) gsl/2.3 6) autotools/1.1

Output from scripts that gathers execution environment information.

```
SLURM_NODELIST=c476-014
LMOD_FAMILY_COMPILER_VERSION=18.0.0
SLURM_CHECKPOINT_IMAGE_DIR=/var/slurm/checkpoint
TACC_GIT_BIN=/opt/apps/git/2.9.0/bin
MKLROOT=/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/mkl
MANPATH=/opt/apps/git/2.9.0/share/man:/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/mpi/man:/home1/apps/intel/18.0.0/documentation_2018/en/man/common:/home1/apps/intel/18.0.0/documentation_2018/en/debugger/gdb-igfx/man:/home1/apps/intel/18.0.0/documentation_2018/en/debugger/gdb-ia/man
TACC_CMAKE_BIN=/opt/apps/cmake/3.10.2/bin
TACC_IMPI_DIR=/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/mpi
SLURM_JOB_NAME=idv17562
XDG_SESSION_ID=17096
HOSTNAME=c476-014
SLURMD_NODENAME=c476-014
SLURM_TOPOLOGY_ADDR=c476-014
_ModuleTable003_=cHRoIl09MCxbInN0YXR1cyJdPSJhY3RpdmU
iLFsidXNlck5hbWUiXT0iZ3NsIix9LGltcGk9e1siZm4iXT0
iL29wdC9hcHBzL2ludGVsMTgvbW9kdWxlZmlsZXMvaW1waS8
xOC4wLjAubHVhIixbImZ1bGxOYW1lIl09ImltcGkvMTguMC4
wIixbImxvYWRPcmRlciJdPTIscHJvcFQ9e30sWyJzdGFja0R
lcHRoIl09MCxbInN0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5
hbWUiXT0iaW1waS8xOC4wLjAiLH0saW50ZWw9e1siZm4iXT0
iL29wdC9hcHBzL21vZHVsZGVsZ2ludGVsLzE4LjAuMC5
sdWEiLFsiZnVsbE5hbWUiXT0iaW50ZWwvMTguMC4wIixbImx
vYWRPcmRlciJdPTEscHJvcFQ9e30sWyJzdGFja0RlcHRoIl0
9MCxbInN0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5hbWUi
TACC_INTEL_INC=/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/compiler/include/intel64
SLURM_PRIO_PROCESS=0
SLURM_NODE_ALIASES=(null)
I_MPI_F77=ifort
INTEL_LICENSE_FILE=/home1/03727/USER/intel/licenses:/home1/apps/intel/18.0.0/licenses:/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/licenses
IPPROOT=/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/ipp
MPICH_HOME=/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux/mpi
SHELL=/bin/bash
TERM=xterm-256color
TACC_INTEL_DIR=/home1/apps/intel/18.0.0/compilers_and_libraries_2018.0.128/linux
```

```
__LMOD_REF_COUNT_MODULEPATH=/opt/apps/bar1_1/modulef
↪ iles:1;/opt/apps/intel18/impi18_0/modulefiles:1;
↪ /opt/apps/intel18/modulefiles:1;/opt/apps/xsede/
↪ modulefiles:1;/opt/apps/modulefiles:1;/opt/modul
↪ efiles:1
NO_HOSTSORT=1
HISTSIZE=1000
IDEV_SETUP_BYPASS=1.0
I_MPI_FABRICS=shm:tmi
SLURM_JOB_QOS=normal
TACC_IMPI_BIN=/home1/apps/intel/18.0.0/compilers_and
↪ _libraries_2018.0.128/linux/mpi/intel64/bin
SSH_CLIENT=206.76.192.54 34734 22
LMOD_SYSTEM_DEFAULT_MODULES=TACC
TMPDIR=/tmp
SLURM_TOPOLOGY_ADDR_PATTERN=node
LIBRARY_PATH=/home1/apps/intel/18.0.0/compilers_and_
↪ libraries_2018.0.128/linux/daal/../tbb/lib/intel
↪ 64_lin/gcc4.4:/home1/apps/intel/18.0.0/compilers
↪ _and_libraries_2018.0.128/linux/daal/lib/intel64
↪ _lin:/home1/apps/intel/18.0.0/compilers_and_libr
↪ aries_2018.0.128/linux/tbb/lib/intel64/gcc4.7:/h
↪ ome1/apps/intel/18.0.0/compilers_and_libraries_2
↪ 018.0.128/linux/mkl/lib/intel64_lin:/home1/apps/
↪ intel/18.0.0/compilers_and_libraries_2018.0.128/
↪ linux/compiler/lib/intel64_lin:/home1/apps/intel
↪ /18.0.0/compilers_and_libraries_2018.0.128/linux
↪ /ipp/lib/intel64
LMOD_PKG=/opt/apps/lmod/lmod
TACC_FAMILY_COMPILER_VERSION=18.0.0
QTDIR=/usr/lib64/qt-3.3
QTINC=/usr/lib64/qt-3.3/include
LMOD_VERSION=7.8.21
SSH_TTY=/dev/pts/0
SLURM_TACC_RUNLIMIT_MINS=60
I_MPI_JOB_FAST_STARTUP=1
__LMOD_REF_COUNT_LOADEDMODULES=intel/18.0.0:1;impi/1
↪ 8.0.0:1;cmake/3.10.2:1;git/2.9.0:1;gsl/2.3:1;aut
↪ otools/1.1:1
QT_GRAPHICSSYSTEM_CHECKED=1
TACC_IMPI_INC=/home1/apps/intel/18.0.0/compilers_and
↪ _libraries_2018.0.128/linux/mpi/intel64/include
TACC_INTEL_BIN=/home1/apps/intel/18.0.0/compilers_an
↪ d_libraries_2018.0.128/linux/bin/intel64
USER=USER
SLURM_NNODES=1
IDEV_QDEL=scancel
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:p
↪ i=40;38;5;11:so=38;5;13:do=38;5;5:bd=48;5;232;38
↪ ;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=0
↪ 5;48;5;232;38;5;15:su=48;5;196;38;5;15:sg=48;5;1
↪ 1;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38;5;1
↪ 6:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;
↪ 34:*.tar=38;5;9:*.tgz=38;5;9:*.arc=38;5;9:*.arj=
↪ 38;5;9:*.taz=38;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.
↪ lzh=38;5;9:*.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5
↪ ;9:*.tzo=38;5;9:*.t7z=38;5;9:*.zip=38;5;9:*.z=38
↪ ;5;9:*.Z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38
↪ ;5;9:*.lz=38;5;9:*.lzo=38;5;9:*.xz=38;5;9:*.bz2=
↪ 38;5;9:*.bz=38;5;9:*.tbz=38;5;9:*.tbz2=38;5;9:*.
↪ tz=38;5;9:*.deb=38;5;9:*.rpm=38;5;9:*.jar=38;5;9
↪ :*.war=38;5;9:*.ear=38;5;9:*.sar=38;5;9:*.rar=38
↪ ;5;9:*.alz=38;5;9:*.ace=38;5;9:*.zoo=38;5;9:*.cp
↪ io=38;5;9:*.7z=38;5;9:*.rz=38;5;9:*.cab=38;5;9:*
↪ .jpg=38;5;13:*.jpeg=38;5;13:*.gif=38;5;13:*.bmp=
↪ 38;5;13:*.pbm=38;5;13:*.pgm=38;5;13:*.ppm=38;5;1
↪ 3:*.tga=38;5;13:*.xbm=38;5;13:*.xpm=38;5;13:*.ti
↪ f=38;5;13:*.tiff=38;5;13:*.png=38;5;13:*.svg=38;
↪ 5;13:*.svgz=38;5;13:*.mng=38;5;13:*.pcx=38;5;13:
↪ *.mov=38;5;13:*.mpg=38;5;13:*.mpeg=38;5;13:*.m2v
↪ =38;5;13:*.mkv=38;5;13:*.webm=38;5;13:*.ogm=38;5
↪ ;13:*.mp4=38;5;13:*.m4v=38;5;13:*.mp4v=38;5;13:*
↪ .vob=38;5;13:*.qt=38;5;13:*.nuv=38;5;13:*.wmv=38
↪ ;5;13:*.asf=38;5;13:*.rm=38;5;13:*.rmvb=38;5;13:
↪ *.flc=38;5;13:*.avi=38;5;13:*.fli=38;5;13:*.flv=
↪ 38;5;13:*.gl=38;5;13:*.dl=38;5;13:*.xcf=38;5;13:
↪ *.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=
↪ 38;5;13:*.axv=38;5;13:*.anx=38;5;13:*.ogv=38;5;1
↪ 3:*.ogx=38;5;13:*.aac=38;5;45:*.au=38;5;45:*.fla
↪ c=38;5;45:*.mid=38;5;45:*.midi=38;5;45:*.mka=38;
↪ 5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg=38;5;45:*
↪ .ra=38;5;45:*.wav=38;5;45:*.axa=38;5;45:*.oga=38
↪ ;5;45:*.spx=38;5;45:*.xspf=38;5;45:
LD_LIBRARY_PATH=/opt/apps/intel18/gsl/2.3/lib:/home1
↪ /apps/intel/18.0.0/compilers_and_libraries_2018.
↪ 0.128/linux/mpi/intel64/lib:/home1/apps/intel/18
↪ .0.0/debugger_2018/libipt/intel64/lib:/home1/app
↪ s/intel/18.0.0/debugger_2018/iga/lib:/home1/apps
↪ /intel/18.0.0/compilers_and_libraries_2018.0.128
↪ /linux/daal/../tbb/lib/intel64_lin/gcc4.4:/home1
↪ /apps/intel/18.0.0/compilers_and_libraries_2018.
↪ 0.128/linux/daal/lib/intel64_lin:/home1/apps/int
↪ el/18.0.0/compilers_and_libraries_2018.0.128/lin
↪ ux/tbb/lib/intel64/gcc4.7:/home1/apps/intel/18.0
↪ .0/compilers_and_libraries_2018.0.128/linux/mkl/
↪ lib/intel64_lin:/home1/apps/intel/18.0.0/compile
↪ rs_and_libraries_2018.0.128/linux/compiler/lib/i
↪ ntel64_lin:/home1/apps/intel/18.0.0/compilers_an
↪ d_libraries_2018.0.128/linux/ipp/lib/intel64:/ho
↪ me1/apps/intel/18.0.0/compilers_and_libraries_20
↪ 18.0.128/linux/compiler/lib/intel64:/opt/apps/gc
↪ c/6.3.0/lib64:/opt/apps/gcc/6.3.0/lib
__TRACKER__=1
```

```
TACC_FAMILY_GSL_VERSION=2.3
TACC_NODE_TYPE=skx
SLURM_TACC_NODES=1
SLURM_JOBID=3272468
_ModuleTable004_=XT0iaW50ZWwvMTguMC4wIix9LH0sbXBhdGh
↪ BPXsiL29wdC9hcHBzL2JhcjFfMS9tb2R1bGVmaWxlcyIsIi9
↪ vcHQvYXBwcy9pbnRlbDE4L2ltcGkxOF8wL21vZHVsZWZpbGV
↪ zIiwiL29wdC9hcHBzL2ludGVsMTgvbW9kdWxlZmlsZXMiLCI
↪ vb3B0L2FwcHMveHNlZGUvbW9kdWxlZmlsZXMiLCIvb3B0L2F
↪ wcHMvbW9kdWxlZmlsZXMiLCIvb3B0L21vZHVsZWZpbGVzIix
↪ 9LFsic3lzdGVtQmFzZU1QQVRIIl09Ii9vcHQvYXBwcy94c2V
↪ kZS9tb2R1bGVmaWxlczovb3B0L2FwcHMvbW9kdWxlZmlsZXM
↪ 6L29wdC9tb2R1bGVmaWxlcyIsfQ==
CPATH=/home1/apps/intel/18.0.0/compilers_and_librari
↪ es_2018.0.128/linux/daal/include:/home1/apps/int
↪ el/18.0.0/compilers_and_libraries_2018.0.128/lin
↪ ux/tbb/include:/home1/apps/intel/18.0.0/compiler
↪ s_and_libraries_2018.0.128/linux/mkl/include:/ho
↪ me1/apps/intel/18.0.0/compilers_and_libraries_20
↪ 18.0.128/linux/ipp/include
IFC_BIN=/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/bin/intel64
TACC_MKL_LIB=/home1/apps/intel/18.0.0/compilers_and_
↪ libraries_2018.0.128/linux/mkl/lib/intel64
TACC_GIT_DIR=/opt/apps/git/2.9.0
__PERSONAL_PATH__=1
__LMOD_REF_COUNT__LMFILES_=/opt/apps/modulefiles/int
↪ el/18.0.0.lua:1;/opt/apps/intel18/modulefiles/im
↪ pi/18.0.0.lua:1;/opt/apps/modulefiles/cmake/3.10
↪ .2.lua:1;/opt/apps/modulefiles/git/2.9.0.lua:1;/
↪ opt/apps/intel18/modulefiles/gsl/2.3.lua:1;/opt/
↪ apps/modulefiles/autotools/1.1.lua:1
OLDSCRATCH=/oldscratch/03727/USER
SLURM_COMMAND=sbatch
TACC_GSL_LIB=/opt/apps/intel18/gsl/2.3/lib
SLURM_NTASKS=48
SLURM_TACC_JOBNAME=idv17562
LMOD_FAMILY_MPI_VERSION=18.0.0
ARCHIVER=ranch.tacc.utexas.edu
NLSPATH=/home1/apps/intel/18.0.0/debugger_2018/gdb/i
↪ ntel64/share/locale/%l_%t/%N:/home1/apps/intel/1
↪ 8.0.0/compilers_and_libraries_2018.0.128/linux/m
↪ kl/lib/intel64/locale/%l_%t/%N:/home1/apps/intel
↪ /18.0.0/compilers_and_libraries_2018.0.128/linux
↪ /compiler/lib/intel64/locale/%l_%t/%N
__LMOD_REF_COUNT_I_MPI_ROOT=/home1/apps/intel/18.0.0
↪ /compilers_and_libraries_2018.0.128/linux/mpi:1
PATH=/opt/apps/autotools/1.1/bin:/opt/apps/intel18/g
↪ sl/2.3/bin:/opt/apps/git/2.9.0/bin:/opt/apps/cma
↪ ke/3.10.2/bin:/opt/apps/intel18/impi/18.0.0/bin:
↪ /home1/apps/intel/18.0.0/compilers_and_libraries
↪ _2018.0.128/linux/mpi/intel64/bin:/home1/apps/in
↪ tel/18.0.0/compilers_and_libraries_2018.0.128/li
↪ nux/bin/intel64:/opt/apps/gcc/6.3.0/bin:/usr/lib
↪ 64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/opt/
↪ dell/srvadmin/bin
MAIL=/var/spool/mail/USER
TACC_SYSTEM=stampede2
_ModuleTable001_=X01vZHVsZVRhYmxlXz17WyJNVHZlcnNpb24
↪ iXT0zLFsiY19yZWJ1aWxkVGltZSJdPWZhbHNlLFsiY19zaG9
↪ ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9e30sZmFtaWx5PXtbIkd
↪ TTCJdPSJnc2wiLFsiTVBJIl09ImltcGkiLFsiY29tcGlsZXI
↪ iXT0iaW50ZWwiLH0sbVQ9e2F1dG90b29scz17WyJmbiJdPSI
↪ vb3B0L2FwcHMvbW9kdWxlZmlsZXMvYXV0b3Rvb2xzLzEuMS5
↪ sdWEiLFsiZnVsbE5hbWUiXT0iYXV0b3Rvb2xzLzEuMSIsWyJ
↪ sb2FkT3JkZXIiXT02LHByb3BUPXt9LFsic3RhY2tEZXB0aCJ
↪ dPTAsWyJzdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl0
↪ 9ImF1dG90b29scyIsfSxjbWFrZT17WyJmbiJdPSIvb3B0L2F
↪ wcHMvbW9kdWxlZmlsZXMvY21ha2UvMy4xMC4yLmx1YSIs
__LMOD_REF_COUNT_NLSPATH=/home1/apps/intel/18.0.0/de
↪ bugger_2018/gdb/intel64/share/locale/%l_%t/%N:1;
↪ /home1/apps/intel/18.0.0/compilers_and_libraries
↪ _2018.0.128/linux/mkl/lib/intel64/locale/%l_%t/%
↪ N:1;/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/compiler/lib/intel64/local
↪ e/%l_%t/%N:1
SLURM_TASKS_PER_NODE=48
ICC_BIN=/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/bin/intel64
STOCKYARD=/work/03727/USER
RUNNING_IDEV=1
SLURM_WORKING_CLUSTER=stampede2:206.76.192.2:6820:84
↪ 48
TACC_GIT_LIB=/opt/apps/git/2.9.0/lib
TACC_GSL_BIN=/opt/apps/intel18/gsl/2.3/bin
_=/bin/env
WORK=/work/03727/USER/stampede2
TACCINFO=/usr/local/etc/taccinfo
SLURM_JOB_ID=3272468
OLDWORK=/work/03727/USER
TBBROOT=/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/tbb
PWD=/home1/03727/USER/Research/Author-Kit
INPUTRC=/etc/inputrc
SLURM_JOB_USER=USER
TACC_GSL_INC=/opt/apps/intel18/gsl/2.3/include
SLURM_QUEUE=skx-normal
SLURM_TACC_NCORES_SET=1
_LMFILES_=/opt/apps/modulefiles/intel/18.0.0.lua:/op
↪ t/apps/intel18/modulefiles/impi/18.0.0.lua:/opt/
↪ apps/modulefiles/cmake/3.10.2.lua:/opt/apps/modu
↪ lefiles/git/2.9.0.lua:/opt/apps/intel18/modulefi
↪ les/gsl/2.3.lua:/opt/apps/modulefiles/autotools/
↪ 1.1.lua
LANG=en_US.UTF-8
MODULEPATH=/opt/apps/bar1_1/modulefiles:/opt/apps/in
↪ tel18/impi18_0/modulefiles:/opt/apps/intel18/mod
↪ ulefiles:/opt/apps/xsede/modulefiles:/opt/apps/m
↪ odulefiles:/opt/modulefiles
_ModuleTable_Sz_=4
LOADEDMODULES=intel/18.0.0:impi/18.0.0:cmake/3.10.2:
↪ git/2.9.0:gsl/2.3:autotools/1.1
```

```
SLURM_JOB_UID=830270
SLURM_NODEID=0
idev_has_user_PERL5LIB=no
__BASHRC_SOURCED__=1
SLURM_TACC_ACCOUNT=TG-DPP130002
SLURM_SUBMIT_DIR=/home1/03727/USER/Research/Author-K
↪ it
LMOD_FAMILY_GSL=gsl
TACC_VEC_FLAGS=-xCORE-AVX2 -axCORE-AVX512,MIC-AVX512
I_MPI_F90=ifort
LMOD_CMD=/opt/apps/lmod/lmod/libexec/lmod
SLURM_TASK_PID=352478
SLURM_NPROCS=48
GIT_TEMPLATE_DIR=/opt/apps/git/2.9.0/share/git-core/
↪ templates
I_MPI_CC=icc
DAALROOT=/home1/apps/intel/18.0.0/compilers_and_libr
↪ aries_2018.0.128/linux/daal
SLURM_CPUS_ON_NODE=96
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
TACC_MKL_INC=/home1/apps/intel/18.0.0/compilers_and_
↪ libraries_2018.0.128/linux/mkl/include
SLURM_PROCID=0
ENVIRONMENT=BATCH
TACC_AUTOTOOLS_DIR=/opt/apps/autotools/1.1
SLURM_JOB_NODELIST=c476-014
HOME=/home1/03727/USER
SHLVL=4
I_MPI_HYDRA_PMI_CONNECT=alltoall
LMOD_FAMILY_GSL_VERSION=2.3
TACC_DOMAIN=stampede2
SLURM_LOCALID=0
TACC_FAMILY_COMPILER=intel
I_MPI_CXX=icpc
__LMOD_REF_COUNT_PATH=/opt/apps/autotools/1.1/bin:1;
↪ /opt/apps/intel18/gsl/2.3/bin:1;/opt/apps/git/2.
↪ 9.0/bin:1;/opt/apps/cmake/3.10.2/bin:1;/opt/apps
↪ /intel18/impi/18.0.0/bin:1;/home1/apps/intel/18.
↪ 0.0/compilers_and_libraries_2018.0.128/linux/mpi
↪ /intel64/bin:1;/home1/apps/intel/18.0.0/compiler
↪ s_and_libraries_2018.0.128/linux/bin/intel64:1;/
↪ opt/apps/gcc/6.3.0/bin:1;/usr/lib64/qt-3.3/bin:1
↪ ;/usr/local/bin:1;/bin:1;/usr/bin:1;/opt/dell/sr
↪ vadmin/bin:1
TACC_INTEL_LIB=/home1/apps/intel/18.0.0/compilers_an
↪ d_libraries_2018.0.128/linux/compiler/lib/intel64
__LMOD_REF_COUNT_CPATH=/home1/apps/intel/18.0.0/comp
↪ ilers_and_libraries_2018.0.128/linux/daal/includ
↪ e:1;/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/tbb/include:1;/home1/apps/
↪ intel/18.0.0/compilers_and_libraries_2018.0.128/
↪ linux/mkl/include:1;/home1/apps/intel/18.0.0/com
↪ pilers_and_libraries_2018.0.128/linux/ipp/includ
↪ e:1
IFC_LIB=/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/compiler/lib/intel64
_ModuleTable002_=WyJmdWxsTmFtZSJdPSJjbWFrZS8zLjEwLjI
↪ iLFsibG9hZE9yZGVyIl09Myxwcm9wVD17fSxbInN0YWNrRGV
↪ wdGgiXT0wLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmF
↪ tZSJdPSJjbWFrZSIsfSxnaXQ9e1siZm4iXT0iL29wdC9hcHB
↪ zL21vZHVsZWZpbGVzL2dpdC8yLjkuMC5sdWEiLFsiZnVsbE5
↪ hbWUiXT0iZ2l0LzIuOS4wIixbImxvYWRPcmRlciJdPTQscHJ
↪ vcFQ9e30sWyJzdGFja0RlcHRoIl09MCxbInN0YXR1cyJdPSJ
↪ hY3RpdmUiLFsidXNlck5hbWUiXT0iZ2l0Iix9LGdzbD17WyJ
↪ mbiJdPSIvb3B0L2FwcHMvaW50ZWwxOC9tb2R1bGVmaWxlcy9
↪ nc2wvMi4zLmx1YSIsWyJmdWxsTmFtZSJdPSJnc2wvMi4zIix
↪ bImxvYWRPcmRlciJdPTUscHJvcFQ9e30sWyJzdGFja0Rl
SLURM_CLUSTER_NAME=stampede2
SLURM_JOB_CPUS_PER_NODE=96
SLURM_JOB_GID=816328
TACC_GSL_DIR=/opt/apps/intel18/gsl/2.3
SLURM_GTIDS=0
SLURM_SUBMIT_HOST=login4.stampede2.tacc.utexas.edu
BASH_ENV=/etc/tacc/tacc_functions
idev_ip=c476-014
I_MPI_FC=ifort
SLURM_JOB_PARTITION=skx-normal
LOGNAME=USER
ICC_LIB=/home1/apps/intel/18.0.0/compilers_and_libra
↪ ries_2018.0.128/linux/compiler/lib/intel64
TACC_FAMILY_MPI=impi
CVS_RSH=ssh
QTLIB=/usr/lib64/qt-3.3/lib
LMOD_SETTARG_TITLE_BAR=yes
SSH_CONNECTION=206.76.192.54 34734 206.76.210.144 22
XDG_DATA_DIRS=/home1/03727/USER/.local/share/flatpak
↪ /exports/share:/var/lib/flatpak/exports/share:/u
↪ sr/local/share:/usr/share
TACC_FAMILY_GSL=gsl
SLURM_JOB_ACCOUNT=TG-DPP130002
MODULESHOME=/opt/apps/lmod/lmod
SLURM_JOB_NUM_NODES=1
__LMOD_REF_COUNT_LIBRARY_PATH=/home1/apps/intel/18.0
↪ .0/compilers_and_libraries_2018.0.128/linux/daal
↪ /../tbb/lib/intel64_lin/gcc4.4:1;/home1/apps/int
↪ el/18.0.0/compilers_and_libraries_2018.0.128/lin
↪ ux/daal/lib/intel64_lin:1;/home1/apps/intel/18.0
↪ .0/compilers_and_libraries_2018.0.128/linux/tbb/
↪ lib/intel64/gcc4.7:1;/home1/apps/intel/18.0.0/co
↪ mpilers_and_libraries_2018.0.128/linux/mkl/lib/i
↪ ntel64_lin:1;/home1/apps/intel/18.0.0/compilers_
↪ and_libraries_2018.0.128/linux/compiler/lib/inte
↪ l64_lin:1;/home1/apps/intel/18.0.0/compilers_and
↪ _libraries_2018.0.128/linux/ipp/lib/intel64:1
LESSOPEN=||/usr/bin/lesspipe.sh %s
LMOD_SETTARG_FULL_SUPPORT=full
```

```
__LMOD_REF_COUNT_LD_LIBRARY_PATH=/opt/apps/intel18/g
↪ sl/2.3/lib:1;/home1/apps/intel/18.0.0/compilers_
↪ and_libraries_2018.0.128/linux/mpi/intel64/lib:1
↪ ;/home1/apps/intel/18.0.0/debugger_2018/libipt/i
↪ ntel64/lib:1;/home1/apps/intel/18.0.0/debugger_2
↪ 018/iga/lib:1;/home1/apps/intel/18.0.0/compilers
↪ _and_libraries_2018.0.128/linux/daal/../tbb/lib/
↪ intel64_lin/gcc4.4:1;/home1/apps/intel/18.0.0/co
↪ mpilers_and_libraries_2018.0.128/linux/daal/lib/
↪ intel64_lin:1;/home1/apps/intel/18.0.0/compilers
↪ _and_libraries_2018.0.128/linux/tbb/lib/intel64/
↪ gcc4.7:1;/home1/apps/intel/18.0.0/compilers_and_
↪ libraries_2018.0.128/linux/mkl/lib/intel64_lin:1
↪ ;/home1/apps/intel/18.0.0/compilers_and_librarie
↪ s_2018.0.128/linux/compiler/lib/intel64_lin:2;/h
↪ ome1/apps/intel/18.0.0/compilers_and_libraries_2
↪ 018.0.128/linux/ipp/lib/intel64:1;/home1/apps/in
↪ tel/18.0.0/compilers_and_libraries_2018.0.128/li
↪ nux/compiler/lib/intel64:1;/opt/apps/gcc/6.3.0/l
↪ ib64:1;/opt/apps/gcc/6.3.0/lib:1
PKG_CONFIG_PATH=/opt/apps/intel18/gsl/2.3/lib/pkgcon
↪ fig
OMP_NUM_THREADS=1
PROMPT_COMMAND=${X_SET_TITLE_BAR:-:}
↪ "$USER@${SHOST}:${PWD/#$HOME/~}"
__Init_Default_Modules=1
LMOD_FAMILY_COMPILER=intel
TACC_IMPI_LIB=/home1/apps/intel/18.0.0/compilers_and
↪ _libraries_2018.0.128/linux/mpi/intel64/lib
XDG_RUNTIME_DIR=/run/user/830270
ARCHIVE=/home/03727/USER
TACC_FAMILY_MPI_VERSION=18.0.0
__LMOD_REF_COUNT_INTEL_LICENSE_FILE=/home1/03727/USE
↪ R/intel/licenses:1;/home1/apps/intel/18.0.0/lice
↪ nses:1;/home1/apps/intel/18.0.0/compilers_and_li
↪ braries_2018.0.128/linux/licenses:1
TACC_AUTOTOOLS_BIN=/opt/apps/autotools/1.1/bin
OLDHOME=/oldhome1/03727/USER
IDEV_PWD=/home1/03727/USER/Research/Author-Kit
LMOD_DIR=/opt/apps/lmod/lmod/libexec
__LMOD_REF_COUNT_MANPATH=/opt/apps/git/2.9.0/share/m
↪ an:1;/home1/apps/intel/18.0.0/compilers_and_libr
↪ aries_2018.0.128/linux/mpi/man:1;/home1/apps/int
↪ el/18.0.0/documentation_2018/en/man/common:1;/ho
↪ me1/apps/intel/18.0.0/documentation_2018/en/debu
↪ gger/gdb-igfx/man:1;/home1/apps/intel/18.0.0/doc
↪ umentation_2018/en/debugger/gdb-ia/man:1
I_MPI_TMI_PROVIDER=psm2
SCRATCH=/scratch/03727/USER
GIT_EXEC_PATH=/opt/apps/git/2.9.0/libexec/git-core
TACC_MPI_GETMODE=impi_hydra
SLURM_TACC_NNODES_SET=1
TACC_MKL_DIR=/home1/apps/intel/18.0.0/compilers_and_
↪ libraries_2018.0.128/linux/mkl
LMOD_FAMILY_MPI=impi
SLURM_TACC_CORES=48
TACC_CMAKE_DIR=/opt/apps/cmake/3.10.2
I_MPI_ROOT=/home1/apps/intel/18.0.0/compilers_and_li
↪ braries_2018.0.128/linux/mpi
BASH_FUNC_sbatch()=() {  echo -e "\nNOTIFICATION:
↪ sbatch not available on compute nodes. Use a login
↪ node.\n"
}
BASH_FUNC_module()=() {  if [ -z
↪ "${LMOD_SH_DBG_ON+x}" ]; then
 case "$-" in
 *v*x*)
 __lmod_sh_dbg='vx'
 ;;
 *v*)
 __lmod_sh_dbg='v'
 ;;
 *x*)
 __lmod_sh_dbg='x'
 ;;
 esac;
 fi;
 if [ -n "${__lmod_sh_dbg:-}" ]; then
 set +$__lmod_sh_dbg;
 echo "Shell debugging temporarily silenced: export
 ↪ LMOD_SH_DBG_ON=1 for Lmod's output";
 fi;
 eval $($LMOD_CMD bash "$@") && eval
 ↪ $(${LMOD_SETTARG_CMD:-:} -s sh);
 local _lmod_my_status=$?;
 if [ -n "${__lmod_sh_dbg:-}" ]; then
 echo "Shell debugging restarted";
 set -$__lmod_sh_dbg;
 unset __lmod_sh_dbg;
 fi;
 return $_lmod_my_status
}
BASH_FUNC_ml()=() {  eval $($LMOD_DIR/ml_cmd "$@")
}
+ lsb_release -a
LSB Version:        :core-4.1-amd64:core-4.1-noarch:
↪ cxx-4.1-amd64:cxx-4.1-noarch:desktop-4.1-amd64:d
↪ esktop-4.1-noarch:languages-4.1-amd64:languages-
↪ 4.1-noarch:printing-4.1-amd64:printing-4.1-noarch
Distributor ID:        CentOS
Description:        CentOS Linux release 7.6.1810
↪ (Core)
Release:        7.6.1810
Codename:        Core
+ uname -a
Linux c476-014.stampede2.tacc.utexas.edu
↪ 3.10.0-957.5.1.el7.x86_64 #1 SMP Fri Feb 1
↪ 14:54:57 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
+ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
```

```
CPU(s):                96
On-line CPU(s) list:   0-95
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Platinum 8160
↪  CPU @ 2.10GHz
Stepping:              4
CPU MHz:               2100.000
BogoMIPS:              4200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              33792K
NUMA node0 CPU(s):
↪  0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34┘
↪  ,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66┘
↪  ,68,70,72,74,76,78,80,82,84,86,88,90,92,94
NUMA node1 CPU(s):
↪  1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35┘
↪  ,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67┘
↪  ,69,71,73,75,77,79,81,83,85,87,89,91,93,95
Flags:                 fpu vme de pse tsc msr pae mce
↪  cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪  dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪  pdpe1gb rdtscp lm constant_tsc art arch_perfmon
↪  pebs bts rep_good nopl xtopology nonstop_tsc
↪  aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
↪  ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
↪  pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt
↪  tsc_deadline_timer aes xsave avx f16c rdrand
↪  lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3
↪  intel_ppin intel_pt ssbd mba ibrs ibpb stibp
↪  tpr_shadow vnmi flexpriority ept vpid fsgsbase
↪  tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid
↪  rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap
↪  clflushopt clwb avx512cd avx512bw avx512vl
↪  xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc
↪  cqm_mbm_total cqm_mbm_local dtherm ida arat pln
↪  pts pku ospke spec_ctrl intel_stibp flush_l1d
+ cat /proc/meminfo
MemTotal:       196438176 kB
MemFree:        191589692 kB
MemAvailable:   190917852 kB
Buffers:               0 kB
Cached:            62136 kB
SwapCached:            0 kB
Active:           112696 kB
Inactive:          39060 kB
Active(anon):      89860 kB
Inactive(anon):    10816 kB
Active(file):      22836 kB
Inactive(file):    28244 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                24 kB
Writeback:             0 kB
AnonPages:         88792 kB
Mapped:            34516 kB
Shmem:             10892 kB
Slab:            1596960 kB
SReclaimable:     172696 kB
SUnreclaim:      1424264 kB
KernelStack:       23440 kB
PageTables:         6336 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    182687500 kB
Committed_AS:     475580 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      1826376 kB
VmallocChunk:   34256930848 kB
HardwareCorrupted:     0 kB
AnonHugePages:     16384 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      646976 kB
DirectMap2M:     8402944 kB
DirectMap1G:    192937984 kB
+ inxi -F -c0
./collect_environment.sh: line 14: inxi: command not
↪  found
+ lsblk -a
NAME              MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                 8:0    0 223.6G  0 disk
├─sda1              8:1    0     1M  0 part
├─sda2              8:2    0     1G  0 part /boot
└─sda3              8:3    0 222.6G  0 part
  ├─rootvg01-lv01 253:0    0    75G  0 lvm  /
  ├─rootvg01-tmp  253:1    0 143.6G  0 lvm  /tmp
  └─rootvg01-var  253:2    0     4G  0 lvm  /var
sdb                 8:16   0         0 disk
+ lsscsi -s
[0:0:0:0]    disk    Generic  MassStorageClass WS01
↪  /dev/sdb        -
[3:0:0:0]    disk    ATA      MZ7KM240HMHQ0D3  GD53
↪  /dev/sda    240GB
+ module list
```

```
+ '[' -z '' ']'
+ case "$-" in
+ __lmod_sh_dbg=x
+ '[' -n x ']'
+ set +x
Shell debugging temporarily silenced: export
↪  LMOD_SH_DBG_ON=1 for Lmod's output

Currently Loaded Modules:
  1) intel/18.0.0   2) impi/18.0.0   3) cmake/3.10.2
  ↪  4) git/2.9.0   5) gsl/2.3   6) autotools/1.1



Shell debugging restarted
+ unset __lmod_sh_dbg
+ return 0
+ nvidia-smi
./collect_environment.sh: line 18: nvidia-smi:
↪  command not found
+ lshw -short -quiet -sanitize
+ cat
./collect_environment.sh: line 19: lshw: command not
↪  found
+ lspci
./collect_environment.sh: line 19: lspci: command not
↪  found
```

ARTIFACT EVALUATION

Verification and validation studies: Compared the space-time numerical solutions with analytical solutions for simple linear addiction-diffusion equation.

Accuracy and precision of timings: All the timings are reported, scalability on 4d tree construction, adaptive matrix-vector multiplications are based on average of 10 executions.