Fast preconditioners for time-harmonic wave equations

Jack Poulson[1] Lexing Ying[1,2] Björn Engquist[1,2]
Sergey Fomel[1,3] Siwei Li[1,3]

[1] ICES, UT Austin

[2] Department of Mathematics, UT Austin

[3] Jackson School of Geosciences

ICERM, January 9, 2012
(revised January 15, 2012)

Outline

Time-harmonic wave equations

Sweeping preconditioners
- $\mathcal{H}$-matrix approach
- Moving PML approach
  - General algorithm
  - Scalability issues

Results

Conclusions

Outline

Time-harmonic wave equations

Wave equations are often approximated by superimposing solutions of their time-harmonic form.

Three common categories:

- Helmholtz equation (from acoustic wave equation)
- Time-harmonic Maxwell's equations
- Time-harmonic linear elasticity

Our strategy is independent of the specifics of the equation and heavily exploits absorbing boundary conditions.[1]

This talk focuses on the simplest case, the Helmholtz equation.

[1] P. Tsuji et al., "A sweeping preconditioner for time-harmonic Maxwell's equations with finite elements"

The Helmholtz equation

$$\left[-\Delta - \frac{\omega^2}{c^2(x)}\right] u(x) = f(x), \quad x \in \Omega \subset \mathbb{R}^d$$

- Helmholtz operator is elliptic, but <u>indefinite</u>
- With real Dirichlet boundary conditions, usual discretizations will be real symmetric (Hermitian) and indefinite
- Sommerfeld radiation condition often imposed on at least one side, but PML yields complex symmetric (non-Hermitian) matrices (least squares methods are another story...)
- Solving large 3d Helmholtz equations is challenging:
  - Standard preconditioners ineffective for high frequencies
  - Sparse-direct solves prohibitively expensive (with n grid points per dimension, $\mathcal{O}(N^2) = \mathcal{O}(n^6)$ work)

The damped Helmholtz equation

$$\left[-\Delta - \frac{(\omega + i\alpha)^2}{c^2(x)}\right] u(x) = f(x), \quad \alpha \approx 2\pi$$

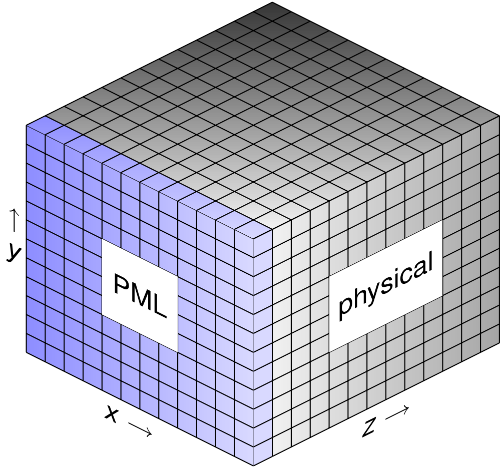
Rough idea: the preconditioning operator's long-range interactions will be less accurate than for short-range, so damp waves by adding a positive imaginary component to the frequency.

- Basic strategy is to use approximate inverse of damped Helmholtz equation as preconditioner for GMRES
- The damping parameter effects the convergence rate and is velocity and frequency dependent, but it can typically be chosen near 2π.

Outline

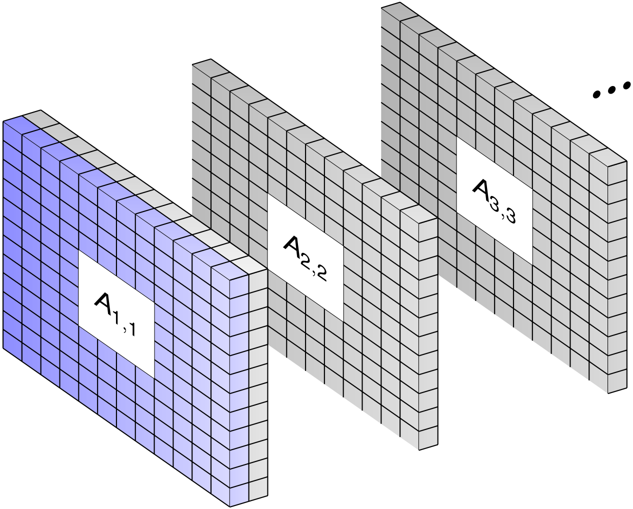
Sweeping preconditioners

Engquist and Ying recently introduced two sweeping preconditioners. Both approximate Schur complements in block LDL^T factorization with a particular ordering:



Sweeping preconditioners

Engquist and Ying recently introduced two sweeping preconditioners. Both approximate Schur complements in block LDL^T factorization with a particular ordering:



Sweeping preconditioners

Engquist and Ying recently introduced two sweeping preconditioners. Both approximate Schur complements in block LDL^T factorization with a particular ordering:

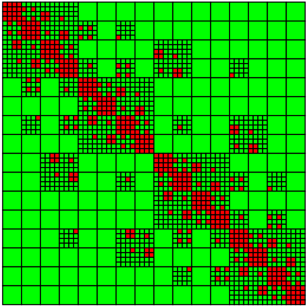
$$\begin{pmatrix} A_{1,1} & A_{2,1}^T & & \\ A_{2,1} & A_{2,2} & \ddots & \\ & \ddots & \ddots & A_{m,m-1}^T \\ & & A_{m,m-1} & A_{m,m} \end{pmatrix} = L_1 \cdots L_{n-1} \begin{pmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_m \end{pmatrix} L_{n-1}^T \cdots L_1^T,$$

- A is block-tridiagonal discrete damped Helmholtz operator
- Each block corresponds to one panel
- $A_{1,1}$ must correspond to a boundary panel with PML
- $S_i^{-1} = (A_{i,i} - A_{i,i-1} S_{i-1}^{-1} A_{i-1,i})^{-1}$, restricted half-space Green's function!
- Each L_i is a block Gauss transform[2], $L_i = I + E_{i+1} A_{i+1,i} S_i^{-1} E_i^T$.

[2] The elementary matrix kind, not a sum of Gaussians

Outline

$\mathcal{H}$-matrix approach

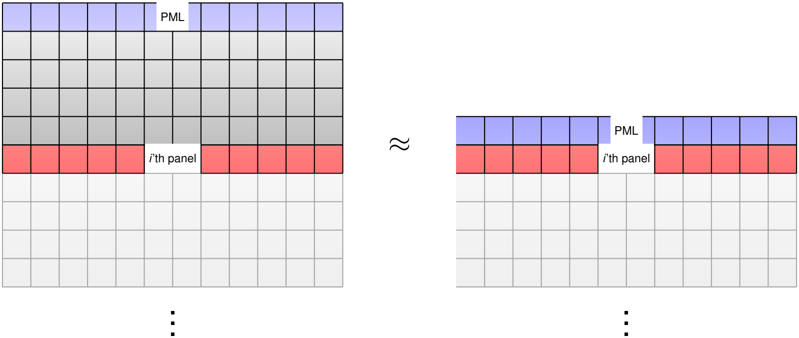


- Original sweeping preconditioner approach
- "Simply" updates and inverts Schur complements of implicit block LDL^T factorization of damped Helmholtz in particular ordering in $\mathcal{H}$-matrix arithmetic
- Inverting $\mathcal{H}$-matrices in parallel is more expensive but scalable (with Schultz iteration)
- Subject of another talk (PP12)...sandbox code called `DMHM`

Outline

Moving PML approach

Key point: S_i^{-1} is the discrete halfspace Green's function restricted to the i'th panel. **Approximate by putting an artificial absorbing boundary condition directly on the panel (which preserves sparsity).**[3]



[3] C.f. Atle and Engquist, "On surface radiation conditions for high-frequency wave scattering"

Moving PML approach

Key point: S_i^{-1} is the discrete halfspace Green's function restricted to the i'th panel. **Approximate by putting an artificial absorbing boundary condition directly on the panel (which preserves sparsity).**[3]

The preconditioner setup is just sparse-direct LDL^T factorizations on each PML-padded subdomain. With $O(n)$ subdomains with $O(n^2)$ degrees of freedom each, complexity is

$$O(n(n^2)^{3/2}) = O(n^4) = O(N^{4/3}),$$

and memory requirement is only

$$O(n(n^2 \log n^2)) = O(n^3 \log n) = O(N \log N)$$

[3] C.f. Atle and Engquist, "On surface radiation conditions for high-frequency wave scattering"

Moving PML approach

Key point: S_i^{-1} is the discrete halfspace Green's function restricted to the i'th panel. **Approximate by putting an artificial absorbing boundary condition directly on the panel (which preserves sparsity).**[3]

Each preconditioner application requires two solves against each subdomain (one each for solving against L and L^T). The application complexity is thus

$$O(n(n^2 \log n)) = O(n^3 \log n) = O(N \log N).$$

Note that subdomains must be solved against one at a time!

[3] C.f. Atle and Engquist, "On surface radiation conditions for high-frequency wave scattering"

Applying approximate Green's functions

$$S_i^{-1} g_i \approx v_i, \quad \begin{pmatrix} * \\ \vdots \\ * \\ v_i \end{pmatrix} = H_i^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ g_i \end{pmatrix}$$

Applying approximate Green's function takes three steps:

1. Extend right-hand side by zeroes on the artificial PML region

$$g_i \mapsto \begin{pmatrix} 0 \\ \vdots \\ 0 \\ g_i \end{pmatrix}$$

Applying approximate Green's functions

$$S_i^{-1} g_i \approx v_i, \quad \begin{pmatrix} * \\ \vdots \\ * \\ v_i \end{pmatrix} = H_i^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ g_i \end{pmatrix}$$

Applying approximate Green's function takes three steps:

2. Perform sparse-direct solve against H_i

$$\begin{pmatrix} * \\ \vdots \\ * \\ v_i \end{pmatrix} := H_i^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ g_i \end{pmatrix}$$

Applying approximate Green's functions

$$S_i^{-1} g_i \approx v_i, \quad \begin{pmatrix} * \\ \vdots \\ * \\ v_i \end{pmatrix} = H_i^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ g_i \end{pmatrix}$$

Applying approximate Green's function takes three steps:

3. Extract original degrees of freedom

$$\begin{pmatrix} * \\ \vdots \\ * \\ v_i \end{pmatrix} \mapsto v_i$$

Challenges for scalability

- Roughly half of the work is in sparse-direct triangular solves (and therefore, dense triangular solves)
- Dense triangular solves with $O(1)$ right-hand sides are, at best, weakly scalable
- Triangular solves with $O(p)$ right-hand sides are scalable, but this requires too much memory
- Parallelism in preconditioner application limited to quasi-2d subdomains!
- Black-box sparse-direct redistributes right-hand sides for solve
- `MUMPS` and `SuperLU_Dist` were not memory scalable, and `WSMP` is not open source, nor does it support large numbers of simultaneous factorizations

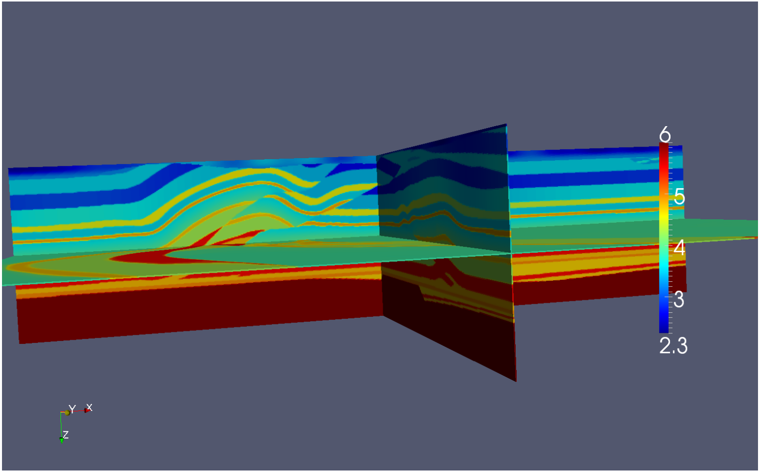
Fighting for scalability

- Wrote custom sparse-direct solver, `Clique`, on top of my distributed dense linear algebra library, `Elemental` (and made sure it was memory scalable!)
- Subdomain sparse-direct factorizations use subtree-to-subcube mappings and 2d front distributions (and redistribute fronts to 1d distribution after factorization)
- Globally reordering global right-hand sides based upon subdomain front distributions avoids communication in sparse-direct subdomain solves
- Dense triangular matrix-vector multiplication has a much lower latency cost than a dense triangular solve...so invert diagonal blocks of distributed fronts after factorization (solve latency drops from $O(m \log p)$ to $O(\log p)$ for $m \times m$ matrix).

Outline

Overthrust model

Velocity in [km/s] of middle *XY*, *XZ*, and *YZ* planes:



Domain is 20 [km] x 20 [km] x 4.65 [km], with low velocity and faults near surface and high velocity near the bottom. Grid is $801 \times 801 \times 187$.

Overthrust convergence

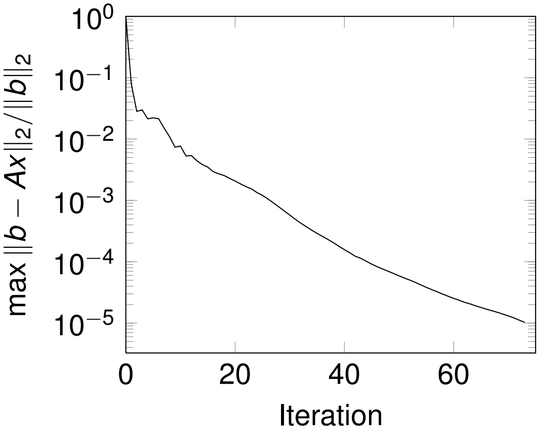


Figure: Convergence of moving PML sweeping preconditioner in GMRES(20) with three near-surface shots for the full Overthrust model with $\omega = 128.63$ [rad/sec] and $\alpha = 2.25\pi$ [rad/sec].

Overthrust runtime on 2048 cores

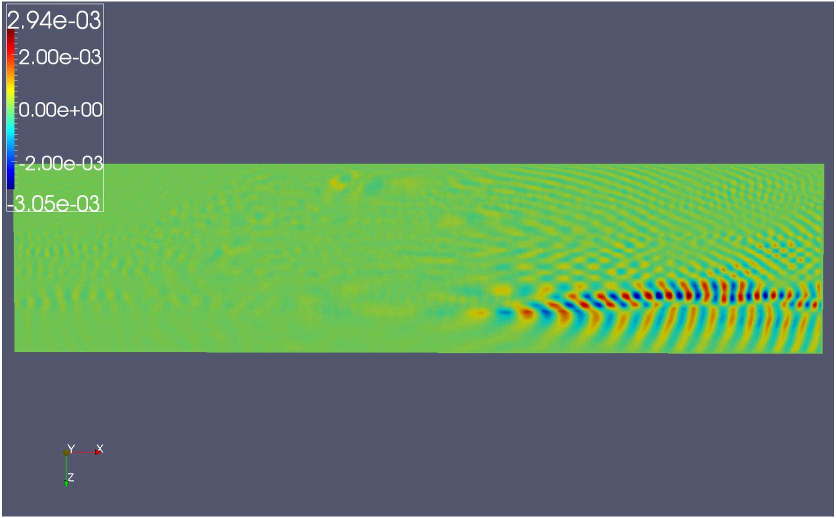
Without distributed diagonal-block inversion:

- Setup time: 250 seconds
- Application time: 90 seconds/iteration
- Total: 72 minutes with 45 iterations (4 digits of accuracy)

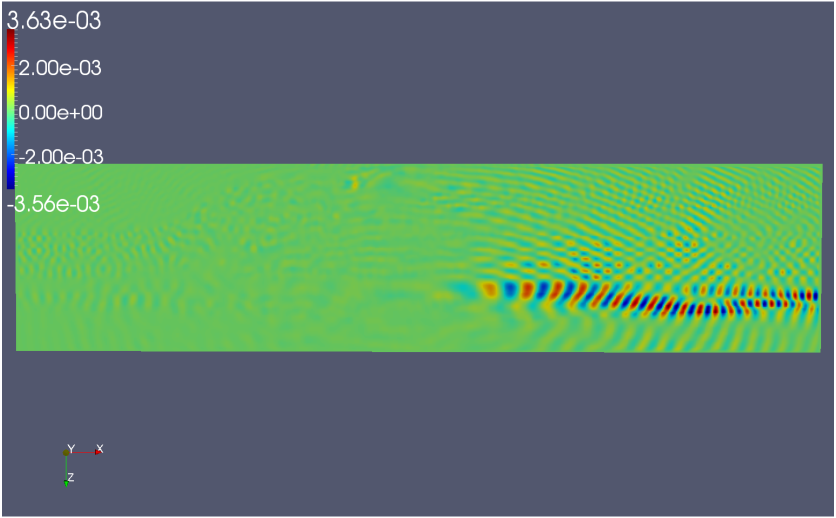
With distributed diagonal-block inversion:

- Setup time: 280 seconds
- Application time: 26 seconds/iteration
- Total: 24 minutes with 45 iterations (4 digits of accuracy)

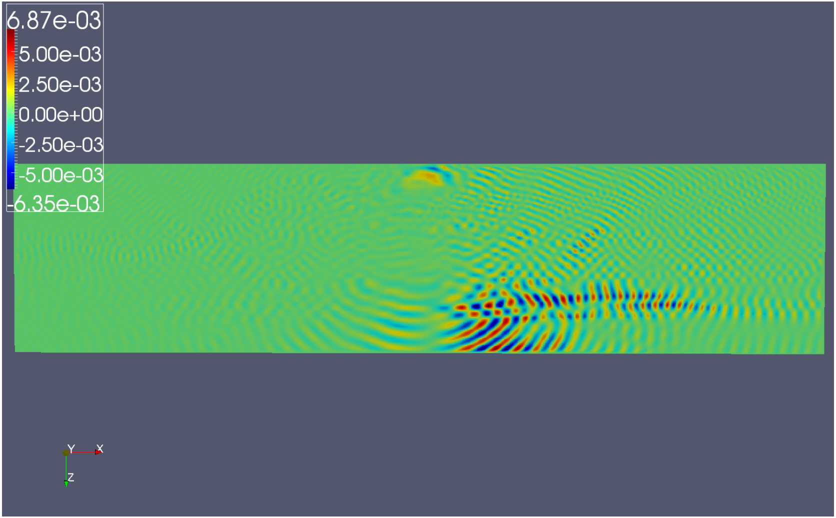
xz-plane solution for top-center shot, $y = 2.025$ [km]



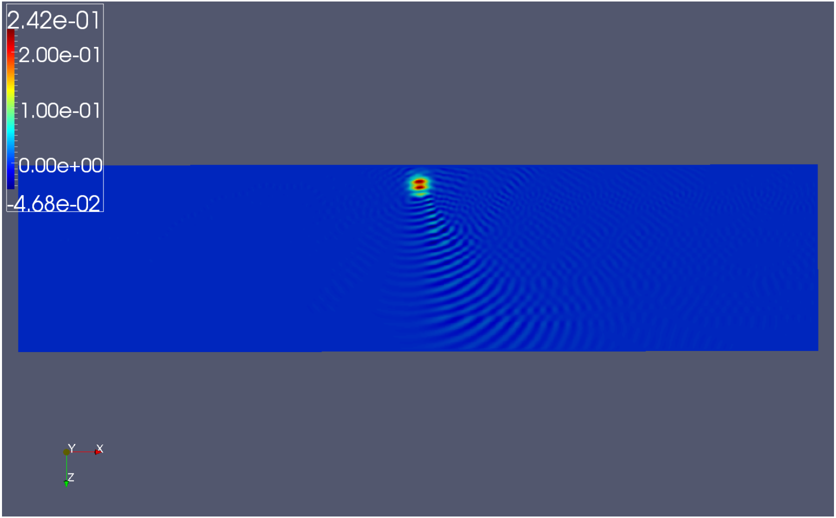
xz-plane solution for top-center shot, $y = 4.025$ [km]



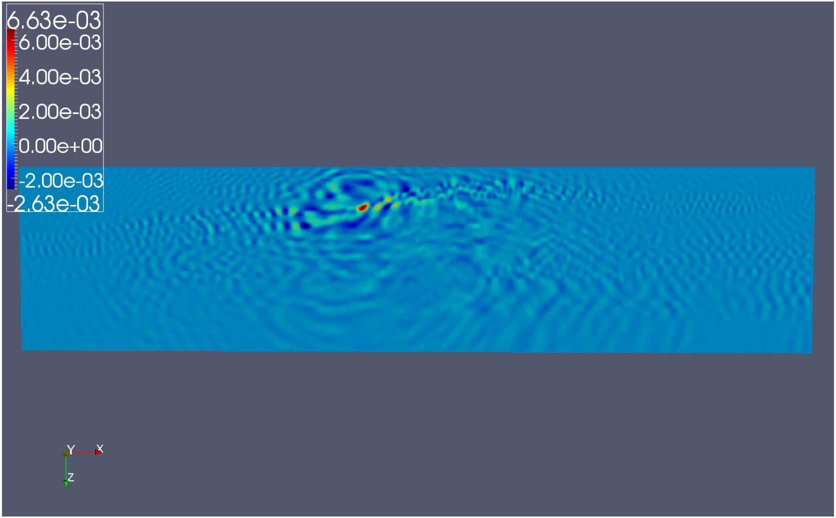
xz-plane solution for top-center shot, $y = 8.025$ [km]



xz-plane solution for top-center shot, $y = 9.825$ [km]



xz-plane solution for top-center shot, $y = 17.025$ [km]



Outline

Conclusions

- The moving PML preconditioner has near-linear complexity and memory usage for realistic models and can be made reasonably scalable
- The setup cost becomes insignificant on large numbers of cores due to better scalability properties
- Inverting diagonal blocks of distributed fronts results in negligible extra work and greatly speeds up preconditioner application

Future work

- Trying larger models on more processors
- Switching to spectral elements
- Trying alternatives to PML (to lower memory usage)
- Block Krylov algorithms
- Adding support for more general geometry
- Adding support for Maxwell and/or elasticity
- Finding cheap estimates of the damping parameter
- Testing efficacy of strongly admissible $\mathcal{H}$-matrix approach
- Performance tuning

Availability

- `Elemental` is available at
  `code.google.com/p/elemental`
- `Clique` will be available in March at
  `bitbucket.com/poulson/clique`
- `PSP` will be available in March at
  `bitbucket.com/poulson/psp`
- `DMHM` sandbox will be available in March at
  `bitbucket.com/poulson/dmhm`