Scalable Chebyshev-Accelerated Schwarz Preconditioning for GPUs

Malachi Phillips, Stefan Kerkemeier, Paul Fischer

March 5, 2021

University of Illinois

Motivation

Packed bed with 146 pebbles, video credit: YuHsiang Lan

Literature

- What's already out there?
    - Fischer et. al [1] introduce p-multigrid preconditioner based on an approximate overlapping Schwarz solve through the fast diagonalization method (FDM).
    - Sundar et. al [2] investigate performance of Chebyshev-accelerated block Jacobi smoothers, even suggesting the usage of Fischer's overlapping Schwarz solver as an approximate inverse.
- What am I contributing in this talk?
    - Combine Fischer's approximate Schwarz solver and Chebyshev smoothers, creating a Chebyshev-accelerated Schwarz scheme.
    - Comparison between Schwarz, Chebyshev-accelerated Jacobi, and Chebyshev-accelerated Schwarz as smoothers in multigrid preconditioner.

Chebyshev Smoother

Algorithm 1: Chebyshev smoother

Input: S, eigenvalue estimates of SA (λ_{min}, λ_{max})

$\theta = \frac{1}{2}(\lambda_{max} + \lambda_{min})$, $\delta = \frac{1}{2}(\lambda_{max} - \lambda_{min})$, $\sigma = \frac{\theta}{\delta}$, $\rho_1 = \frac{1}{\sigma}$

$r = S(b - Ax)$, $d_1 = \frac{1}{\theta} r$, $x_1 = 0$

for $k = 1, \ldots, \mathtt{chebyshevOrder}$ **do**

$$x_{k+1} = x_k + d_k$$

$$r_{k+1} = r_k - SAd_k$$

$$\rho_{k+1} = \frac{1}{2\sigma - \rho_k}$$

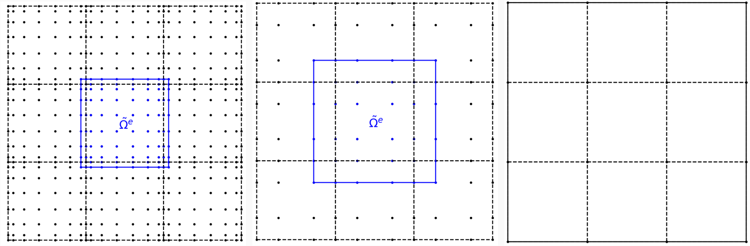
$$d_{k+1} = \rho_{k+1}\rho_k d_k + \frac{2\rho_{k+1}}{\delta} r_{k+1}$$

end

$x_{k+1} = x_k + d_k$

return x_{k+1}

Schwarz Method

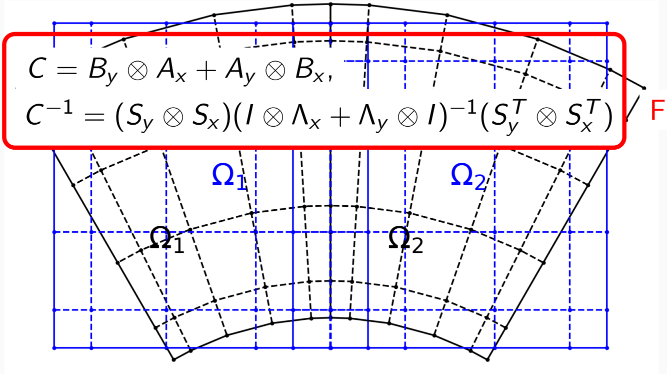


$p = 7$ $p = 3$ $p = 1$

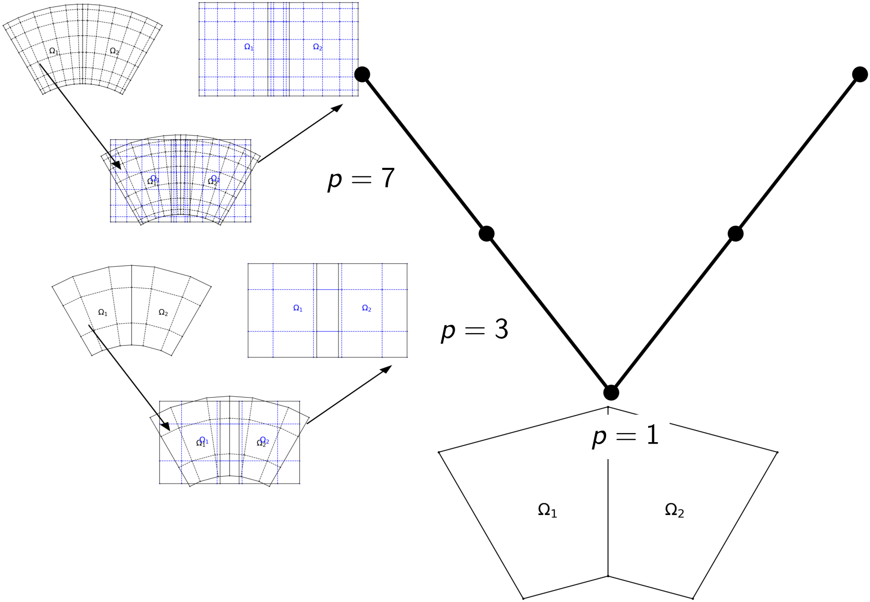
$$C = B_y \otimes A_x + A_y \otimes B_x,$$

$$C^{-1} = (S_y \otimes S_x)(I \otimes \Lambda_x + \Lambda_y \otimes I)^{-1}(S_y^T \otimes S_x^T)$$

FDM



- Extend box-like element by a single node to create subdomains.
- Construct $\bar{A}_i^e$ and $\bar{B}_i^e$.
- Solve eigenvalue problem $\bar{A}_i{}^e s = \lambda \bar{B}_i{}^e s$ to compute S_i^e and Λ_i^e such that $S_i^{e\,T} \bar{A}_i{}^e S_i^e = \Lambda_i^e$ and $S_i^{e\,T} \bar{B}_i^e S_i^e = I$.


Ω_1
Ω_2
$p = 7$
$p = 3$
$p = 1$

Multigrid V-cycle

Algorithm 2: Single pass multigrid V-cycle

$x = x + M(b - Ax)$ // smooth

$r = b - Ax$ // re-evaluate residual

$r_C = P^T r$ // coarsen

$e_C = A_C^{-1} r_C$ // solve coarse grid problem/re-apply V-cycle

$e = Pe_C$ // prolongate

$x = x + e$ // update solution

$x = x + M(b - Ax)$ // post smoothing

For Schwarz smoother:

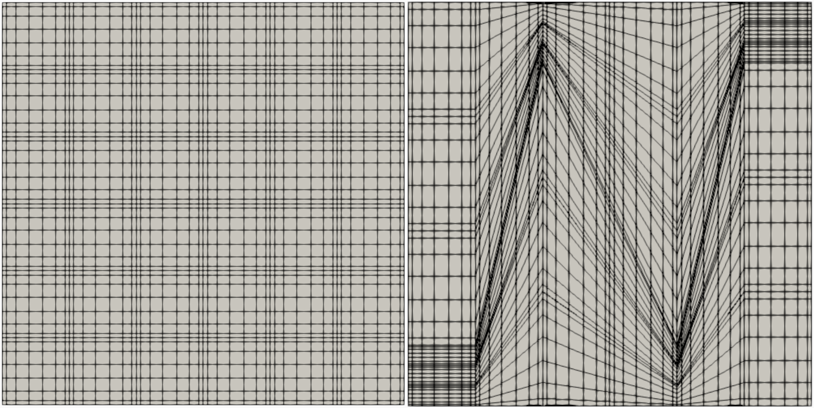
- Residual is not re-evaluated, e.g., $r = b$.
- No post-smoothing is applied.

Methods

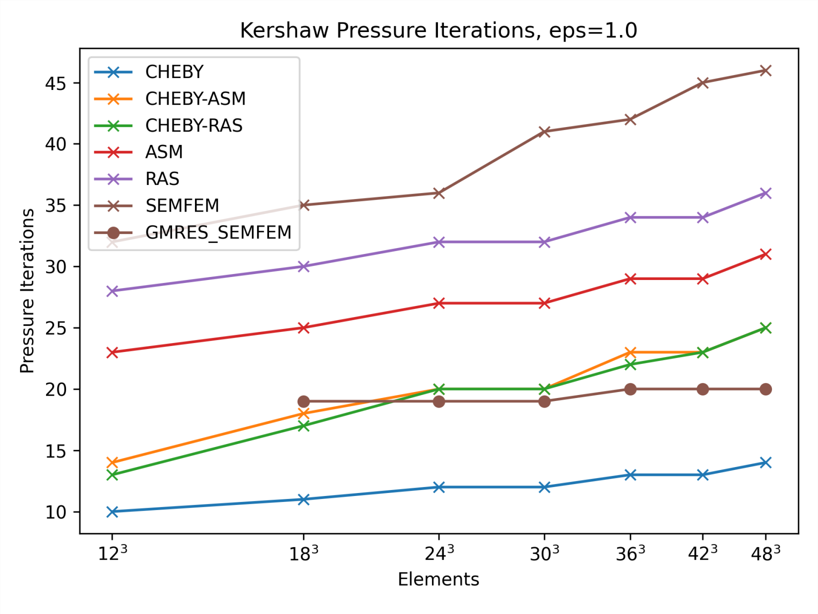
Flexible PCG used as outer solver. Common BoomerAMG settings: V-cycle + HMIS. Preconditioner methods:

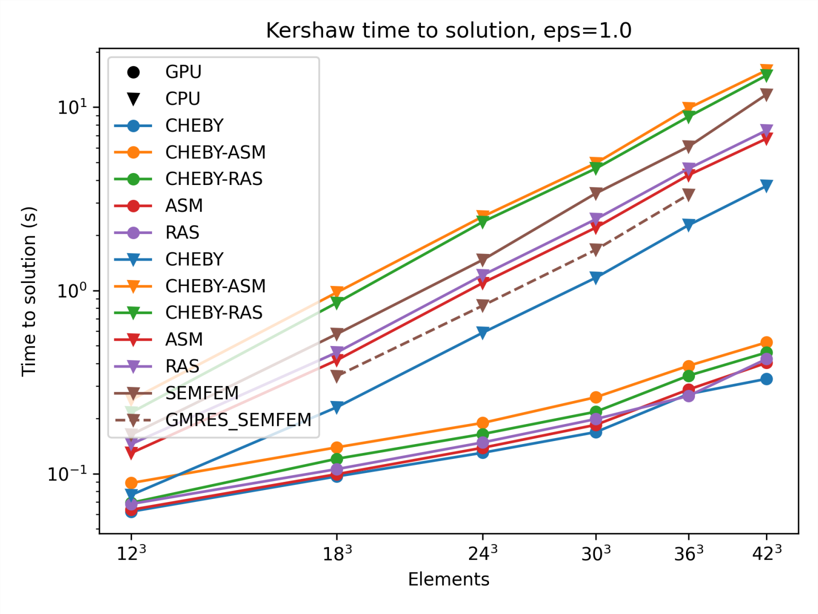
- ASM: Additive Schwarz, single cycle BoomerAMG
- RAS: Restrictive additive Schwarz, single cycle BoomerAMG
- SEMFEM: precondition with low-order FEM, single cycle BoomerAMG
- CHEBY: 2nd order Chebyshev-accelerated Jacobi, two cycle BoomerAMG
- CHEBY+ASM: 1st order Chebyshev-accelerated additive Schwarz, single cycle BoomerAMG
- CHEBY+RAS: 1st order Chebyshev-accelerated restrictive additive Schwarz, single cycle BoomerAMG

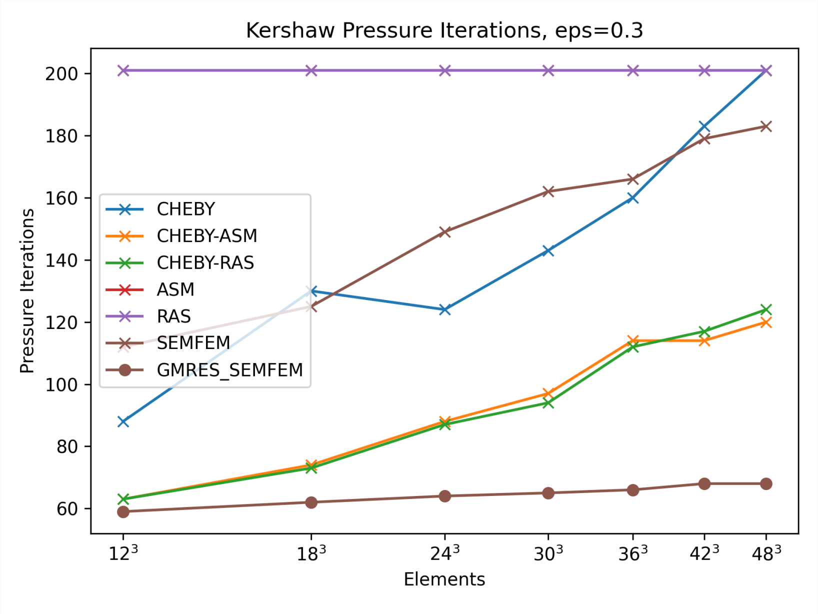
Kershsaw Mesh



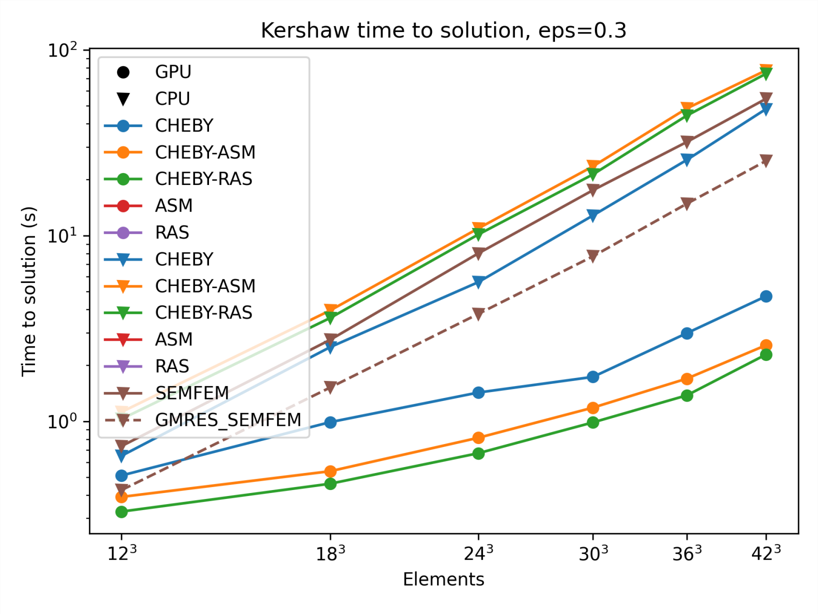
Left: $\epsilon = 1.0$, right: $\epsilon = 0.3$. Vary element count from $E = 12^3$ to $E = 48^3$, all $p = 7$. Reduce relative residual tolerance by 10^{-8}. Single node Summit, CPU only (42 IBM Power9).


Kershaw Pressure Iterations, eps=1.0
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
SEMFEM
GMRES_SEMFEM
Pressure Iterations
10
15
20
25
30
35
40
45
Elements
12^3
18^3
24^3
30^3
36^3
42^3
48^3


Kershaw time to solution, eps=1.0
GPU
CPU
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
SEMFEM
GMRES_SEMFEM
Time to solution (s)
10^1
10^0
10^{-1}
12^3
18^3
24^3
30^3
36^3
42^3
Elements

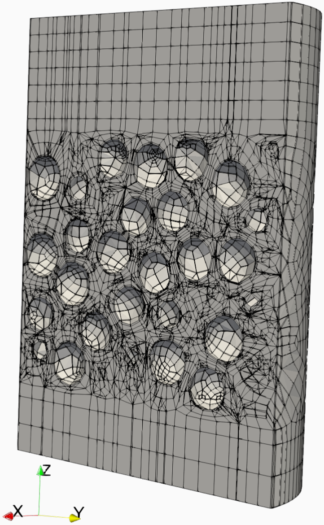


Kershaw Pressure Iterations, eps=0.3
Pressure Iterations
Elements
200
180
160
140
120
100
80
60
12^3
18^3
24^3
30^3
36^3
42^3
48^3
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
SEMFEM
GMRES_SEMFEM



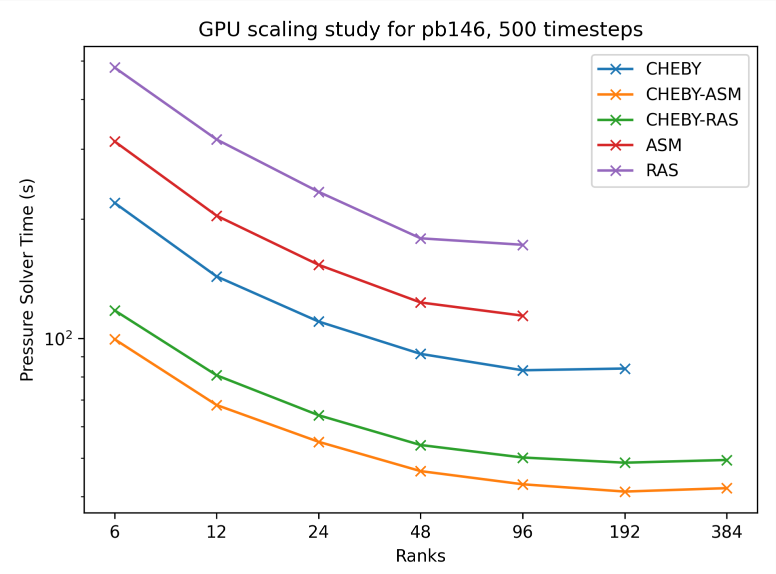
Kershaw time to solution, eps=0.3
Time to solution (s)
10^2
10^1
10^0
GPU
CPU
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
SEMFEM
GMRES_SEMFEM
12^3
18^3
24^3
30^3
36^3
42^3
Elements

146 Pebble Case

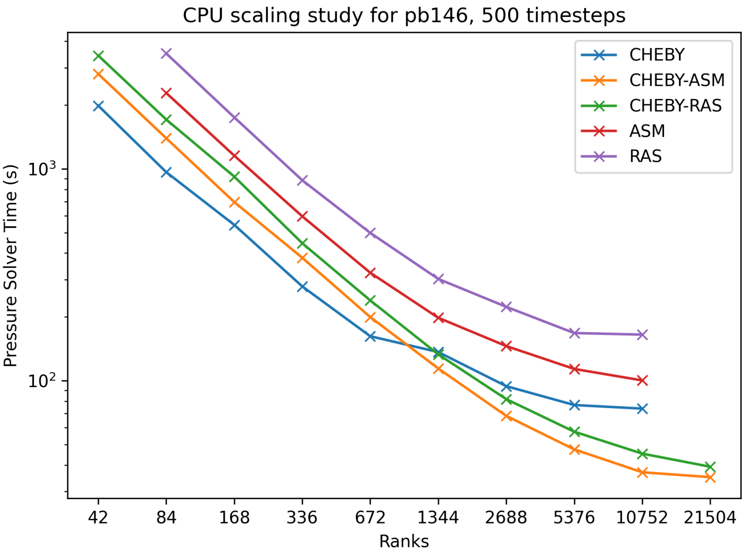


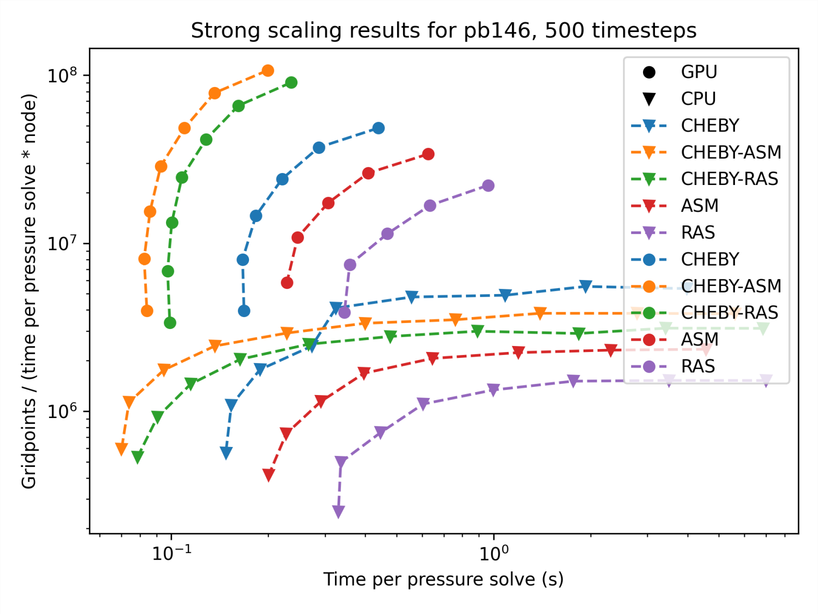
- $(E, p) = (62132, 7)$, pressure tolerance 10^{-4} residual reduction
- Solution projection for pressure initial guess
- Compare CPU and GPU strong scaling on Summit (42 IBM Power9 CPUs + 6 NVIDIA V100 GPUs/node)

GPU Scaling Study



CPU Scaling Study





Strong scaling results for pb146, 500 timesteps
Gridpoints / (time per pressure solve * node)
Time per pressure solve (s)
10^8
10^7
10^6
10^{-1}
10^0
GPU
CPU
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS
CHEBY
CHEBY-ASM
CHEBY-RAS
ASM
RAS

Conclusion

- On GPU, Chebyshev-accelerated Schwarz schemes outperform and outscale both Schwarz and Chebyshev-accelerated Jacobi schemes.
- On CPU, Chebyshev-accelerated Schwarz schemes underperform Chebyshev-accelerated Jacobi schemes for low processor count.
- FDM based Schwarz solves shift the work per depth ratio, leading to better strong scaling (Amdahl's Law)
- Relative preconditioner performance is heavily impacted by the specific case, mesh aspect ratio, etc.

Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

References

P. F. Fischer and J. W. Lottes.
Hybrid schwarz-multigrid methods for the spectral element method: Extensions to navier-stokes.
In *Domain Decomposition Methods in Science and Engineering*, pages 35–49. Springer, 2005.

H. Sundar, G. Stadler, and G. Biros.
Comparison of multigrid algorithms for high-order continuous finite element discretizations.
Numerical Linear Algebra with Applications, 22(4):664–680, 2015.

nekRS

- Fast, scalable open-source Navier Stokes solver
- MPI+X hybrid parallelism supporting CUDA, HIP, OpenCL, and OpenMP
- Visit and Paraview support for data analysis and visualization
- Started as a fork of libParanumal (https://github.com/paranumal/libparanumal)
- Available: https://github.com/nek5000/nekrs