A PARALLEL SWEEPING PRECONDITIONER FOR HETEROGENEOUS 3D HELMHOLTZ EQUATIONS*

JACK POULSON†, BJÖRN ENGQUIST‡, SIWEI LI§, AND LEXING YING¶

Abstract. A parallelization of a sweeping preconditioner for 3D Helmholtz equations without internal resonance is introduced and benchmarked for several challenging velocity models. The setup and application costs of the sequential preconditioner are shown to be $O(\gamma^2 N^{4/3})$ and $O(\gamma N \log N)$, where $\gamma(\omega)$ denotes the modestly frequency-dependent number of grid points per Perfectly Matched Layer. Several computational and memory improvements are introduced relative to using black-box sparse-direct solvers for the auxiliary problems, and competitive runtimes and iteration counts are reported for high-frequency problems distributed over thousands of cores. Two open-source packages are released along with this paper: *Parallel Sweeping Preconditioner (PSP)* and the underlying distributed multifrontal solver, *Clique*.

Key words. Helmholtz, time-harmonic, sweeping, preconditioner, parallel

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction.

While definite elliptic partial differential equations can be efficiently solved by a wide variety of techniques (e.g., multigrid, ILU, or structured matrix factorizations), indefinite elliptic equations tend to be more challenging. This paper is concerned with three-dimensional heterogeneous Helmholtz equations of the form,

$$\mathcal{A}u \equiv \left[-\Delta - \frac{\omega^2}{c^2(x)}\right] u(x) = f(x), \tag{1.1}$$

where $c(x)$ is the spatially varying wave speed, and $u(x)e^{-i\omega t}$ is the time-harmonic response to acoustic wave equation with forcing function $f(x)e^{-i\omega t}$. It is important to recognize that $-\Delta$ is positive-definite and that its combination with the negative-definite $-\frac{\omega^2}{c^2}$ term results in an indefinite system.

Before discussing the overall asymptotic complexity of solution techniques, it is helpful to first motivate why high frequency problems require large numbers of degrees of freedom: Given the wave speed bounds $c_{\min} \leq c(x) \leq c_{\max}$, we can define the minimum wavelength as $\lambda_{\min} = 2\pi c_{\min}/\omega$. In order to resolve oscillations in the solution using piecewise polynomial basis functions, e.g., with finite-difference and finite-element methods, it is necessary to increase the number of degrees of freedom in each direction at least linearly with the number of wavelengths spanned by the domain. In order to combat pollution effects [4], which are closely related to phase errors in the discrete solution, one must use asymptotically more than a constant number of

*This work was partially supported by the sponsors of the Texas Consortium for Computational Seismology.

†ICES, University of Texas at Austin, 1 University Station C0200, Austin, TX, 78712 (poulson@ices.utexas.edu). This author was also supported by a CAM fellowship.

‡Department of Mathematics and ICES, University of Texas at Austin, 1 University Station C1200, Austin, TX, 78712 (engquist@ices.utexas.edu). This author was also supported by NSF grant DMS-1016577.

§Jackson School of Geosciences, University of Texas at Austin, 1 University Station C1160, Austin, TX, 78712 (siwei.li@utexas.edu).

¶Department of Mathematics and ICES, University of Texas at Austin, 1 University Station C1200, Austin, TX, 78712 (lexing@math.utexas.edu). This author was supported by NSF CAREER grant DMS-0846501, NSF grant DMS-1016577, and funding from KAUST.

grid points per wavelength with standard discretization schemes. Nevertheless, it is common practice to resolve the domain to as few as ten points per wavelength. In any case, piecewise polynomial discretizations require $\Omega(\omega^d)$ degrees of freedom in d dimensions.

Until recently, doubling the frequency of Eq. (1.1) not only increased the size of the linear system by at least a factor of 2^d, it also doubled the number of iterations required for convergence with preconditioned Krylov methods [6, 17, 18]. Thus, denoting the number of degrees of freedom in a three-dimensional finite-element or finite-difference discretization as $N = \Omega(\omega^3)$, every linear solve required $\Omega(\omega^4)$ work with iterative techniques. Engquist and Ying recently introduced two classes of *sweeping* preconditioners for Helmholtz equations without internal resonance [14, 15]: Both approaches approximate a block LDL^T factorization of the Helmholtz operator in block tridiagonal form in a manner which exploits a radiation boundary condition. The first approach performs a block tridiagonal factorization algorithm in $\mathcal{H}$-matrix arithmetic [25, 22], while the second approach approximates the Schur complements of the factorization using auxiliary problems with artificial radiation boundary conditions. Though the $\mathcal{H}$-matrix sweeping preconditioner has theoretical support for two-dimensional problems [14, 28], there is not yet justification for three-dimensional problems.

This paper therefore focuses on the second approach, which relies on multifrontal factorizations [27, 34, 12, 21] of the approximate auxiliary problems in order to achieve an $O(\gamma^2 N^{4/3})$ setup cost and an $O(\gamma N \log N)$ application cost, where $\gamma(\omega)$ denotes the number of grid points used for each Perfectly Matched Layer (PML) [29]. While the sweeping preconditioner is competitive with existing techniques even for a single right-hand side, its main advantage is for problems with large numbers of right-hand sides, as the preconditioner appears to converge in $O(1)$ iterations for problems without internal resonance. Thus, after setting up the preconditioner, typically only $O(\gamma N \log N)$ work is required for each solution.

1.1. *Moving PML* sweeping preconditioner. The focus of this paper is on parallelization of the second class of sweeping preconditioners mentioned above, which makes use of auxiliary problems with artificial radiation boundary conditions in order to approximate the Schur complements that arise during block LDL^T factorization. The approach is referred to as a *moving PML* preconditioner since the introductory paper represented the artificial radiation boundary conditions using PML.

One interpretation of radiation conditions is that they allow for the analysis of a finite portion of an infinite domain, as their purpose is to enforce the condition that waves propagate outwards and not reflect at the boundary of the truncated domain. This concept is crucial to understanding the Schur complement approximations that take place within the moving PML sweeping preconditioner which is reintroduced in this paper for the sake of completeness.

For the sake of simplicity, we will describe the preconditioner in the context of a second-order finite-difference discretization over the unit cube, with PML used to approximate a radiation boundary condition on the $x_3 = 0$ face and homogeneous Dirichlet boundary conditions applied on all other boundaries (an $x_1 x_3$ cross-section is shown in Fig. 1.1). If the domain is discretized into an $n \times n \times n$ grid, then ordering the vertices in the grid such that vertex (i_1, i_2, i_3) is assigned index $i_1 + i_2 n + i_3 n^2$

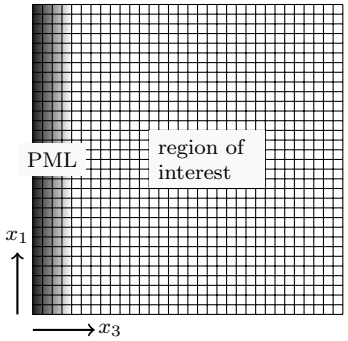

FIG. 1.1. *An x_1x_3 cross section of a cube with PML on its $x_3 = 0$ face. The domain is shaded in a manner which loosely corresponds to its extension into the complex plane.*

results in a block tridiagonal system of equations, say

$$\begin{pmatrix} A_{0,0} & A_{1,0}^T & & & \\ A_{1,0} & A_{1,1} & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & A_{n-1,n-2}^T \\ & & & A_{n-1,n-2} & A_{n-1,n-1} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-2} \\ f_{n-1} \end{pmatrix}, \tag{1.2}$$

where $A_{i,j}$ propagates sources from the i'th x_1x_2 plane into the j'th x_1x_2 plane, and the overall linear system is complex symmetric (*not* Hermitian) due to the imaginary terms introduced by the PML [15].

If we were to ignore the sparsity within each block, then the following naïve factorization and solve algorithms would be appropriate for a direct solver, where the application of S_i^{-1} implicitly makes use of the factorization of S_i.

Algorithm 1.1: Naïve block tridiagonal LDL^T factorization. $O(n^7) = O(N^{7/3})$ work is required.

$S_0 := A_{0,0}$
Factor S_0
for $i = 0, ..., n-2$ **do**
  $S_{i+1} := A_{i+1,i+1} - A_{i+1,i} S_i^{-1} A_{i+1,i}^T$
  Factor S_{i+1}

While the computational complexities of Algs. 1.1 and 1.2 are significantly higher than multifrontal algorithms [27, 12, 21], which have $O(N^2)$ factorization complexity and $O(N^{4/3})$ solve complexity for regular three-dimensional meshes, they are the conceptual starting points of the sweeping preconditioner.[1]

Suppose that we paused Alg. 1.1 after computing the i'th Schur complement, S_i, where the i'th x_1x_2 plane is in the interior of the domain (i.e., it is not in a PML region). Due to the ordering imposed on the degrees of freedom of the discretization, the

[1] In fact, they are the starting points of *both* classes of sweeping preconditioners. The $\mathcal{H}$-matrix approach essentially executes these algorithms with $\mathcal{H}$-matrix arithmetic.

Algorithm 1.2: Naïve block LDL^T solve. $O(n^5) = O(N^{5/3})$ work is required.

```
// Initialize u := f
for i = 0, ..., n-1 do
    u_i := f_i
// Apply L^{-1}
for i = 0, ..., n-2 do
    u_{i+1} := u_{i+1} - A_{i+1,i}(S_i^{-1} u_i)
// Apply D^{-1}
for i = 0, ..., n-1 do
    u_i := S_i^{-1} u_i
// Apply L^{-T}
for i = n-2, ..., 0 do
    u_i := u_i - S_i^{-1}(A_{i+1,i}^T u_{i+1})
```

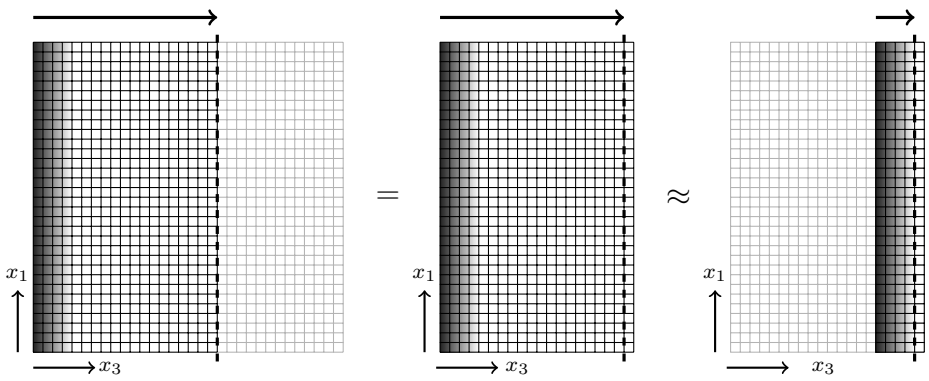

FIG. 1.2. *(Left) A depiction of the portion of the domain involved in the computation of the Schur complement of an x_1x_2 plane (marked with the dashed line) with respect to all of the planes to its left during execution of Alg. 1.1. (Middle) An equivalent auxiliary problem which generates the same Schur complement; the original domain is truncated just to the right of the marked plane and a homogeneous Dirichlet boundary condition is placed on the cut. (Right) A local auxiliary problem for generating an approximation to the relevant Schur complement; the radiation boundary condition of the exact auxiliary problem is moved next to the marked plane.*

first i Schur complements are equivalent to those that would have been produced from applying Alg. 1.1 to an auxiliary problem formed by placing a homogeneous Dirichlet boundary condition on the $(i+1)$'th x_1x_2 plane and ignoring all of the successive planes (see the left half of Fig. 1.2). While this observation does not immediately yield any computational savings, it does allow for a qualitative description of the inverse of the i'th Schur complement, S_i^{-1}: it is the restriction of the half-space Green's function of the exact auxiliary problem onto the i'th x_1x_2 plane (recall that PML can be interpreted as approximating the effect of a domain extending to infinity).

The main approximation made in the sweeping preconditioner can now be succinctly described: since S_i^{-1} is a restricted half-space Green's function which incorporates the velocity field of the first i planes, it is natural to approximate it with another restricted half-space Green's function which only takes into account the *local* velocity field, i.e., by moving the PML next to the i'th plane (see the right half of Fig. 1.2).

If we use $\gamma(\omega)$ to denote the number of grid points of PML as a function of the frequency, ω, then it is important to recognize that the depth of the approximate auxiliary problems in the x_3 direction is $\Omega(\gamma)$.[2] If the depth of the approximate auxiliary problems was $O(1)$, then combining nested dissection with the multifrontal method over the regular $n \times n \times O(1)$ mesh would only require $O(n^3)$ work and $O(n^2 \log n)$ storage [21]. However, if the PML size is a slowly growing function of frequency, then applying 2D nested dissection to the *quasi-2D* domain requires $O(\gamma^3 n^3)$ work and $O(\gamma^2 n^2 \log n)$ storage, as the number of degrees of freedom in each front increases by a factor of γ and dense factorizations have cubic complexity.

Let us denote the quasi-2D discretization of the local auxiliary problem for the i'th plane as H_i, and its corresponding approximation to the Schur complement S_i as $\tilde{S}_i$. Since $\tilde{S}_i$ is essentially a dense matrix, it is advantageous to come up with an abstract scheme for applying $\tilde{S}_i^{-1}$: Assuming that H_i was ordered in a manner consistent with the $(i_1, i_2, i_3) \mapsto i_1 + i_2 n + i_3 n^2$ global ordering, the degrees of freedom corresponding to the i'th plane come last and we find that

$$H_i^{-1} = \begin{pmatrix} \star & \star \\ \star & \tilde{S}_i^{-1} \end{pmatrix}, \tag{1.3}$$

where the irrelevant portions of the inverse have been marked with a $\star$. Then, trivially,

$$H_i^{-1} \begin{pmatrix} 0 \\ u_i \end{pmatrix} = \begin{pmatrix} \star & \star \\ \star & \tilde{S}_i^{-1} \end{pmatrix} \begin{pmatrix} 0 \\ u_i \end{pmatrix} = \begin{pmatrix} \star \\ \tilde{S}_i^{-1} u_i \end{pmatrix}, \tag{1.4}$$

which implies a method for quickly computing $\tilde{S}_i^{-1} u_i$ given a factorization of H_i:

Algorithm 1.3: Application of $\tilde{S}_i^{-1}$ to u_i given a multifrontal factorization of H_i. $O(\gamma^2 n^2 \log n)$ work is required.

Form $\hat{u}_i$ as the extension of u_i by zero over the artificial PML
Form $\hat{v}_i := H_i^{-1} \hat{u}_i$
Extract $\tilde{S}_i^{-1} u_i$ from the relevant entries of $\hat{v}_i$

From now on, we write T_i to refer to the application of the (approximate) inverse of the Schur complement for the i'th plane.

There are two more points to discuss before defining the full serial algorithm. The first is that, rather than performing an approximate LDL^T factorization using a discretization of $\mathcal{A}$, the preconditioner is instead built from a discretization of an *artificially damped* version of the Helmholtz operator, say

$$\mathcal{J} \equiv \left[-\Delta - \frac{(\omega + i\alpha)^2}{c^2(x)} \right], \tag{1.5}$$

where $\alpha \approx 2\pi$ is responsible for the artificial damping. This is in contrast to shifted Laplacian preconditioners [5, 16], where α is typically $O(\omega)$ [18], and our motivation is to avoid introducing large long-range dispersion error by damping the long range interactions in the preconditioner. Just as A refers to the discretization of the original Helmholtz operator, $\mathcal{A}$, we will use J to refer to the discretization of the artificially damped operator, $\mathcal{J}$.

[2] In all of the experiments in this paper, $\gamma(\omega)$ was either 5 or 6, and the subdomain depth never exceeded 10.

The second point is much easier to motivate: since the artificial PML in each approximate auxiliary problem is of depth $\gamma(\omega)$, processing a single plane at a time would require processing $O(n)$ subdomains with $O(\gamma^3 n^3)$ work each. Clearly, processing $O(\gamma)$ planes at once has the same asymptotic complexity as processing a single plane, and so combining $O(\gamma)$ planes reduces the setup cost from $O(\gamma^3 N^{4/3})$ to $O(\gamma^2 N^{4/3})$. Similarly, the memory usage becomes $O(\gamma N \log N)$ instead of $O(\gamma^2 N \log N)$.[3] If we refer to these sets of $O(\gamma)$ contiguous planes as *panels*, which we label from 0 to $m-1$, where $m = O(n/\gamma)$, and correspondingly redefine $\{u_i\}$, $\{f_i\}$, $\{S_i\}$, $\{T_i\}$, and $\{H_i\}$, we have the following algorithm for setting up an approximate block LDL^T factorization of the discrete artificially damped Helmholtz operator:

Algorithm 1.4: Setup phase of the sweeping preconditioner. $O(\gamma^2 N^{4/3})$ work is required.

$S_0 := J_{0,0}$
Factor S_0
for $i = 1, ..., m-1$ **do**
  Form H_i by prefixing PML to $J_{i,i}$
  Factor H_i

Once the preconditioner is set up, it can be applied using a straightforward modification of Alg. 1.2 which avoids redundant solves by combining the application of L^{-1} and D^{-1}:

Algorithm 1.5: Application of the sweeping preconditioner. $O(\gamma N \log N)$ work is required.

// Initialize $u := f$
for $i = 0, ..., m-1$ **do**
  $u_i := f_i$
// Apply L^{-1} and D^{-1}
for $i = 0, ..., m-2$ **do**
  $u_i := T_i u_i$
  $u_{i+1} := u_{i+1} - J_{i+1,i} u_i$
$u_{m-1} := T_{m-1} u_{m-1}$
// Apply L^{-T}
for $i = m-2, ..., 0$ **do**
  $u_i := u_i - T_i (J_{i+1,i}^T u_{i+1})$

Given that the preconditioner can be set up with $O(\gamma^2 N^{4/3})$ work, and applied with $O(\gamma N \log N)$ work, it provides a near-linear scheme for solving Helmholtz equations if only $O(1)$ iterations are required for convergence. The experiments in this paper seem to indicate that, as long as the velocity model does not result in resonance, the sweeping preconditioner indeed only requires $O(1)$ iterations.

Though this paper is focused on the parallel solution of Helmholtz equations, which are the time-harmonic form of acoustic wave equations, Tsuji et al. have shown

[3] Note that increasing the number of planes per panel provides a mechanism for interpolating between the sweeping preconditioner and a full multifrontal factorization.

that the moving PML sweeping preconditioner is equally effective for time-harmonic Maxwell's equations [38, 39], and we believe that the same will hold true for time-harmonic linear elasticity. The rest of the paper will be presented in the context of the Helmholtz equation, but we emphasize that the parallelization should carry over to more general wave equations in a conceptually trivial way.

1.2. Related work. A domain decomposition variant of the sweeping preconditioner was recently introduced [37] which results in fast convergence rates, albeit at the expense of requiring PML padding on both sides of each subdomain. Recalling our previous analysis with respect to the PML size, γ, the memory usage of the preconditioner scales linearly with the PML size, while the setup cost scales quadratically. Thus, the domain decomposition approach should be expected to use twice as much memory and require four times as much work for the setup phase. On the other hand, doubling the subdomain sizes allows for more parallelism in both the setup and solve phases, and less sweeps seem to be required.

Another closely related method is the *Analytic ILU factorization* [19]. Like the sweeping preconditioner, it uses local approximations of the Schur complements of the block LDL^T factorization of the Helmholtz matrix represented in block tridiagonal form. There are two crucial differences between the two methods:

- Roughly speaking, AILU can be viewed as using Absorbing Boundary Conditions (ABC's) [13] instead of PML when forming approximate subdomain auxiliary problems. While ABC's result in strictly 2D local subproblems, versus the *quasi-2D* subdomain problems which result from using PML, they are well-known to be less effective approximations of the Sommerfeld radiation condition (and thus the local Schur complement approximations are less effective). The sweeping preconditioner handles its non-trivial subdomain factorizations via a multifrontal algorithm.
- Rather than preconditioning with an approximate LDL^T factorization of the original Helmholtz operator, the sweeping preconditioner sets up an approximate factorization of a *slightly damped* Helmholtz operator in order to mitigate the dispersion error which would result from the Schur complement approximations.

These two improvements are responsible for transitioning from the $O(\omega)$ iterations required with AILU to the $O(1)$ iterations needed with the sweeping preconditioner (for problems without internal resonance).

Two other iterative methods warrant mentioning: the two-grid shifted-Laplacian approach of [8] and the multilevel-ILU approach of [6]. Though both require $O(\omega)$ iterations for convergence, they have very modest memory requirements. In particular, [8] demonstrates that, with a properly tuned two-grid approach, large-scale heterogeneous 3D problems can be solved with impressive timings.

There has also been a recent effort to extend the fast-direct methods presented in [43] from definite elliptic problems into the realm of low-to-moderate frequency time-harmonic wave equations [40, 41]. While their work has resulted in a significant constant speedup versus applying a classical multifrontal algorithm to the full 3D domain [41], their results have so far still demonstrated the same $O(N^2)$ asymptotic complexity as standard sparse-direct methods.

2. Parallel sweeping preconditioner. The setup and application stages of the sweeping preconditioner (Algs. 1.4 and 1.5) essentially consist of $m = O(n/\gamma)$ multifrontal factorizations and solves, respectively. The most important detail is that *the subdomain factorizations can be performed in parallel, while the subdomain solves*

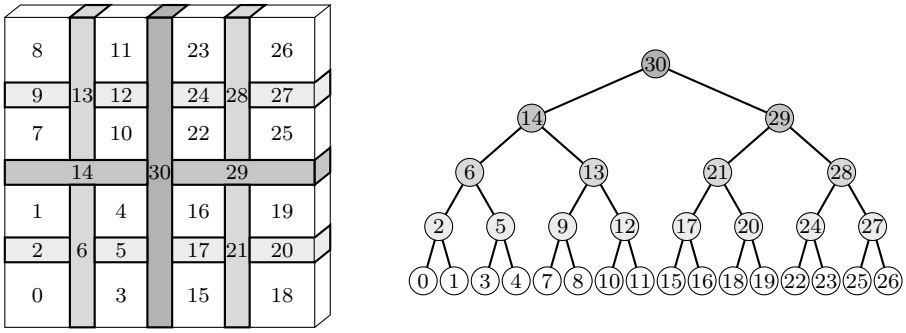

FIG. 2.1. *A separator-based supernodal elimination tree (right) over a quasi-2D subdomain (left).*

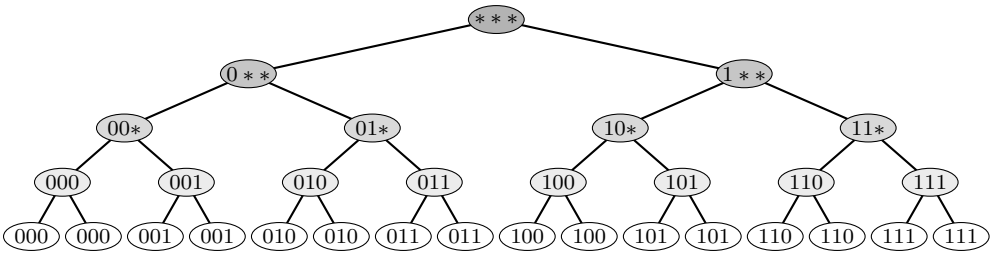

FIG. 2.2. *Overlay of the process ranks (in binary) of the owning subteams of each supernode from the elimination tree in Fig. 2.1 when the tree is assigned to eight processes using a subtree-to-subteam mapping; a '∗' is used to denote both 0 and 1, so that '00∗' represents processes 0 and 1, '01∗' represents processes 2 and 3, and '∗ ∗ ∗' represents all eight processes.*

must happen sequentially.[4] When we also consider that each subdomain factorization requires $O(\gamma^3 n^3)$ work, while subdomain solves only require $O(\gamma n^2 \log n)$ work, we see that, relative to the subdomain factorizations, subdomain solves must extract $m = O(n/\gamma)$ more parallelism from a factor of $O(\gamma^2 n/\log n)$ less operations. We thus have a strong hint that, unless the subdomain solves are carefully handled, they will be the limiting factor in the scalability of the sweeping preconditioner.

2.1. Parallel multifrontal algorithms. While a large number of techniques exist for parallelizing multifrontal factorizations and triangular solves, we focus on parallelizations which combine subtree-to-subteam [20] mappings of processes to the elimination tree [34] that also make use of two-dimensional distributions of the frontal matrices [35].[5] More specifically, we make use of supernodal [3] elimination trees defined through nested dissection (see Figs. 2.1 and 2.2), which have been shown to result in highly scalable factorizations [24, 23] and moderately scalable triangular solutions [26].

Roughly speaking, the analysis in [26] shows that, if p_F processes are used in the multifrontal factorization of our quasi-2D subdomain problems, then we must have $\gamma n = \Omega(p_F^{1/2})$ in order to maintain constant efficiency as p_F is increased; similarly, if p_S

[4] While it is tempting to try to expose more parallelism with techniques like cyclic reduction (which is a special case of a multifrontal algorithm), their straightforward application destroys the Schur complement properties that we exploit for our fast algorithm.

[5] Cf. [1], which advocates for only distributing the root frontal matrix two-dimensionally and using a one-dimensional distribution for all other fronts.

processes are used in the multifrontal triangular solves for a subdomain, then we must have $\gamma n \approx \Omega(p_S)$ (where we use $\approx$ to denote that the equality holds within logarithmic factors). Since we can simultaneously factor the $m = O(n/\gamma)$ subdomain matrices, we denote the total number of processes as p and set $p_S = p$ and $p_F = O(p/m)$; then the subdomain factorizations only require that $n^3 = \Omega(p/\gamma)$, while the subdomain solves have the much stronger constraint that $n \approx \Omega(p/\gamma)$. This last constraint should be considered unacceptable, as we have the conflicting requirement that $n^3 \approx o(p/\gamma)$ in order to store the factorizations in memory. It is therefore advantageous to consider more scalable alternatives to standard multifrontal triangular solves, even if they require additional computation.

2.2. Selective inversion. The lackluster scalability of dense triangular solves is well known and a scheme known as *selective inversion* was introduced in [32] and implemented in [31] specifically to avoid the issue; the approach is characterized by directly inverting every distributed dense triangular matrix which would have been solved against in a normal multifrontal triangular solve. Thus, after performing selective inversion, every parallel dense triangular solve can be translated into a parallel dense triangular matrix-vector multiply.

Suppose that we have paused a multifrontal LDL^T factorization just before processing a particular front, F, which corresponds to some supernode, $\mathcal{S}$. Then all of the fronts for the descendants of $\mathcal{S}$ have already been handled, and F can be partitioned as

$$F = \begin{pmatrix} F_{TL} & \star \\ F_{BL} & F_{BR} \end{pmatrix}, \tag{2.1}$$

where F_{TL} holds the Schur complement of supernode $\mathcal{S}$ with respect to all of its descendants, F_{BL} represents the coupling of $\mathcal{S}$ and its descendants to $\mathcal{S}$'s ancestors, and F_{BR} holds the Schur complement updates from the descendants of $\mathcal{S}$ for the ancestors of $\mathcal{S}$. Using hats to denote input states, e.g., $\hat{F}_{TL}$ to denote the input state of F_{TL}, the first step in processing the frontal matrix F is to overwrite F_{TL} with its LDL^T factorization, which is to say that $\hat{F}_{TL}$ is overwritten with the strictly lower portion of a unit lower triangular matrix L_F and a diagonal matrix D_F such that $\hat{F}_{TL} = L_F D_F L_F^T$.

The partial factorization of F can then be completed via the following steps:

1. Solve against L_F^T to form $F_{BL} := F_{BL} L_F^{-T}$.
2. Form the temporary copy $Z_{BL} := F_{BL}$.
3. Finalize the coupling matrix as $F_{BL} := F_{BL} D_F^{-1}$.
4. Finalize the update matrix as $F_{BR} := F_{BR} - \hat{F}_{BL} \hat{F}_{TL}^{-1} \hat{F}_{BL}^T = F_{BR} - Z_{BL} F_{BL}^T$.

After adding F_{BR} onto the parent frontal matrix, only F_{TL} and F_{BL} are needed in order to perform a multifrontal solve. For instance, applying L^{-1} to some vector x proceeds up the elimination tree (starting from the leaves) in a manner similar to the factorization; after handling all of the work for the descendants of some supernode $\mathcal{S}$, only a few dense linear algebra operations with $\mathcal{S}$'s corresponding frontal matrix, say F, are required. Denoting the portion of x corresponding to the degrees of freedom in supernode $\mathcal{S}$ by $x_{\mathcal{S}}$, we must perform:

1. $x_{\mathcal{S}} := L_F^{-1} x_{\mathcal{S}}$
2. $x_U \equiv -F_{BL} x_{\mathcal{S}}$
3. Add x_U onto the entries of x corresponding to the parent supernode.

The key insight of selective inversion is that, *if we invert each distributed dense unit lower triangular matrix L_F in place, all of the parallel dense triangular solves in a*

multifrontal triangular solve are replaced by parallel dense matrix-vector multiplies. It is also observed in [32] that the work required for the selective inversion is typically only a modest percentage of the work required for the multifrontal factorization, and that the overhead of the selective inversion is easily recouped if there are several right-hand sides to solve against.

Since each application of the sweeping preconditioner requires two multifrontal solves for each of the $m = O(n/\gamma)$ subdomains, which are relatively small and likely distributed over a large number of processes, selective inversion will be shown to yield a very large performance improvement. We also note that, while it is widely believed that direct inversion is numerically unstable, in [11] Druinsky and Toledo provide a review of (apparently obscure) results dating back to Wilkinson (in [42]) which show that $x := \text{inv}(A)*b$ is as accurate as a backwards stable solve if reasonable assumptions are met on the accuracy of $\text{inv}(A)$. Since $\text{inv}(A)*b$ is argued to be more accurate when the columns of $\text{inv}(A)$ have been computed with a backwards-stable solver, and both $\text{inv}(F_{TL})$ and $\text{inv}(F_{TL}^T)$ must be applied after selective inversion, it might be worthwhile to modify selective inversion to compute and store two different inverses of each F_{TL}: one by columns and one by rows.

2.3. Global vector distributions. The goal of this subsection is to determine an appropriate scheme for distributing global vectors, i.e., ones representing a function over the entire domain (as opposed to only over a panel). And while the factorizations themselves may have occurred on subteams of $O(p/m)$ processes each, in order to make use of all available processes for every subdomain solve, at this point we assume that each auxiliary problem's frontal tree has been selectively inverted and is distributed using a subtree-to-subteam mapping (recall Fig. 2.2) over the entire set of p processes.[6]

Since a subtree-to-subteam mapping will assign each supernode of an auxiliary problem to a team of processes, and each panel of the original 3D domain is by construction a subset of the domain of an auxiliary problem, it is straightforward to extend the supernodal subteam assignments to the full domain. We should then be able to distribute global vectors so that no communication is required for readying panel subvectors for subdomain solves (via extension by zero for interior panels, and trivially for the first panel). Since our nested dissection process does not partition in the shallow dimension of quasi-2D subdomains (see Fig. 2.1), extending a vector defined over a panel of the original domain onto the PML-padded auxiliary domain simply requires individually extending each supernodal subvector by zero in the x_3 direction.

Consider an element-wise two-dimensional cyclic distribution [30] of a frontal matrix F over q processes using an $r \times c$ process grid, where r and c are $O(\sqrt{q})$. Then the (i, j) entry will be stored by the process in the $(i \bmod r, j \bmod c)$ position in the process grid. Using the notation from [30], this distributed front would be denoted as $F[M_C, M_R]$, while its top-left quadrant would be referred to as $F_{TL}[M_C, M_R]$ (see Fig. 2.3 for a depiction of an $[M_C, M_R]$ matrix distribution).

Loosely speaking, $F[X, Y]$ means that each column of F is distributed using the scheme denoted by X, and each row is distributed using the scheme denoted by Y. For the element-wise two-dimensional distribution used for F, $[M_C, M_R]$, we have that the columns of F are distributed like Matrix Columns (M_C), and the rows of F are distributed like Matrix Rows (M_R). While this notation may seem vapid when only

[6] In cases where the available solve parallelism has been exhausted but the problem cannot be solved on less processes due to memory constraints, it would be preferable to solve with subdomains stored on subsets of processes.

$$\begin{pmatrix} 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \end{pmatrix} \qquad \begin{matrix} 0 & - & 2 & - & 4 \\ | & & | & & | \\ 1 & - & 3 & - & 5 \end{matrix}$$

FIG. 2.3. *Overlay of the owning process ranks of an 7 × 7 matrix distributed over a 2 × 3 process grid in the $[M_C, M_R]$ distribution, where M_C assigns row i to process row $i \bmod 2$, and M_R assigns column j to process column $i \bmod 3$ (left). The process grid is shown on the right.*

$$\begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} 0 \\ 2 \\ 4 \\ 1 \\ 3 \\ 5 \\ 0 \end{pmatrix}$$

FIG. 2.4. *Overlay of the owning process ranks of a vector of height 7 distributed over a 2 × 3 process grid in the $[V_C, \star]$ vector distribution (left) and the $[V_R, \star]$ vector distribution (right).*

working with a single distributed matrix, it is useful when working with products of distributed matrices. For instance, if a '$\star$' is used to represent rows/columns being redundantly stored (i.e., not distributed), then the result of every process multiplying its local submatrix of $A[X, \star]$ with its local submatrix of $B[\star, Y]$ forms a distributed matrix $C[X, Y] = (AB)[X, Y] = A[X, \star]\, B[\star, Y]$, where the last expression refers to the local multiplication process.

We can now decide on a distribution for each supernodal subvector, say $x_{\mathcal{S}}$, based on the criteria that it should be fast to form $F_{TL} x_{\mathcal{S}}$ and $F_{TL}^T x_{\mathcal{S}}$ in the same distribution as $x_{\mathcal{S}}$, given that F_{TL} is distributed as $F_{TL}[M_C, M_R]$. Suppose that we define a Column-major Vector distribution (V_C) of $x_{\mathcal{S}}$, say $x_{\mathcal{S}}[V_C, \star]$, to mean that entry i is owned by process $i \bmod q$, which corresponds to position $(i \bmod r, \lfloor i/r \rfloor \bmod c)$ in the process grid (if the grid is constructed with a column-major ordering of the process ranks; see the left side of Fig. 2.4). Then a call to `MPI_Allgather` [10] within each row of the process grid would allow for each process to collect all of the data necessary to form $x_{\mathcal{S}}[M_C, \star]$, as for any process row index $s \in \{0, 1, ..., r-1\}$,

$$\{i \in \mathbb{N}_0 : i \bmod r = s\} = \bigcup_{t=0}^{c-1} \{i \in \mathbb{N}_0 : i \bmod q = s + tr\}. \tag{2.2}$$

See the left side of Fig. 2.5 for an example of an $[M_C, \star]$ distribution of a 7×3 matrix.

Similarly, if $x_{\mathcal{S}}$ was distributed with a Row-major Vector distribution (V_R), as shown on the right side of Fig. 2.4, say $x_{\mathcal{S}}[V_R, \star]$, so that entry i is owned by the process in position $(\lfloor i/c \rfloor \bmod r, i \bmod c)$ of the process grid, then a call to `MPI_Allgather` within each column of the process grid would provide each process with the data necessary to form $x_{\mathcal{S}}[M_R, \star]$. Under reasonable assumptions, both of these redistributions can be shown to have per-process communication volume lower bounds of $\Omega(n/\sqrt{p})$ (if F_{TL} is $n \times n$) and latency lower bounds of $\Omega(\log_2(\sqrt{p}))$ [9]. We also note that translating between $x_{\mathcal{S}}[V_C, \star]$ and $x_{\mathcal{S}}[V_R, \star]$ simply requires permuting which process

$$\begin{pmatrix} \{0,2,4\} \\ \{1,3,5\} \\ \{0,2,4\} \\ \{1,3,5\} \\ \{0,2,4\} \\ \{1,3,5\} \\ \{0,2,4\} \end{pmatrix}, \qquad \begin{pmatrix} \{0,1\} \\ \{2,3\} \\ \{4,5\} \\ \{0,1\} \\ \{2,3\} \\ \{4,5\} \\ \{0,1\} \end{pmatrix}$$

FIG. 2.5. *Overlay of the owning process ranks of a vector of height 7 distributed over a 2 × 3 process grid in the $[M_C, \star]$ distribution (left) and the $[M_R, \star]$ distribution (right).*

owns each local subvector, so the communication volume would be $O(n/p)$, while the latency cost is $O(1)$.

We have thus described efficient techniques for redistributing $x_{\mathcal{S}}[V_C, \star]$ to both the $x_{\mathcal{S}}[M_R, \star]$ and $x_{\mathcal{S}}[M_C, \star]$ distributions, which are the first steps for our parallel algorithms for forming $F_{TL} x_{\mathcal{S}}$ and $F_{TL}^T x_{\mathcal{S}}$, respectively: Denoting the distributed result of each process in process column $t \in \{0, 1, ..., c-1\}$ multiplying its local submatrix of $F_{TL}[M_C, M_R]$ by its local subvector of $x_{\mathcal{S}}[M_R, \star]$ as $z^{(t)}[M_C, \star]$, it holds that $(F_{TL} x_{\mathcal{S}})[M_C, \star] = \sum_{t=0}^{c-1} z^{(t)}[M_C, \star]$. Since Eq. (2.2) also implies that each process's local data from a $[V_C, \star]$ distribution is a subset of its local data from a $[M_C, \star]$ distribution, a simultaneous summation and scattering of $\{z^{(t)}[M_C, \star]\}_{t=0}^{c-1}$ within process rows, perhaps via `MPI_Reduce_scatter` or `MPI_Reduce_scatter_block`, yields the desired result, $(F_{TL} x_{\mathcal{S}})[V_C, \star]$. An analogous process with $(F_{TL}[M_C, M_R])^T = F_{TL}^T[M_R, M_C]$ and $x_{\mathcal{S}}[M_C, \star]$ yields $(F_{TL}^T x_{\mathcal{S}})[V_R, \star]$, which can then be cheaply permuted to form $(F_{TL}^T x_{\mathcal{S}})[V_C, \star]$. Both calls to `MPI_Reduce_scatter_block` can be shown to have the same communication lower bounds as the previously discussed `MPI_Allgather` calls [9].

As discussed at the beginning of this section, defining the distribution of each supernodal subvector specifies a distribution for a global vector, say $[\mathcal{G}, \star]$. While the $[V_C, \star]$ distribution shown in the left half of Fig. 2.4 simply assigns entry i of a supernodal subvector $x_{\mathcal{S}}$, distributed over q processes, to process $i \bmod q$, we can instead choose an alignment parameter, σ, where $0 \leq \sigma < q$, and assign entry i to process $(i + \sigma) \bmod q$. If we simply set $\sigma = 0$ for every supernode, as the discussion at the beginning of this subsection implied, then at most $O(\gamma n)$ processes will store data for the root separator supernodes of a global vector, as each root separator only has $O(\gamma n)$ degrees of freedom by construction. However, there are $m = O(n/\gamma)$ root separators, so we can easily allow for up to $O(n^2)$ processes to share the storage of a global vector if the alignments are carefully chosen. It is important to notice that the top-left quadrants of the frontal matrices for the root separators can each be distributed over $O(\gamma^2 n^2)$ processes, so $O(\gamma^2 n^2)$ processes can actively participate in the corresponding triangular matrix-vector multiplications.

2.4. Parallel preconditioned GMRES(k). Since, by hypothesis, only $O(1)$ iterations of GMRES(k) will be required for convergence with the sweeping preconditioner, a cursory inspection of Algs. 1.5 and 2.1 reveals that the vast majority of the work will be in the multifrontal solves during the preconditioner application, but a modest portion will also be spent in sparse matrix-vector multiplication with the discrete Helmholtz operator, A, and the off-diagonal blocks of the discrete artificially damped Helmholtz operator, J. It is thus important to parallelize the sparse matrix-vector multiplies, but it is not crucial that the scheme be optimal, and so we simply

distribute A and J in the same manner as vectors, i.e., with the $[\mathcal{G},\star]$ distribution derived from the auxiliary problems' frontal distributions.

Algorithm 2.1: Naïve algorithm for GMRES(k) combined with the sweeping preconditioner. Corner cases such as lucky breakdowns are ignored, as are the details for the online factorization of the upper Hessenberg matrix H_j.

```
x := x_0
r := b - Ax,  ρ := ||r||_2
while ρ/||b||_2 < tolerance do
    // Run GMRES for at most k steps with current x as initial guess
    x_0 := x,  H := zeros(k, k)
    w := M^{-1} r (apply sweeping preconditioner)
    β := ||w||_2,  v_0 := w/β
    for j = 0 : k - 1 do
        d := A v_j,  δ := ||d||_2
        d := M^{-1} d (apply sweeping preconditioner)
        // Run j'th step of Arnoldi
        for i = 0 : j do
            H(i, j) := v_i^H d
            d := d - H(i, j) v_i
        δ := ||d||_2
        if j + 1 ≠ k then
            v_{j+1} := v_{j+1}/δ
        // Solve for residual minimizer and form next iterate
        y := H_j^{-1}(β e_0)
        x := x_0 + V_j y
        r := b - Ax,  ρ := ||r||_2
        if ρ/||b||_2 < tolerance then
            break
```

For performance reasons, it is beneficial to solve as many right-hand sides simultaneously as possible: both the communication latency and the costs of loading the local data from frontal and sparse matrices from main memory can be amortized over all of the right-hand sides. Another idea is to extend the so-called `trsm` algorithm for triangular solves with many right-hand sides (i.e., more right-hand sides than processes), which is well-known in the field of dense linear algebra [30], into the realm of sparse-direct solvers via the dense frontal triangular solves. This approach was not pursued in this paper due to the modest storage space available on Lonestar and is left for future work. Another performance improvement might come from exploiting block variants of GMRES [36], which can potentially lower the number of required iterations.

2.5. Clique. In order to implement the previously discussed techniques for scalable multifrontal factorizations and solves (via selective inversion), an open-source distributed multifrontal solver named Clique was built on top of Elemental [30], a library for distributed-memory dense linear algebra. In addition to being designed to support the techniques we discussed above: selective inversion, subtree-to-submesh mappings, and two-dimensional frontal matrix distributions, it was also written with a strong emphasis on *memory scalability*. This is because the sweeping preconditioner

requires large numbers of factorizations of relatively small sparse matrices, and so it is crucial that the per-process memory usage for each subdomain factorization decreases inversely with the total number of processes.

We note that Clique was designed specifically to provide a memory-scalable multifrontal implementation for our parallel sweeping preconditioner, and so there is not yet support for pivoting. We plan to add a pivoted LU factorization in the near future.

2.6. Parallel Sweeping Preconditioner (PSP). Given the discussion in Section 2, it is most convenient to describe our prototype implementation of a parallel sweeping preconditioner based upon its deviations from our proposed approach. The primary difference is that there is not yet support for simultaneously factoring the subdomain auxiliary problems and then redistributing each frontal tree to the entire set of processes. This will certainly lead to large improvements in the scalability of the setup phase, but it is left for future work.

3. Experimental results. Our experiments were performed on the Texas Advanced Computing Center (TACC) machine, Lonestar, which is comprised of 1,888 compute nodes, each equipped with two hex-core 3.33 GHz processors and 24 GB of memory, which are connected with QDR InfiniBand using a fat-tree topology. Our tests launched eight MPI processes per node in order to provide each MPI process with 3 GB of memory.

Our experiments took place over five different 3D velocity models:

- A uniform background with a high-contrast barrier. The domain is the unit cube and the wave speed is 1 except in $[0,1] \times [0.25, 0.3] \times [0, 0.75]$, where it is 10^{10}.
- A wedge problem over the unit cube, where the wave speed is set to 2 if $Z \leq 0.4 + 0.1x_2$, 1.5 if otherwise $Z \leq 0.8 - 0.2x_2$, and 3 in all other cases.
- A two-layer model defined over the unit cube, where $c = 4$ if $x_2 < 0.5$, and $c = 1$ otherwise.
- A waveguide over the unit cube: $c(\mathbf{x}) = 1.25(1 - 0.4e^{-32(|x_1-0.5|^2+|x_2-0.5|^2)})$.
- The SEG/EAGE Overthrust model [2], see Fig. 3.2.

In all of the following experiments, the shortest wavelength of each model is resolved with roughly ten grid points and the performance of the preconditioner is tested using the following four forcing functions:

- a single *shot* centered at $\mathbf{x}_0$, $f_0(\mathbf{x}) = ne^{-10n\|\mathbf{x}-\mathbf{x}_0\|^2}$,
- three shots, $f_1(\mathbf{x}) = \sum_{i=0}^{2} ne^{-10n\|\mathbf{x}-\mathbf{x}_i\|^2}$,
- a Gaussian beam centered at $\mathbf{x}_2$ in direction $\mathbf{d}$, $f_2(\mathbf{x}) = e^{i\omega\mathbf{x}\cdot\mathbf{d}}e^{-4\omega\|\mathbf{x}-\mathbf{x}_2\|^2}$, and
- a plane wave in direction $\mathbf{d}$, $f_3(\mathbf{x}) = e^{i\omega\mathbf{x}\cdot\mathbf{d}}$,

where $\mathbf{x}_0 = (0.5, 0.5, 0.1)$, $\mathbf{x}_1 = (0.25, 0.25, 0.1)$, $\mathbf{x}_2 = (0.75, 0.75, 0.5)$, and $\mathbf{d} = (1, 1, -1)/\sqrt{3}$. Note that, in the case of the Overthrust model, these source locations should be interpreted proportionally (e.g., an x_3 value of 0.1 means a depth which is 10% of the total depth of the model). Due to the oscillatory nature of the solution, we simply choose the zero vector as our initial guess in all experiments.

The first experiment was meant to test the convergence rate of the sweeping preconditioner over domains spanning 50 wavelengths in each direction (resolved to ten points per wavelength) and each test made use of 256 nodes of Lonestar. During the course of the tests, it was noticed that a significant amount of care must be taken when setting the amplitude of the derivative of the PML takeoff function (i.e., the "C" variable in Eq. (2.1) from [15]). For the sake of brevity, hereafter we refer to

	velocity model			
	barrier	wedge	two-layers	waveguide
Hz	50	75	50	37.5
PML amplitude	3.0	4.0	4.0	2.0
# of its.	28	49	48	52

TABLE 3.1

The number of iterations required for convergence for four model problems (with four forcing functions per model). The grid sizes were 500^3 and roughly 50 wavelengths were spanned in each direction. Five grid points were used for all PML discretizations, four planes were processed per panel, and the damping factors were all set to 7.

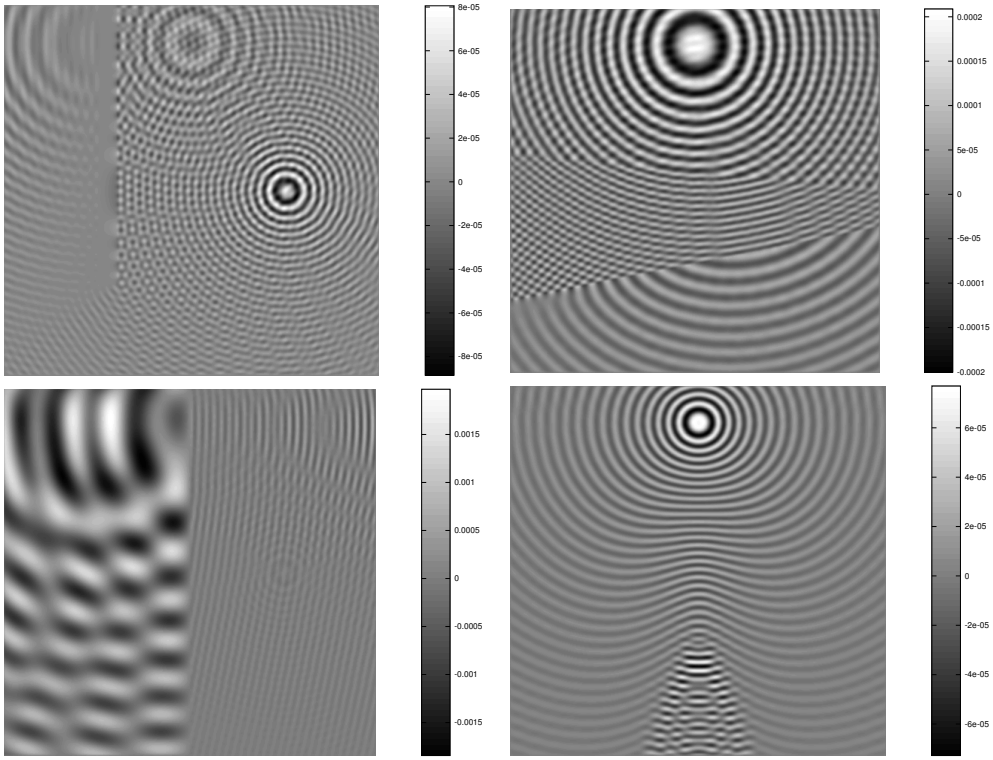

FIG. 3.1. *A single x_2x_3 plane from each of the four analytical velocity models over a 500^3 grid with the smallest wavelength resolved with ten grid points. (Top-left) the three-shot solution for the barrier model near $x_1 = 0.7$, (bottom-left) the three-shot solution for the two-layer model near $x_1 = 0.7$, (top-right) the single-shot solution for the wedge model near $x_1 = 0.7$, and (bottom-right) the single-shot solution for the waveguide model near $x_1 = 0.55$.*

this variable as the *PML amplitude*. A modest search was performed in order to find a near-optimal value for the PML amplitude for each problem. These values are reported in Table 3.1 along with the number of iterations required for the relative residuals for all four forcing functions to reduce to less than 10^{-5}.

It was also observed that, at least for the waveguide problem, the convergence rate would be significantly improved if 6 grid points of PML were used instead of 5. In particular, the 52 iterations reported in Table 3.1 reduce to 27 if the PML size is increased by one. Interestingly, the same number of iterations are required for convergence of the waveguide problem at half the frequency (and half the resolution)

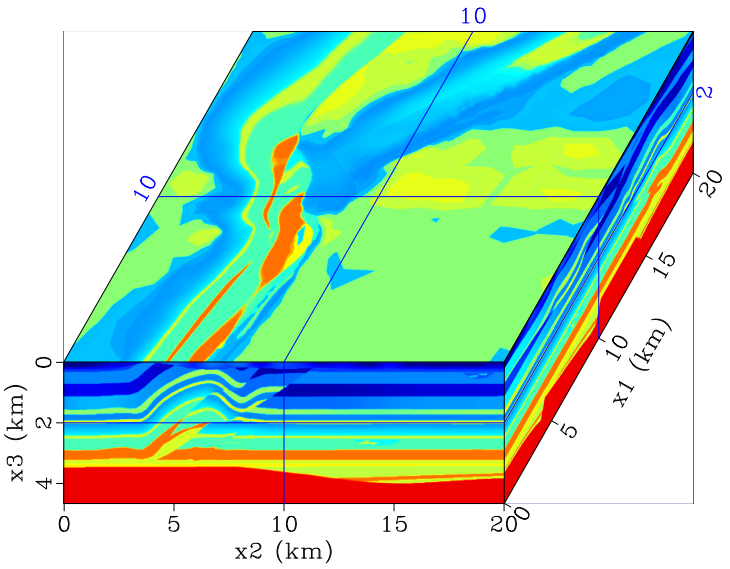

FIG. 3.2. *Three cross-sections of the SEG/EAGE Overthrust velocity model, which represents an artificial* 20 km × 20 km × 4.65 km *domain, containing an* overthrust *fault, using samples every* 25 m. *The result is an* 801 × 801 × 187 *grid of wave speeds varying discontinuously between* 2.179 km/sec *(blue) and* 6.000 km/sec *(red).*

with five grid points of PML. Thus, it appears that the optimal PML size is a slowly growing function of the frequency.[7] We also note that, though it was not intentional, each of the these first four velocity models is invariant in one or more direction, and so it would be straightforward to sweep in a direction such that only $O(1)$ panel factorizations would need to be performed, effectively reducing the complexity of the setup phase to $O(\gamma^2 N)$.

The last experiment was meant to simultaneously test the convergence rates and scalability of the new sweeping preconditioner using the Overthrust velocity model (see Fig. 3.2) at various frequencies, and the results are reported in Table 3.2. It is important to notice that this is not a typical weak scaling test, as the number of grid points per process was fixed, *not* the local memory usage or computational load, which both grow superlinearly with respect to the total number of degrees of freedom. Nevertheless, including the setup phase, it took less than three minutes to solve the 3.175 Hz problem with four right-hand sides with 128 cores, and just under seven and a half minutes to solve the corresponding 8 Hz problem using 2048 cores. Also, while it is by no means the main message of this paper, the timings without making use of selective inversion are also reported in parentheses, as the technique is not widely implemented.[8]

4. Conclusions. A parallelization of the *moving PML* sweeping preconditioner has been presented which has allowed us to efficiently solve resonance-free 3D Helmholtz equations in parallel with essentially $O(1)$ iterations, with the only observed frequency-dependence arising from a moderate growth in the PML size with increasing frequency. This size of the PML, $\gamma(\omega)$ was explained to result in a linear growth in the memory

[7] A similar observation is also made in [37].
[8] Other than Clique, the only other implementation appears to be in DSCPACK [31].

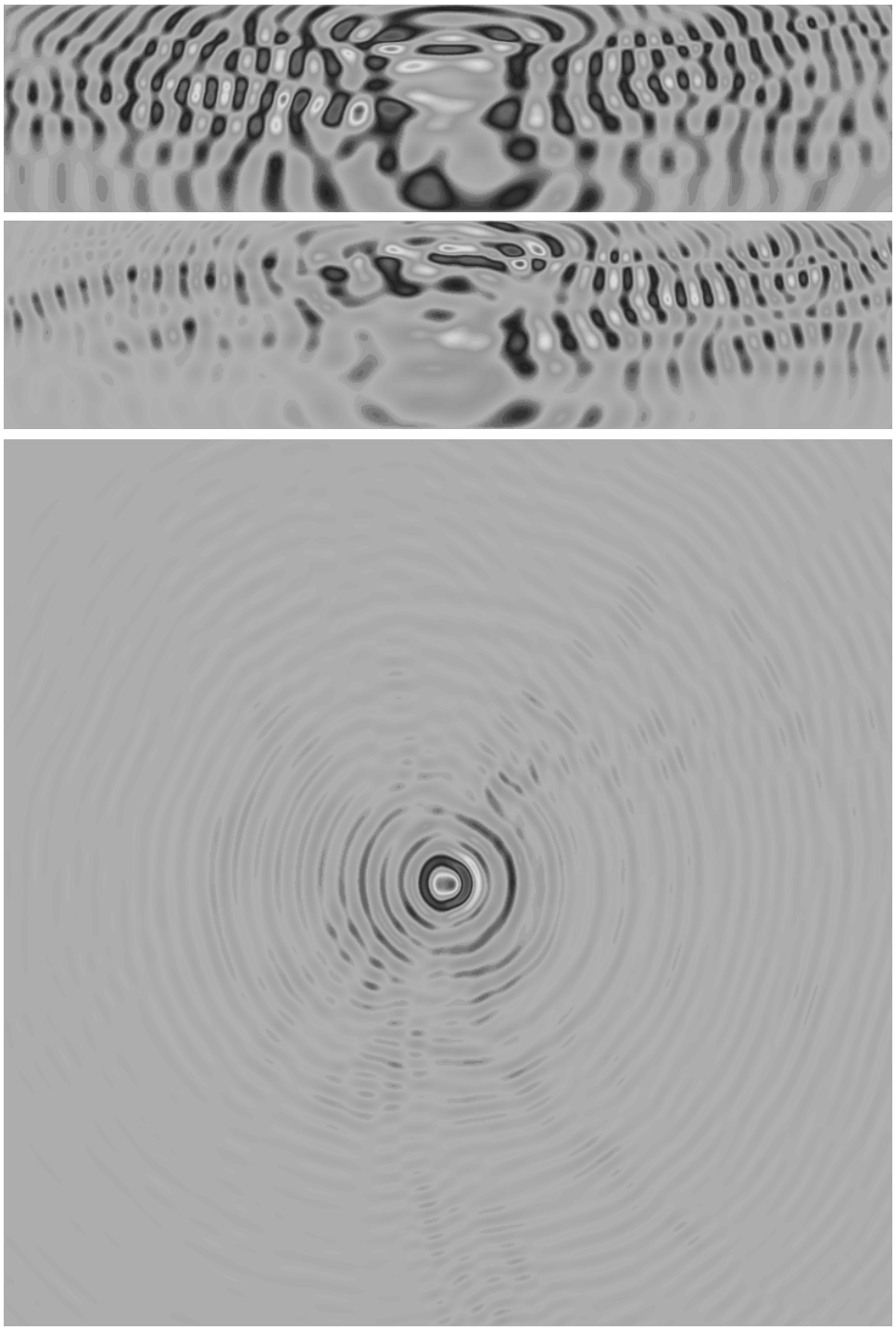

FIG. 3.3. *Three planes from an 8 Hz solution with the Overthrust model at its native resolution,* $801 \times 801 \times 187$*, with a single localized shot at the center of the* x_1x_2 *plane at a depth of 456 m: (top) a* x_2x_3 *plane near* $x_1 = 14$ *km, (middle) an* x_1x_3 *plane near* $x_2 = 14$ *km, and (bottom) an* x_1x_2 *plane near* $x_3 = 0.9$ *km.*

	number of processes				
	128	256	512	1024	2048
Hz	3.175	4	5.04	6.35	8
grid	$319\times319\times75$	$401\times401\times94$	$505\times505\times118$	$635\times635\times145$	$801\times801\times187$
setup	48.40 (46.23)	66.33 (63.41)	92.95 (86.90)	130.4 (120.6)	193.0 (175.2)
apply	1.87 (4.26)	2.20 (5.11)	2.58 (9.61)	2.80 (13.3)	3.52 (24.5)
3 digits	42	44	42	39	40
4 digits	54	57	57	58	58
5 digits	63	69	70	68	72

TABLE 3.2

Convergence rates and timings (in seconds) on TACC's Lonestar for the SEG/EAGE Overthrust model, where timings in parentheses do not make use of selective inversion. All cases used a complex double-precision second-order finite-difference stencil with five grid spacings for all PML (with a magnitude of 7.5), and a damping parameter of 2.25π*. The preconditioner was configured with four planes per panel and eight processes per node. The 'apply' timings refer to a single application of the preconditioner to four right-hand sides, and all timings in parentheses do not make use of selective inversion.*

requirements of the preconditioner and a quadratic growth in the setup cost. Results were then presented for a variety of models, one of which had a velocity field which varied by ten orders of magnitude, and convergence was shown to be essentially independent of frequency for the challenging Overthrust model.

Also, despite the requirement that each panel must be solved against one at a time when applying the preconditioner, a custom approach was introduced and implemented which eliminates most of the communication associated with performing traditional black-box sparse-direct factorizations and solves over each subdomain. These implementations are now released as part of the open-source packages Clique and Parallel Sweeping Preconditioner (PSP). There are at least five important directions for future work:

- developing a heuristic for tailoring the PML profile to the velocity field,
- extending the preconditioner to more general discretizations and time-harmonic wave equations,
- finding a fast preconditioner for problems with internal resonance (perhaps through more general local auxiliary problems),
- testing the performance improvements resulting from simultaneously factoring the subdomain problems and then redistributing the frontal trees, as well as a `trsm` approach to solving many right-hand sides, and
- making use of a distributed sparse eigensolver to gain a better understanding of behavior of the spectrum of the preconditioned 3D operator.

Availability. The distributed dense linear algebra library, Elemental, is available under the New BSD License at `http://code.google.com/p/elemental`. The distributed multifrontal solver, Clique, is available under the GPLv3 at `http://github.com/poulson/Clique`. The Parallel Sweeping Preconditioner (PSP) code is available under the GPLv3 at `http://github.com/poulson/PSP`.

Acknowledgments. The authors acknowledge TACC for usage of their computing resources and thank Bill Barth for suggesting the `tacc_affinity` option and Tommy Minyard for helping with several large runs. We also thank Anshul Gupta for WSMP details and the referees for their insightful comments.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal., 23 (2001), no. 1, pp. 15–41.

[2] F. Aminzadeh, J. Brac, and T. Kunz, *3-D Salt and Overthrust Models*, SEG/EAGE 3-D Modeling Series 1, Society of Exploration Geophysicists, Tulsa, OK, 1997.

[3] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large sparse linear systems on vector supercomputers*, Internat. J. Supercomputer Applications, 1 (1987), pp. 10–30.

[4] I. M. Babuška and S. A. Sauter, *Is the pollution effect of the FEM avoidable for the Helmholtz equation considering high wave numbers?*, SIAM Review, 42 (2000), no. 3, pp. 451–484.

[5] A. Bayliss, C. Goldstein, and E. Turkel, *An iterative method for the Helmholtz equation*, J. Comput. Phys., 49 (1983), pp. 443–457.

[6] M. Bollhoefer, M. Grote, and O. Schenk, *Algebraic multilevel preconditioner for the Helmholtz equation in heterogeneous media*, SIAM J. Sci. Comp., 31 (2009), pp. 3781–3805.

[7] M. Bollhoefer and Y. Saad, *Multilevel preconditioners constructed from inverse-based ILUs*, SIAM J. Sci. Comp., 27 (2006), pp. 1627–1650.

[8] H. Calandra, S. Gratton, X. Pinel, and X. Vasseur, *An improved two-grid preconditioner for the solution of three-dimensional Helmholtz problems in heterogeneous media*, CERFACS, Toulouse, France, Technical Report, 2012, TR/PA/12/2. Available at: `http://www.cerfacs.fr/algor/reports/2012/TR_PA_12_2.pdf`.

[9] E. Chan, M. Heimlich, A. Purkayastha, and R. A. van de Geijn, *Collective communication: theory, practice, and experience*, Concurrency and Computation: Practice and Experience, 19 (2007), no. 13, pp. 1749–1783.

[10] J. J. Dongarra and D. W. Walker, *MPI: A standard message passing interface*, Supercomputer, 12 (1996), no. 1, pp. 56–68.

[11] A. Druinsky and S. Toledo, *How accurate is* $\mathrm{inv}(A) * b?$, CoRR, abs/1201.6035 (2012), 9 pages. Available at: `http://arxiv.org/abs/1201.6035`.

[12] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[13] B. Engquist and A. Majda, *Absorbing Boundary Conditions for the numerical simulation of waves*, Mathematics of Computation, 31 (1977), pp. 629–651.

[14] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation*, Commun. on Pure and App. Math., 64 (2011), pp. 697–735.

[15] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers*, SIAM J. Multiscale Modeling and Simulation, 9 (2011), pp. 686–710.

[16] Y. Erlangga, C. Vuik, and C. Oosterlee, *On a class of preconditioners for solving the Helmholtz equation*, Applied Numer. Math., 50 (2004), pp. 409–425.

[17] Y. Erlangga, *Advances in iterative methods and preconditioners for the Helmholtz equation*, Archives Comput. Methods in Engin., 15 (2008), pp. 37-66.

[18] O. G. Ernst and M. J. Gander, *Why it is difficult to solve Helmholtz problems with classical iterative methods*, in Numerical Analysis of Multiscale Problems, I. Graham, T. Hou, O. Lakkis, and R. Scheichl, eds., Springer-Verlag, New York, NY, 2011, pp. 325–363.

[19] M. J. Gander and F. Nataf, *AILU for Helmholtz problems: a new preconditioner based on the analytic parabolic factorization*, in J. Comput. Acoustics, 9 (2001), pp. 1499–1506.

[20] A. George, J. W. H. Liu, and E. Ng, *Communication reduction in parallel sparse cholesky factorization on a hypercube*, in Hypercube Multiprocessors, M. T. Heath, ed., SIAM, Philadelphia, PA, 1987, pp. 576–586.

[21] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[22] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of* $\mathcal{H}$*-matrices*, Computing, 70 (2003), no. 4, pp. 295–334.

[23] A. Gupta, S. Koric, and T. George, *Sparse matrix factorization on massively parallel computers*, Proc. of Conf. on High Perf. Comp. Networking, Storage, and Anal. (SC '09), ACM, New York, NY, 2009. Article 1, 12 pages. Available at: `http://doi.acm.org/10.1145/1654059.1654061`.

[24] A. Gupta, G. Karypis, and V. Kumar, *A highly scalable parallel algorithm for sparse matrix factorization*, IEEE Trans. Parallel and Dist. Systems, 8 (1997), no. 5, pp. 502–520.

[25] W. Hackbusch, *A sparse matrix arithmetic based on* $\mathcal{H}$*-matrices. I. Introduction to* $\mathcal{H}$*-matrices*, Computing, 62 (1999), no. 2, pp. 89–108.

[26] M. Joshi, A. Gupta, G. Karypis, and V. Kumar, *A high-performance two dimensional scalable parallel algorithm for solving sparse triangular systems*, Proc. of Internat. Conf.

on High Perf. Comp. (HiPC), (1997), pp. 137–143.
[27] J. W. H. Liu, *The multifrontal method for sparse matrix solution: theory and practice*, SIAM Rev., 34 (1992), no. 1, pp. 82–109.
[28] P.-G. Martinsson and V. Rokhlin, *A fast direct solver for scattering problems involving elongated structures*, J. Comput. Phys., 221 (2007), no. 1, pp. 288–302.
[29] S. G. Johnson, *Notes on perfectly matched layers (PMLs)*, Massachusetts Institute of Technology, Technical Report, 2007; updated 2010. Available at: `http://www-math.mit.edu/~stevenj/18.369/pml.pdf`.
[30] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, *Elemental: a new framework for distributed memory dense matrix computations*, ACM Trans. Math. Software, Note: to appear.
[31] P. Raghavan, *Domain-Separator Codes for the parallel solution of sparse linear systems*, The Pennsylvania State University, University Park, PA, Technical Report, 2002, CSE-02-004.
[32] P. Raghavan, *Efficient parallel sparse triangular solution using selective inversion*, Parallel Processing Letters, 8 (1998), no. 1, pp. 29–40.
[33] Y. Saad and M. H. Schultz, GMRES*: A generalized minimal residual method for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[34] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), no. 3, pp. 256–276.
[35] R. Schreiber, *Scalability of sparse direct solvers*, in Graph Theory and Sparse Matrix Computation, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer-Verlag, New York, NY, 1993, pp. 191–209.
[36] V. Simoncini and E. Gallopoulos, *Convergence properties of block GMRES and matrix polynomials*, Linear Algebra and its Applications, 247 (1996), pp. 97–119.
[37] C. Stolk, *A rapidly converging domain decomposition method for the Helmholtz equation*, CoRR, abs/1208.3956 (2012), 14 pages. Available at: `http://arxiv.org/abs/1208.3956`.
[38] P. Tsuji, B. Engquist, and L. Ying, *A sweeping preconditioner for time-harmonic Maxwell's equations with finite elements*, J. Comp. Phys, Note: to appear.
[39] P. Tsuji and L. Ying, *A sweeping preconditioner for Yee's finite difference approximation of time-harmonic Maxwell's Equations*, J. Frontiers of Math. China, Note: to appear.
[40] S. Wang, M. V. de Hoop, and J. Xia, *On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver*, Geophysical Prospecting, 59 (2011), pp. 857–873.
[41] S. Wang, X. S. Li, J. Xia, Y. Situ, and M. V. de Hoop, *Efficient scalable algorithms for hierarchically semiseparable matrices*, Submitted to SIAM J. Sci. Comput., 2011. Available at: `http://www.math.purdue.edu/~xiaj/work/parhss.pdf`.
[42] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Prentice-Hall, Englewood Cliffs, N. J., 1963.
[43] J. Xia, S. Chandrasekaran, M. Gu, and X. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), no. 3, pp. 1382–1411.