A Task Parallel Algorithm For Fast Hybridized PDE Solvers

Joseph McLaughlin[1 [0009−0008−4114−5202]], Brittany Erickson[1 [0000−0001−9457−8572]], and Jee Choi[1 [0000−0002−6938−8221]]

University of Oregon, Eugene, Oregon, USA

Abstract. Earthquake-cycle simulations require solving a large sparse elliptic system at every timestep, making repeated linear solves the dominant cost. We present a task-parallel hybridized finite-difference solver for multi-core CPUs based on SBP-SAT discretizations. SBP-SAT hybridization preserves sparse stencil operators within each subproblem, making subproblem size a tunable parameter. We exploit this to develop a dependency-aware task-parallel algorithm that overlaps subproblem solves with global trace assembly, reducing bulk-synchronous barriers and improving cache reuse, and a Roofline-based performance model that selects the subproblem size minimizing end-to-end solve time. On 64-core Sapphire Rapids and 104-core Ice Lake systems, our solver achieves up to 9.1× speedup over a baseline hybridized solver, up to 4.7× over IC(0)-preconditioned CG, and up to 12× over sparse multifrontal QR. Model-guided sizing achieves up to 31× speedup over commonly used fixed sizes, substantially reducing per-solve costs for earthquake-cycle simulations.

Keywords: Hybridized methods, · task-parallel solvers · multi-core CPUs · earthquake-cycle simulations · PDEs.

1 Introduction

A primary challenge when solving partial differential equations (PDEs) is their reliance on low arithmetic intensity operations, where memory accesses dominate compute, leaving processing units idle. Techniques such as cache-obvious algorithms [2], sparse matrix formats [1, 6], specialized preconditioners [22, 5], address this constraint at different levels of the solver stack. In contrast, hybridization is a technique that reduces the degrees-of-freedom (DOFs) required to represent the underlying system of linear equations when solving elliptic PDEs [11]. Hybridization decomposes the problem into independent *subproblems* connected at interfaces, which define a smaller *global trace system*. We exploit two key properties of hybridization—data-independent subproblems and reduced memory footprint—that directly address the memory and parallelism constraints of modern architectures to develop a new parallel algorithm for solving PDEs on multi-core CPUs.

Our approach relies on a key distinction between hybridized finite differences (FD) and finite element methods (FEM). In hybridized FEM, such as

hybridizable discontinuous Galerkin (HDG)[15], subproblem interiors are typically represented by dense local matrices, so local factorization and solve costs increase rapidly with subproblem size. Consequently, large subproblems are often unattractive in practice. In contrast, our hybridized SBP-SAT finite-difference formulation preserves sparse, stencil-structured operators within each subproblem. This makes the subproblem size a flexible tuning parameter, allowing it to grow without incurring the dense-local algebra penalty and enabling a principled trade-off between local work and the cost of the global trace solve.

We target earthquake-cycle simulations, which model multi-scale deformation of the Earth over hundreds of years [8]. These simulations require implicit time integration, solving a large sparse system at every timestep, often requiring hundreds of thousands of solves per simulation. In solve-dominant workloads, even modest per-solve speedups translate directly to reduced wall-clock time.

1.1 Contributions

We make three key contributions towards improving the performance of hybridized PDE solvers.

1. We develop a task-based parallel algorithm that captures data dependencies across subproblems and global problem assembly, reducing bulk-synchronous barriers and improving cache reuse (§2.3). On 64-core Sapphire Rapids and 104-core Ice Lake systems, this achieves up to 9.1× speedup over a baseline parallel hybridized solver (§4.2).
2. We develop a Roofline-derived model that predicts the optimal subproblem size by balancing local solve costs against global trace solve costs (§3). Our model selects the optimal parameter in 87.7% of cases, with average performance within 1.7% of an exhaustive search (§4.3).
3. We compare against widely-deployed *non-hybridized* commercial solvers, IC(0)-preconditioned CG and multi-frontal QR from Intel MKL, achieving up to 4.7× and 12× speedup, respectively (§4.6).

1.2 Related work

The existing literature on hybridization for solving PDEs largely focuses on its use as a memory reduction technique, leaving its potential as a performance optimization strategy largely underexplored. Work prioritizing memory reduction increases subproblem density (i.e., number of subproblems), making each subproblem as small as possible [9, 13, 3]. This grows the size of the global trace system, motivating preconditioner research [20, 7] for faster convergence. Low-density approaches [10] use larger subproblems to shrink the trace system but provide no guidance on optimal sizing. One study [14] compared three densities and observed that performance degrades at extremes, but did not provide guidance on selecting the optimal density. Our work explicitly models this trade-off.

Hybridization is distinct from algebraic domain decomposition and bordered block-diagonal (BBD) methods. In BBD formulations, the global matrix is partitioned after discretization, producing a border that couples subproblems; this

border is typically dense and partition-dependent.Classical substructuring methods similarly operate algebraically and often solve interface coupling implicitly. In contrast, hybridization introduces trace unknowns during discretization, ensuring the global system is sparse and structured by the mesh layout. This explicit sparse structure is what enables our task-parallel algorithm and performance model. Importantly, algebraic techniques (*e.g.*, domain decomposition preconditioners) can be applied to the subproblems and trace system produced by hybridization, making it a distinct stage in the solver workflow.

Task parallelism has been applied throughout PDE solver workflows, including factorization [25, 4] and preconditioning [26]. However, these efforts target solver and assembly phases, not the numerical method itself. Hybridized methods naturally decompose into independent subproblems coupled through a sparse trace system, making them well-suited to task-based execution. Yet, existing implementations largely rely on bulk-synchronous models. We show that task-based execution improves cache reuse and reduces synchronization overhead.

2 Parallel Hybridized PDE Solver

We solve a version of the 2D anti-plane shear deformation problem [18] with *variable* coefficients:

$$-\left(\frac{\partial}{\partial x}+\frac{\partial}{\partial y}\right)\cdot\left(\mu(x,y)\left(\frac{\partial}{\partial x}+\frac{\partial}{\partial y}\right)u(x,y)\right)=f(x,y) \tag{1a}$$

on the unite square $0 \leq x, y \leq 1$ with mixed Dirichlet/Neumann boundary conditions. This problem arises in earthquake-cycle simulations, where implicit time integration requires solving a linear system at every timestep. Because matrix factors are reused across timesteps, the dominant cost is the solve phase.

2.1 Hybridized scheme

We discretize using the summation-by-parts finite difference method with simultaneous approximation terms (SBP-SAT) [17, 19, 24], which provides high-order accuracy and provable energy stability within stenciled kernel [16]. Hybridization decomposes the domain into sub-domains coupled at interfaces (Fig. 1a-b),

$$\begin{bmatrix}\mathbf{M} & \mathbf{F}\\ \mathbf{F}^{\intercal} & \mathbf{D}\end{bmatrix}\begin{pmatrix}\mathbf{u}\\ \boldsymbol{\lambda}\end{pmatrix}=\begin{pmatrix}\bar{\mathbf{g}}\\ \bar{\mathbf{g}}_{\delta}\end{pmatrix}. \tag{2}$$

Here, $\mathbf{M}$ is block-diagonal with each block encoding a subproblem, $\mathbf{F}$ encodes interface condtions, $\mathbf{D}$ is a sparse diagonal matrix, and $\boldsymbol{\lambda}$ contains trace variables on internal interfaces. Vectors $\bar{\mathbf{g}}$ and $\bar{\mathbf{g}}_{\delta}$ contain the source data for the external boundaries and internal interfaces. The solution is obtained using the Schur complement,

$$\boldsymbol{\Lambda}_{\mathbf{A}}=\mathbf{D}-\mathbf{F}^{\intercal}\mathbf{M}^{-1}\mathbf{F}, \tag{3a}$$

$$\boldsymbol{\lambda}_{\mathbf{b}}=\bar{\mathbf{g}}_{\delta}-\mathbf{F}^{\intercal}\mathbf{M}^{-1}\bar{\mathbf{g}}, \tag{3b}$$

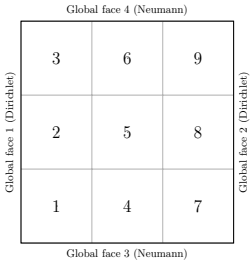
$$\boldsymbol{\Lambda}_{\mathbf{A}}\boldsymbol{\lambda} = \boldsymbol{\lambda}_{\mathbf{b}}, \tag{3c}$$

$$\mathbf{M}\mathbf{u} = (\bar{\mathbf{g}} - \mathbf{F}\boldsymbol{\lambda}). \tag{3d}$$

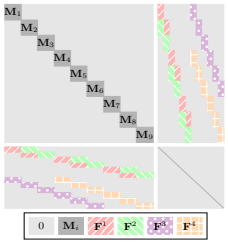
Eqs. 3b and 3d decompose into independent subproblem solves; Eq. 3c is the global trace solve. The trace operator $\boldsymbol{\Lambda}_{\mathbf{A}}$ is computed once and reused across timesteps.

2.2 Problem configuration

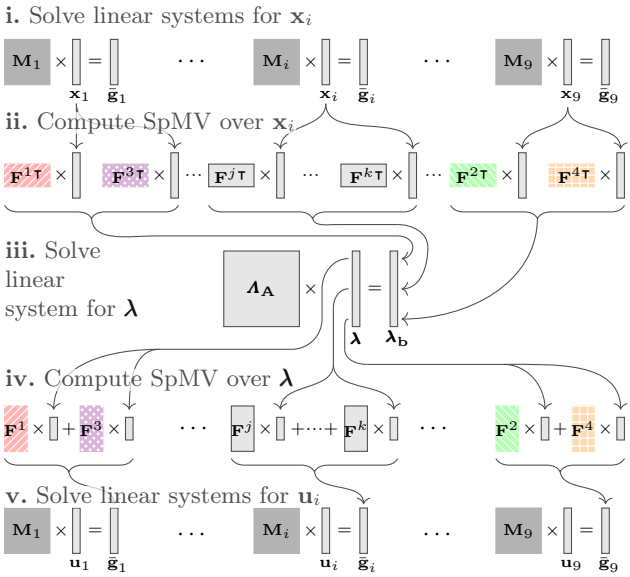
We use square subproblems of uniform size for simplicity, though extending the problem to non-uniform decomposition is straightforward. The total number of subproblems is ℓ^2, each containing n^2 grid points, giving a global problem size of $\bar{n}^2 = (n\ell)^2$. The number of internal interfaces is $r = 2\ell^2 - 2\ell$. Fig. 1a illustrates a 3×3 decomposition with $\ell^2 = 9$ and $r = 12$ interfaces. We strictly consider the 2D anti-plane shear problem, a standard benchmark problem in earthquake-cycle simulation.



(a) Division of the global domain into subproblems.



(b) Nonzero pattern of the computational domain.



(c) Task graph of the hybridized method. Note that steps **i**–**v** are conventionally computed with a barrier after each step.

Fig. 1: Visualizations of the **(a)** global domain, **(b)** computational domain, and **(c)** task graph for a 3×3 decomposition of the hybridized problem specified in Eq. 2. Submatrices of $\mathbf{F}$ are shaded in the same color/pattern if they have the same nonzero pattern.

2.3 Task-parallel algorithm

General Schur complement reductions are typically computed using large single-kernel methods (Alg. 1) because they often lack guarantees about the matrix structure. In contrast, the non-zero structure of the hybridized SBP-SAT method allows us to break this work into fine-grained computations, as shown in Fig. 1c, where the work is broken into five stages (**i**–**v**), each with multiple computational tasks.

There are read-after-write (RAW) dependencies between tasks in stages **i** and **ii** (over $\mathbf{x}_i$), tasks in stages **iii** and **iv** (over $\boldsymbol{\lambda}$), and tasks in stages **iv** and **v** (over $\mathbf{g}_i$). There is also a write-after-write (WAW) dependency between tasks in stage **ii** and the global solve in stage **iii** (over $\boldsymbol{\lambda}_b$). Conventional bulk synchronous parallelism (BSP) would require global synchronization between stages to address the RAW dependencies, as well as additional inter-task synchronization (in stage **ii**), mutex locks, or thread-private buffers with global reduction, to address the WAW dependency.

Alg. 2 describes a BSP algorithm for stages **i** and **ii**. Instead of using mutex locks or thread-private buffers with global reduction, both of which are computationally more expensive, we use a global barrier (between line 2 and 3) to eliminate the WAW dependency. The second barrier (between line 8 and 9), in particular, divides the computation into sets of non-adjacent subproblems (i.e.,

Algorithm 1: Naively parallel.

```
1  parallel for i ∈ 1 ··· ℓ²
2      Solve M_i x_i = g_i
3  x = x_1 ⊕ ··· ⊕ x_n
4  λ_b = F^T x
```

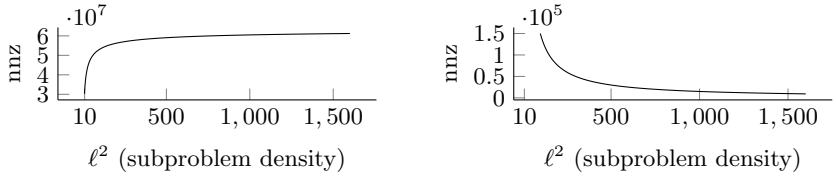
Algorithm 2: Bulk synchronous parallel.

```
1  parallel for i ∈ 1 ··· ℓ²
2      Solve M_i x_i = g_i
3  parallel for i ∈ 1 ··· ℓ²
       // First set of subproblems
4      if i/n  mod (2 − n  mod 2) ≠ 1 then
5          parallel for j ∈ 1 ··· r
6              Retrieve sub-matrix F_ij
7              if F_ij exists then
8                  λ_bj = F_ij^T x_i
9  parallel for i ∈ 1 ··· ℓ²
       // Second set of subproblems
10     if i/n  mod (2 − n  mod 2) ≠ 0 then
11         parallel for j ∈ 1 ··· r
12             Retrieve sub-matrix F_ij
13             if F_ij exists then
14                 λ_bj = F_ij^T x_i
```

Algorithm 3: Task parallel.

```
1  for i ∈ 1 ··· ℓ²
2      Task A: out(x_i)
3          Solve M_i x_i = g_i
4      for j ∈ 1 ··· r
5          Retrieve sub-matrix F_ij
6          if F_ij exists then
7              Task B: in(x_i),
                 mtx(λ_bj)
8                  λ_bj = F_ij^T x_i
```

Fig. 2: Three methods of computing stages **i** & **ii** from the task graph in Fig. 1c: using a naively parallel, single-kernel method (Alg. 1), a bulk-synchronous parallel method (Alg. 2) and a task-parallel method (Alg. 3). The task-parallel method prioritizes temporal locality, improving the likelihood that intermediate data ($\mathbf{x}_i$) will remain in cache for subsequent tasks.



(a) The number of nonzero elements in $\boldsymbol{\Lambda}_{\mathbf{A}}$ as given by Eq. 5.

(b) The number of nonzero elements in $\mathbf{M}_i$ as given by Eq. 6.

Fig. 3: The number of nonzero elements in **(a)** $\boldsymbol{\Lambda}_{\mathbf{A}}$ and **(b)** $\mathbf{M}_i$ as a function of the subproblem density (ℓ^2) for a constant global volume of $\bar{n}^2 = 1e6$.

subproblems that do not share an interface with others in the same set), eliminating write conflicts to $\boldsymbol{\lambda}_{\mathbf{b}}$ when subproblems are computed in parallel. For the example in Fig. 1a, these would be even and odd-numbered subproblem sets.

The BSP model requires more synchronization than is absolutely necessary so we analyze the dependencies between tasks across all five stages to construct a task graph, as shown by the arrows in Fig. 1c. This allows us to use task-parallelism to execute more tasks asynchronously, which reduces the synchronization overhead and increase temporal locality of intermediate data. Additionally, some intermediate data computed in one stage (*e.g.*, $\mathbf{x}_5$ in stage **i**) is consumed by more tasks in the subsequent stage (*e.g.*, stage **ii**) compared to other intermediate data (*e.g.*, $\mathbf{x}_1$), due to having more internal interfaces (*e.g.*, subproblem 5 has four interfaces, whereas subproblem 1 only has two). We further exploit this dependency structure by prioritizing tasks whose intermediate results are consumed by a greater number of downstream tasks. Because these tasks are memory-bandwidth bound, executing them before others improves temporal locality and reduces evictions of frequently reused data.

Our task-parallel algorithm for stages **i**, **ii** is described in Alg. 3. We omit the algorithm for stages **iv**, **v**, as it is similar but constructed from the edge-reversed graph of the former. We implement these routines using OpenMP's task constructs. We use the `depend` clause to assign dependencies on the intermediate data, the `mutexinoutset` clause to serialize writes to the same memory location, and the `priority` clause to prioritize the execution of tasks that generate more frequently consumed intermediate data.

3 Performance Model

A key parameter in hybridized PDE solvers is the subproblem density (*i.e.*, the number of subproblems for a given total DOFs), which significantly impacts performance [14]. However, prior work has only explored it heuristically or for a small set of values, without guidance on optimal selection. To address this, we develop a performance model that selects the optimal subproblem density for a given global volume size and processor. Our model derives from the Roofline model [27], which predicts performance based on arithmetic intensity and peak compute and memory throughput. Our model first derives the number of nonzero

elements (nnz) in $\boldsymbol{\Lambda}_{\mathbf{A}}$ (global problem) and $\mathbf{M}$ (subproblems) as functions of ℓ^2, which are then used to compute the number of arithmetic operations and bytes accessed by Eqs. 3b, 3c, and 3d. Our model then predicts execution times, which we balance between the subproblems and the global problem to identify the subproblem density that minimizes total execution time.

3.1 Deriving the number of nonzero elements

To compute $\boldsymbol{\Lambda}_{\mathbf{A}}$, we solve the sparse system $\mathbf{Mx} = \mathbf{F}$ in 3a, storing each $\mathbf{F}^i$ boundary as a $n^2 \times n$ matrix. The number of unique submatrices is $p \leq \ell^2$ (with $p = \ell^2$ for variable-coefficient problems), and the number of interfaces is $r = 2\ell^2 - 2\ell$. Solving this system yields $(p \cdot n \cdot 2r)$ vectors $\mathbf{y}$ of length n^2, and since $n = \bar{n}/\ell$, increasing the subproblem density ℓ^2 decreases each subproblem size n^2 for fixed $\bar{n}^2$. Each $\mathbf{y}$ is used in the product $\mathbf{F}^{i\intercal}\mathbf{y}$, yielding $\boldsymbol{\Lambda}_{\mathbf{A}} \in \mathbb{R}^{rn \times rn}$. The nnz in $\boldsymbol{\Lambda}_{\mathbf{A}}$ is given by

$$\mathrm{nnz}_{\boldsymbol{\Lambda}} = n^2 \sum_{i=1}^{\ell^2} (\phi_i^2), \tag{4a}$$

where $\phi \in \mathbb{Z}^{\ell^2}$ encodes the number of interfaces for each subproblem,

$$\phi = \left[\underbrace{2, 2, 2, 2}_{4}, \underbrace{3 \cdots 3}_{4\ell-8}, \underbrace{4 \cdots 4}_{(\ell-2)^2} \right], \tag{4b}$$

corresponding to corner (4), boundary $(4\ell - 8)$, and interior $((\ell - 2)^2)$ subproblems, respectively. Substituting Eq. 4b into Eq. 4a gives

$$\mathrm{nnz}_{\boldsymbol{\Lambda}}(\ell) = (\bar{n}^2/\ell^2)(16\ell^2 - 28\ell + 8). \tag{5}$$

Similarly, the second-order SBP-SAT discretization of Eq. 1a yields five nonzeros per row, so nnz in $\mathbf{M}$ is given by

$$\mathrm{nnz}_{\mathbf{M}}(\ell) = 5\bar{n}^2/\ell^2. \tag{6}$$

As shown in Fig. 3, $\mathrm{nnz}_{\boldsymbol{\Lambda}}$ grows with ℓ^2 while $\mathrm{nnz}_{\mathbf{M}}$ decreases, illustrating the cost trade-off between the global and subproblems.

3.2 Deriving the execution time model

We use $\mathrm{nnz}_{\boldsymbol{\Lambda}}$ and $\mathrm{nnz}_{\mathbf{M}}$ to derive execution time models for the global problem (Eq. 3c) and subproblems (Eq. 3d), then minimize their sum to find the optimal subproblem density. This is possible since any weighted sum of $\mathrm{nnz}_{\boldsymbol{\Lambda}}$ and $\mathrm{nnz}_{\mathbf{M}}$ is bounded by a second-degree polynomial, and is thus convex with a unique global minimum.

The global problem is modeled as memory-bound, giving

$$T_{\mathbf{\Lambda}} = \frac{\text{bytes}_{\mathbf{\Lambda}}}{\beta_{\text{peak}}}, \tag{7}$$

where $\text{bytes}_{\mathbf{\Lambda}}$ depends on the storage scheme: $(rn)^2$ or $(rn)^2/2$ for dense LU or Cholesky, and $2 \cdot \text{nnz}_{\mathbf{\Lambda}}$ for sparse methods. The subproblem execution time is modeled as

$$T_{\mathbf{M}} = \frac{\text{bytes}_{\mathbf{M}}^{(1+\varepsilon)}}{\beta_{\text{peak}}}, \tag{8}$$

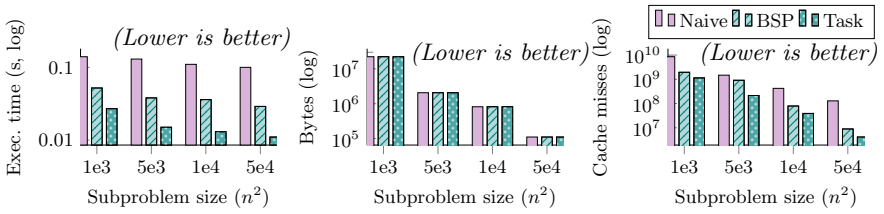
where ε is a parallel penalty term capturing solver implementation overheads, and $\text{bytes}_{\mathbf{M}}$ is $(n)^2$, $(n)^2/2$, or $2 \cdot \text{nnz}_{\mathbf{M}}$ for LU, Cholesky, or sparse storage, respectively. The optimal subproblem size is then

$$\tau(n) = \arg\min_{n} \left\{ T_{\mathbf{\Lambda}} + 2T_{\mathbf{M}} \right\}. \tag{9}$$

This model has a unique global minimum because $T_{\mathbf{M}}$ and $T_{\mathbf{\Lambda}}$ trade off with ℓ: as subproblem density increases, each subproblem shrinks, with total subproblem cost scaling as $\ell^{-2\varepsilon}$ (decreasing for $\varepsilon > 0$), while $T_{\mathbf{\Lambda}}$ grows since more interfaces enlarge the global problem. The parameter ε scales cleanly with $\bar{n}^2$, allowing it to be measured offline once and reused across problem sizes.

3.3 Model analysis

The primary role of this model is to identify subproblem sizes that balance local and global costs without exhaustive empirical tuning. Eq. 9 has a unique global minimum because $T_{\mathbf{M}}$ and $T_{\mathbf{\Lambda}}$ trade off with ℓ so the total subproblem cost scales as $\ell^{-2\varepsilon}$ (decreasing for $\varepsilon > 0$), since each subproblem shrinks faster than their



(a) Exec. time (b) Bytes acc. (c) Cache misses

Fig. 4: Comparison of three variants of our parallel hybridized PDE solver for a global volume of $\bar{n}^2 = 1\text{e}6$ grid points. Naive (Alg. 1) stores the interface matrix $\mathbf{F}$ in CSR and performs a single SpMV to compute $\mathbf{F}^\intercal\mathbf{x}$; BSP stores $\mathbf{F}$ as multiple submatrices and computes multiple SpMVs; Task eliminates implicit barriers via task-based parallelism. Fig. **(a)** shows execution time improves significantly with both optimizations, with the lowest time at the largest subproblem size due to fewer total bytes accessed, as shown in Fig. **(b)**. Fig. **(c)** confirms that L3 cache misses also decrease with each optimization.

number grows, while $T_{\boldsymbol{\Lambda}}$ increases with ℓ as each additional subproblem introduces new interfaces, growing the global problem as ℓ^2 for dense factorizations. The penalty ε captures solver-specific inefficiencies and is stable across problem sizes and platforms, so it can be computed once and reused.

4 Experimental Results

We evaluate the impact of using the bulk-synchronous (Alg. 2) and tasked-base (Alg. 3) parallel methods on the performance of our parallel hybridized PDE solver across different subproblem sizes (§4.2). We then evaluate how accurately our performance model predicts the optimal subproblem size that minimizes the overall execution time for a range of solver algorithms, global volume sizes, CPU platforms, and number of CPU cores (§4.3). Lastly, we compare the performance achieved by our parallel hybridized PDE solver when using sub-block sizes predicted by our model against using sizes from prior studies (§4.5), and against non-hybridized PDE solvers, CG and multi-frontal QR method (§4.6).

4.1 Experimental setup

We evaluate on two Intel CPU platforms, each configured with one thread per core and fixed clock frequency. All measurements are averaged over 10 runs and collected using PAPI [12]. Code was compiled with the Intel OneAPI compiler (release 2024.1.0) and Intel OpenMP library, using the `-O3` and `--fiopenmp` flags, with `OMP_NUM_THREADS=64` and `104` for the Sapphire Rapids and Ice Lake platforms, respectively, and `OMP_PLACES="cores"` and `OMP_PROC_BIND="close"` for thread placement. This configuration outperformed thread spreading, as the largely independent subproblems favor locality-preserving access patterns with limited cross-socket traffic. All linear algebra operations use Intel MKL routines, including dense Cholesky, CG, and multi-frontal QR solvers. We evaluate global volume sizes from $\bar{n}^2 = 1\text{e}6$ to $\bar{n}^2 = 3.2\text{e}7$; larger sizes are excluded due to excessive runtime and/or memory requirements of competing methods. Throughout, we report subproblem *size* (n^2) rather than subproblem *density* (ℓ^2) for clarity, where $n^2 = \frac{\bar{n}^2}{\ell^2}$ for a fixed global domain.

4.2 Impact of task-based parallelism

We evaluate the bulk-synchronous (Alg. 2) and task-parallel (Alg. 3) methods against a naive parallel baseline for a global volume of $\bar{n}^2 = 1\text{e}6$ grid points and $\ell^2 = 64$ subproblems; sufficient to fully occupy the Sapphire Rapids cores. Other global volume sizes show similar trends and are omitted. As shown in Fig. 4a, both methods improve performance over the baseline, with task parallel achieving the lowest execution time across all subproblem sizes. The largest subproblem size yields the best performance, with 3.4× and 9.1× speedups for BSP and task parallel, respectively, consistent with the reduction in total bytes accessed as shown in Fig. 4b. Fig. 4c confirms that task parallelism incurs the fewest cache

Table 1: Comparison of predicted (Model) and empirically determined optimal subproblem sizes for the Sapphire Rapids (SPR) and Ice Lake (ICX) platforms, using maximum available cores of 64 and 104, respectively. In two cases (highlighted in gray), the predicted optimal size differs from the empirically determined optimal, but only by one valid step size.

Dense Cholesky

$\bar{n}^2$	Model	SPR	Model	ICX
1e6	91	91	77	77
2e6	100	100	85	85

Sparse multi-frontal QR

$\bar{n}^2$	Model	SPR	Model	ICX
1e6	200	200	142	142
2e6	218	218	172	172
4e6	246	282	200	200

Sparse multi-frontal QR

$\bar{n}^2$	Model	SPR	Model	ICX
8e6	282	312	258	258
1.6e7	340	340	300	300
3.2e7	400	400	360	360

misses, confirming that prioritizing intermediate data reuse improves temporal locality and cache utilization.

4.3 Performance model accuracy

We evaluate our model by comparing predicted subproblem sizes against those found by exhaustive empirical search (oracle), across global volume sizes, solver algorithms, CPU platforms, and core counts. Exhaustive search is infeasible at the largest volumes due to varying memory footprints, so we restrict comparison to regimes where sufficient empirical data can be collected. We test two contrasting solvers, dense Cholesky (smaller problems) and sparse multi-frontal QR (larger problems), to demonstrate that the model generalizes across algorithms. Overall, our model matches the oracle in 79 of 90 cases (87.7%). Mismatches arise from projecting a continuous model onto discrete subproblem sizes that evenly divide the global volume. Even in incorrect cases, the execution time penalty is minimal and amounts to an average slowdown of 4.8% for mispredictions, and 1.7% across all cases.

Tab. 1 shows a subset of the optimal subproblem sizes predicted by our model compared to the empirically determined optimal sizes across the Sapphire Rapids and Ice Lake platforms when *using all available cores*. For Cholesky, our model selects the correct subproblem sizes on both systems for global volume sizes of $\bar{n}^2 = 1e6$ and $2e6$. For multi-frontal QR, our model selects the correct subproblem sizes in 10 out of 12 cases across both platforms. The two incorrect predictions occur on Sapphire Rapids and the predictions differ by only a single discrete step from the oracle due to rounding and divisibility constraints on subproblem sizes. These results demonstrate that our performance model is both accurate and robust. Notably, performance is relatively insensitive to small deviations from the model-predicted optimum, with neighboring subproblem sizes exhibiting comparable execution times.

4.4 Performance model parameters

Our model uses two theoretically derived machine parameters, α_{peak} and β_{peak}, computed directly from hardware specifications (8.2 Tflops/s and 562 GB/s for Sapphire Rapids; 5.3 Tflops/s and 762 GB/s for Ice Lake), and one empirically

derived parallel penalty ε, which captures implementation-dependent solver inefficiencies. We find $\varepsilon = 1.2$ for dense Cholesky and $\varepsilon = 1.1$ for multi-frontal QR, consistent with Cholesky exposing less parallelism. Notably, ε is stable across global volume sizes and platforms, so only a single offline measurement is needed.

4.5 Comparison against prior hybridized solvers

To demonstrate the importance of subproblem sizing, we compare against two prior studies using rectilinear meshes for second-order accurate 2D Poisson problems [23, 21]. Since source code is unavailable, we use our model to predict speedup over author-provided sizes, then validate by constructing and executing analogous problems. We also include an Intel Broadwell platform, as these studies predate our hardware. For the 1.5e6-point problem from [23] ($n^2 = 36$), our model predicts up to 3× speedup on Sapphire Rapids using the optimal subproblem size; our analogous implementation achieves up to 4.9×. For the 1.7e6-point problem from [21] ($n^2 = 53$), our model predicts up to 30× speedup, and our implementation achieves up to 31×. These results are shown in Fig. 5.

4.6 Performance against other sparse solvers

Finally, we compare our hybridized PDE solver against alternative *non-hybridized* PDE solvers to demonstrate the effectiveness of hybridization as a performance optimization strategy. We compare against both an iterative method, CG, and a sparse direct method, multi-frontal QR. When comparing each solver, we use the same method to solve the subproblems and the global problem in our hybridized method to make the comparison as fair as possible.

Fig. 6 shows the speedup achieved by our parallel hybridized PDE solver against IC(0) preconditioned CG (Fig. 6a) and multi-frontal QR (Fig. 6b) methods, for global volume sizes ranging from $\bar{n}^2 = 1e6$ to $3.2e7$, on our two test platforms. Against CG, our parallel hybridized PDE solver achieves speedup up to 4.7× on the Sapphire Rapids platform, and up to 4.4× on the Ice Lake platform, demonstrating significantly better performance. IC(0) preconditioning is

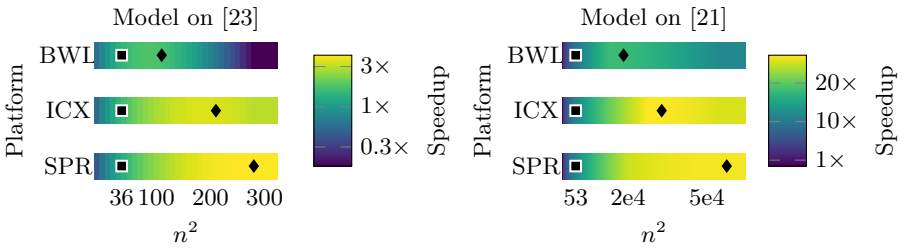


Fig. 5: Our performance model applied to the 1.5e6 problem from [23] and the 1.7e6 problem from [21]. We plot the predicted speedup over the performance of the respective author's chosen subproblem sizes. This is plotted for Intel Broadwell (BWL), Ice Lake (ICX), and Sapphire Rapids (SPR) CPU platforms.

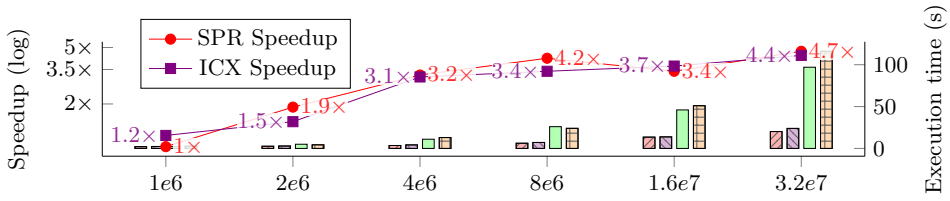
a strong baseline for structured-grid Poisson problems, typically reducing iteration counts by $3-5\times$ compared to diagonal preconditioning while maintaining good parallel scalability. The fact that our hybridized solver outperforms this well-tuned iterative method demonstrates the effectiveness of our method.

Against multi-frontal QR, our parallel algorithm achieves speedup ranging from $4.3\times$ to $12\times$ on the Sapphire Rapids platform, and speedup ranging from $5.8\times$ to $11\times$ on the Ice Lake platform. Speedup over sparse direct decreases with problem size, but for large problems direct methods hit a memory wall and become impossible to factorize without large amounts of memory. This is shown in Fig. 6b, where we see that multi-front QR runs out of memory during factorization (256 GB) for $\bar{n}^2 \geq 8e6$.

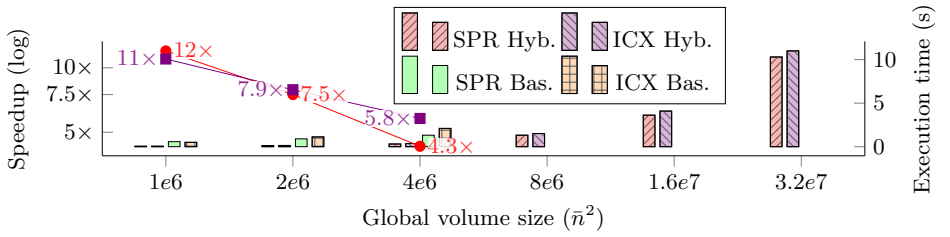
The ability of this method to scale up to larger problems beyond $\bar{n}^2 \geq 8e6$ is unique to hybridized SBP-SAT, as the subproblem interiors remain as sparse finite difference operators. Conventional HDG, with dense subproblem interiors, hits a scaling limit as the dense problem requires a larger global problem that becomes a majority of the cost at these scales.

5 Summary

This work presents the first analytical study of sparse interior hybridized methods (hybridized SBP-SAT). Unlike dense interior methods (e.g., HDG), sparse interior methods preserve sparsity within subproblems, enabling scaling to larger



(a) Comparison against CG



(b) Comparison against sparse multi-frontal QR

Fig. 6: Speedup of our parallel hybridized PDE solver (Hyb.) against baseline (Bas.) non-hybridized CG and sparse multi-frontal QR solvers (Intel MKL) on the Sapphire Rapids and Ice Lake platforms using 64 and 104 cores, respectively. The sparse QR solver ran out of memory at $\bar{n}^2 \geq$ 8e6 and could not be compared. Our optimally configured hybridized solver is significantly faster than both competing methods, even at large global volume sizes.

problem sizes without prohibitive memory costs. Building on this, we develop a performance optimization strategy and a novel task-parallel algorithm for multi-core CPUs that eliminates bulk-synchronous barriers and overlaps subproblem solves with trace assembly, achieving up to 9.1× speedup over a baseline hybridized solver. We also develop a performance model that predicts the optimal subproblem size, matching an oracle in 87.7% of cases with only 1.7% average execution time overhead. Using this optimal size, our algorithm achieves up to 4.7× speedup over preconditioned CG and up to 12× over multi-frontal QR, while requiring less memory. These results establish hybridized SBP-SAT as an effective strategy for sparse PDE solvers, remaining practical where sparse direct methods exhaust memory, and well-suited to high-order discretizations and time-dependent simulations such as earthquake cycle modeling. Future work will target GPUs, distributed-memory systems, and extensions to 3D and anisotropic problems.

References

1. Bian, H., Huang, J., Dong, R., Liu, L., Wang, X.: Csr2: A new format for simd-accelerated spmv. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). pp. 350–359 (2020)
2. Blelloch, G.E., Gibbons, P.B., Simhadri, H.V.: Low depth cache-oblivious algorithms. In: Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures. pp. 189–199. SPAA '10, Association for Computing Machinery (2010)
3. Bonnasse-Gahot, M., Calandra, H., Diaz, J., Lanteri, S.: Hybridizable discontinuous galerkin method for the 2-d frequency-domain elastic wave equations. Geophysical Journal International **213**(1), 637–659 (2018)
4. Bouteiller, A., Herault, T., Cao, Q., Schuchart, J., Bosilca, G.: Parsec: Scalability, flexibility, and hybrid architecture support for task-based applications in ecp. The International Journal of High Performance Computing Applications **39**(1), 147–166 (2025)
5. Chen, A., Erickson, B.A., Kozdon, J.E., Choi, J.: Matrix-free sbp-sat finite difference methods and the multigrid preconditioner on gpus. In: Proceedings of the 38th ACM International Conference on Supercomputing. pp. 400–412 (2024)
6. Choi, J.W., Singh, A., Vuduc, R.W.: Model-driven autotuning of sparse matrix-vector multiply on gpus. In: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 115–126. PPoPP '10, Association for Computing Machinery (2010)
7. Dobrev, V., Kolev, T., Lee, C.S., Tomov, V., Vassilevski, P.S.: Algebraic hybridization and static condensation with application to scalable h(div) preconditioning. SIAM Journal on Scientific Computing **41**(3), B425–B447 (2019)
8. Erickson, B.A., Jiang, J., Barall, M., Lapusta, N., Dunham, E.M., Harris, R., Abrahams, L.S., Allison, K.L., Ampuero, J.P., Barbot, S., et al.: The community code verification exercise for simulating sequences of earthquakes and aseismic slip (seas). Seismological Research Letters **91**(2A), 874–890 (2020)
9. Fernandez, P., Nguyen, N.C., Peraire, J.: The hybridized discontinuous galerkin method for implicit large-eddy simulation of transitional turbulent flows. Journal of Computational Physics **336**, 308–329 (2017)

10. Frigo, M., Castelletto, N., Ferronato, M., White, J.A.: Efficient solvers for hybridized three-field mixed finite element coupled poromechanics. Computers & Mathematics with Applications **91**, 36–52 (2021)
11. Guyan, R.J.: Reduction of stiffness and mass matrices. AIAA journal **3**(2), 380–380 (1965)
12. Jagode, H., Danalis, A., Anzt, H., Dongarra, J.: Papi software-defined events for in-depth performance analysis. The International Journal of High Performance Computing Applications **33**(6), 1113–1127 (2019)
13. Jaust, A., Schütz, J., Aizinger, V.: An efficient linear solver for the hybridized discontinuous galerkin method. PAMM **16**(1), 845–846 (2016)
14. King, J., Yakovlev, S., Fu, Z., Kirby, R.M., Sherwin, S.J.: Exploiting batch processing on streaming architectures to solve 2d elliptic finite element problems: a hybridized discontinuous galerkin (hdg) case study. Journal of Scientific Computing **60**, 457–482 (2014)
15. Kirby, R.M., Sherwin, S.J., Cockburn, B.: To cg or to hdg: a comparative study. Journal of Scientific Computing **51**(1), 183–212 (2012)
16. Kozdon, J.E., Erickson, B.A., Wilcox, L.C.: Hybridized summation-by-parts finite difference methods. Journal of Scientific Computing **87**(3), 85 (2021)
17. Kreiss, H.O., Scherer, G.: Finite element and finite difference methods for hyperbolic partial differential equations. In: Mathematical aspects of finite elements in partial differential equations, pp. 195–212. Elsevier (1974)
18. Lubarda, M.V., Lubarda, V.A.: Intermediate Solid Mechanics. Cambridge University Press (2020)
19. Mattsson, K., Nordström, J.: Summation by parts operators for finite difference approximations of second derivatives. Journal of Computational Physics **199**(2), 503–540 (2004)
20. Muralikrishnan, S., Bui-Thanh, T., Shadid, J.N.: A multilevel approach for trace system in hdg discretizations. Journal of Computational Physics **407**, 109240 (2020)
21. Rhebergen, S., Wells, G.N.: Preconditioning for a pressure-robust hdg discretization of the stokes equations. SIAM Journal on Scientific Computing **44**(1), A583–A604 (2022)
22. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia, PA, 2nd edn. (2003)
23. Sevilla, R., Giacomini, M., Huerta, A.: A face-centred finite volume method for second-order elliptic problems. International Journal for Numerical Methods in Engineering **115**(8), 986–1014 (2018)
24. Svärd, M., Nordström, J.: Review of summation-by-parts schemes for initial–boundary-value problems. Journal of Computational Physics **268**, 17–38 (2014)
25. Trott, C.R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D.S., Ibanez, D., et al.: Kokkos 3: Programming model extensions for the exascale era. IEEE Transactions on Parallel and Distributed Systems **33**(4), 805–817 (2021)
26. Vu, N.L., Pfeiffer, H.P., Bonilla, G.S., Deppe, N., Hébert, F., Kidder, L.E., Lovelace, G., Moxon, J., Scheel, M.A., Teukolsky, S.A., et al.: A scalable elliptic solver with task-based parallelism for the spectre numerical relativity code. Physical Review D **105**(8), 084027 (2022)
27. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (Apr 2009)