Matrix-free Methods for Summation-by-Parts Finite Difference Operators on GPUs

Alexandre Chen (yiminc@uoregon.edu), Brittany A. Erickson, Jee Whan Choi, University of Oregon
Jeremy E. Kozdon, Naval Postgraduate School (now NextSilicon)

Outline of the Talk

1. SBP operators and the SBP-SAT method
2. Problem description and motivation
3. Matrix-free GPU kernel for SBP-SAT
4. Multi-grid method for the SBP-SAT method
5. Multi-grid preconditioned conjugate gradient (CG) method
6. Conclusions and Future Work

Problem Description

- Earthquake simulation
- Displacement of earth -> this tells you stress on the fault -> stress is combined with the friction to determine the earthquake mechanics (e.g., how fast the fault slides, or the velocity of the movement) -> the displacement/stress/friction happens heterogeneously (e.g., for some regions, frictions increases with the velocity (unlikely to have seismic slip), others frictions decreases with velocity (more likely to have seismic slip))
- You need different resolution and coefficients for different regions
  - Resolution depends on the method you are using the solve the problem
  - Coefficients are physically measured/observed (i.e., field experiments) are input to the simulation
- Summary - very challenging problem to solve and the most computationally expensive part is calculating the displacement from linear elasticity (from solid mechanics) in 3D

Why matrix-free methods and GPU computing?

- We are solving big problems that are **memory intensive**:
  - A problem that you wish to solve may involve a computational domain of hundreds of kms with frictional length scale on the order of millimeters
  - A simplified problem in 3D can easily exceed (1000 * 1000 * 1000 ~ 1 Billion unknowns), >10 GB to store results, >100 GB to store the sparse matrix, > 1TB for matrix factorization
- Why **GPU computing**?
  - GPUs have high bandwidth but limited memory capacity
    - Limited memory capacity - ill suited for large problems using matrix-explicit solutions
  - Therefore, we need a matrix-free method that greatly reduces memory footprint

Why SBP Operators and SBP-SAT Methods

- SBP-SAT summation-by-parts-simultaneous-approximation-terms
- vs. using matrix-free FEM - requires domain transformation (e.g., Fourier) which is more costly than using neighborhood-based (i.e., stencil) in FDM
- Using transformation reduces accuracy because you don’t need to do the transformation which creates weaker formulation of the problem
- Also, less complex to formulate the problem, especially for square-shaped domains
- vs. traditional FDM methods, SBP-SAT numerical stabile when enforcing boundary conditions
-

SBP-SAT Discretization in Matrix-explicit Form

The SBP-SAT discretization is given by

$$-\mathbf{D}_2 \boldsymbol{u} = \boldsymbol{f} + \mathbf{b}^N + \mathbf{b}^S + \mathbf{b}^W + \mathbf{b}^E, \quad (1)$$

where $\mathbf{D}_2 = \left(\mathbf{I} \otimes \mathbf{D}_{xx}\right) + \left(\mathbf{D}_{yy} \otimes \mathbf{I}\right)$

$$\mathbf{D}_{xx} = \frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & 1 & -2 & 1 \end{bmatrix}$$

the red number resembles traditional Laplacian operator in the domain interior.

is the discrete Laplacian operator in 2D and $\mathbf{u}$ is the grid function approximating the solution, formed as a stacked vector of vectors.

The SAT terms $\mathbf{b}^N, \mathbf{b}^S, \mathbf{b}^W, \mathbf{b}^E$ enforce all boundary conditions weakly. To illustrate the structure of these vectors, the SAT term enforcing Dirichlet condition on the west boundary is given by

$$\mathbf{b}^W = \alpha \left(\mathbf{H}^{-1} \otimes \mathbf{I}\right)(\mathbf{E}_\mathrm{W}\mathbf{u} - \mathbf{e}_\mathrm{W}^T\mathbf{g}_\mathrm{W}) - \left(\mathbf{H}^{-1}\mathbf{e_0}\mathbf{d_0}^T \otimes \mathbf{I}\right)(\mathbf{E}_\mathrm{W}\mathbf{u} - \mathbf{e}_\mathrm{W}^T\mathbf{g}_\mathrm{W})$$

The Linear system (1) is rendered SPD by multiplying of $\mathbf{H} \otimes \mathbf{H}$

If we move all the “u” to the left-hand-side, then you form the linear system Ax=b

(**Erickson, B. A.** and Dunham, E. M. (2014))

Coordinate Transformation on GPUs

- For complex domain or variable coefficients, SBP-SAT methods can be combined with coordinate transformation. (**Kozdon** et al. 2020)
  - Coordinate transformation requires only local information in the Jacobian matrices, similar to the stencil computation for Laplacian, which can be calculated matrix-free.
  - Jacobian matrices are stored as input data for GPU kernels (current implementation), but can also be evaluated within GPU kernels to save memory I/O (future work).

Problem Description

We are solving the 2D Poisson equation motivated by large scale earthquake cycle simulations.

Simplification: Anti-plane strain, coordinate transformation for the domain (**Kozdon** et al. 2020)

$$
\begin{aligned}
-\bar{\nabla}\cdot\left(\mathbf{c}\,\bar{\nabla}u\right) &= Jf, && \text{for } (r,s)\in\bar{\Omega},\\
u &= g_1, && \text{face 1},\\
u &= g_2, && \text{face 2},\\
\hat{\boldsymbol{n}}^3\cdot\mathbf{c}\,\bar{\nabla}u &= S_J^3 g_3, && \text{face 3},\\
\hat{\boldsymbol{n}}^4\cdot\mathbf{c}\,\bar{\nabla}u &= S_J^4 g_4, && \text{face 4},
\end{aligned}
$$

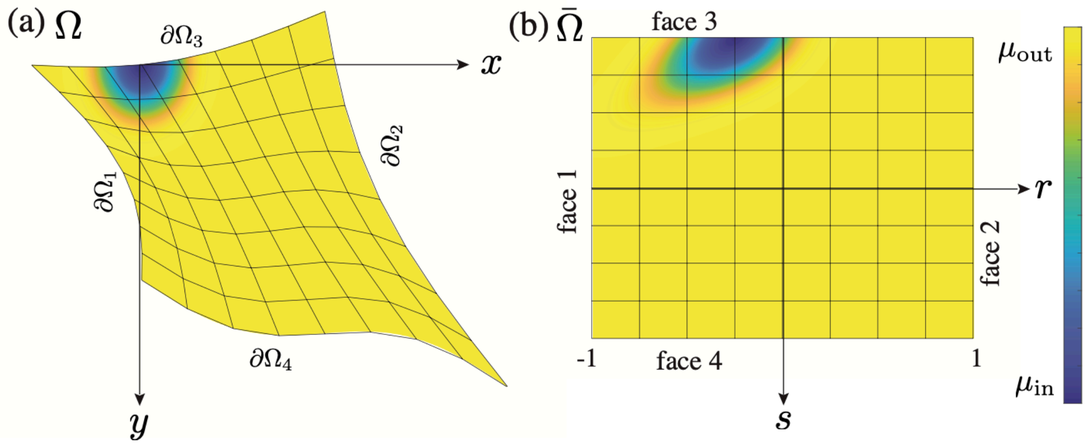
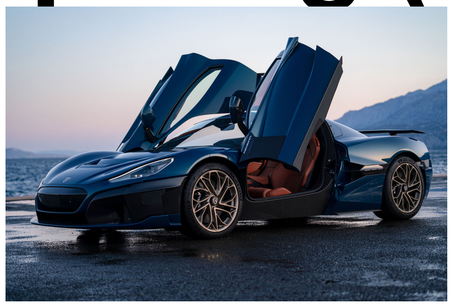


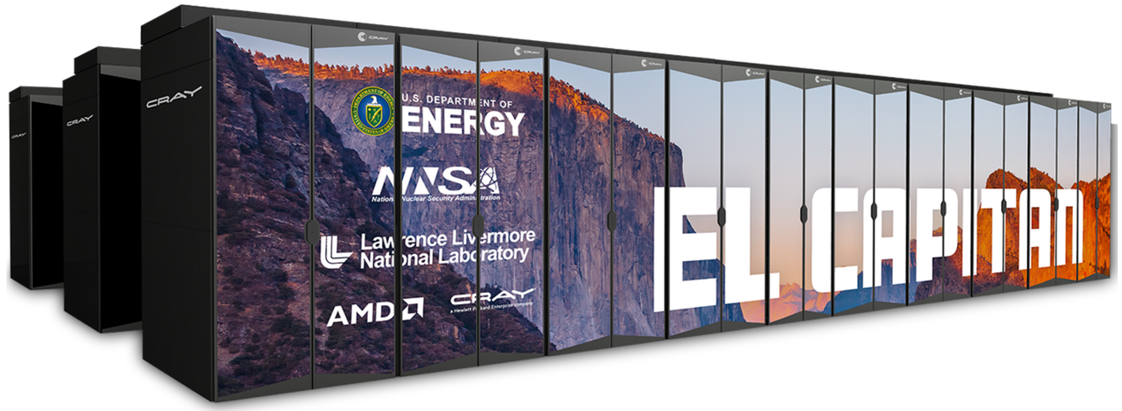
Figure 1. (a) Geometrically complex physical domain Ω with material stiffness that increases from μ_{in} within a shallow, ellipsoidal sedimentary basin, to stiffer host rock given by μ_{out}. (b) Ω is transformed to the regular, square domain $\bar{\Omega}$ via conformal mapping.

General Purpose GPU Computing (GPGPU)



- GPU and CPU are optimized for different purposes
  - CPU: more versatile, **low latency**, small number of faster cores
  - GPU: less versatile, high latency but **high throughput**, large number of slower cores in parallel
  - Things are starting to change: AMD's new 128-core EPYC
- Parallel Scheme: Single Instruction Multiple Data (SIMD)

- Toolkits for GPGPU:
  - CUDA (NVIDIA) is the mainstream platform
  - OpenCL (CPUs + GPUs, different vendors)
  - ROCm (AMD), oneAPI (Intel)



A Survey of General-Purpose Computation on Graphics Hardware, John D. Owens et. al

The proposed exa-scale super computer El Capitan is built using AMD CPUs and GPUs.

Matrix-free stencils for SBP-SAT method

- Idea: split the domain write simpler kernels for interior points

Matrix-free GPU kernel Action of matrix-free A for interior nodes

function `mfA!`(odata, idata, $c_{rr}, c_{rs}, c_{ss}, h_r, h_s$)

i, j =get_global_thread_IDs()

$g = (i-1) * (N+1) + j$ ▷ compute global index

if $2 \leq i, j \leq N$ ▷ interior nodes

odata[g] = (*hs/hr*)(- (0.5c_{rr}[g-1] + 0.5c_{rr}[g])idata[g-1] +
+ (0.5c_{rr}[g-1] + c_{rr}[g] - 0.5c_{rr}[g+1])idata[g] +
- (0.5c_{rr}[g] + 0.5c_{rr}[g+1])idata[g+1]) + ▷ compute M_{rr} stencil

+ 0.5c_{rs}[g-1](-0.5idata[g-N-2] + 0.5idata[g+N]) +
- 0.5c_{rs}[g+1](-0.5idata[g-N] + 0.5idata[g+N+1]) + ▷ compute M_{rs} stencil

+ 0.5c_{rs}[g-N-1](-0.5idata[g-N-2] + 0.5idata[g-N]) +
- 0.5c_{rs}[g+N+1](-0.5idata[g-N] + 0.5idata[g+N+2]) + ▷ compute M_{sr} stencil

- (0.5c_{ss}[g-N-1] + 0.5c_{ss}[g])idata[g-N-1] +
+ (0.5c_{ss}[g-N-1] + c_{ss}[g] + 0.5c_{ss}[g+N+1])idata[g] -
- (0.5c_{ss}[g] + 0.5c_{ss}[g+N+1])idata[g+N+1])) ▷ compute M_{ss} stencil

end

... ▷ boundary nodes, e.g. code block 1

return *nothing*

end

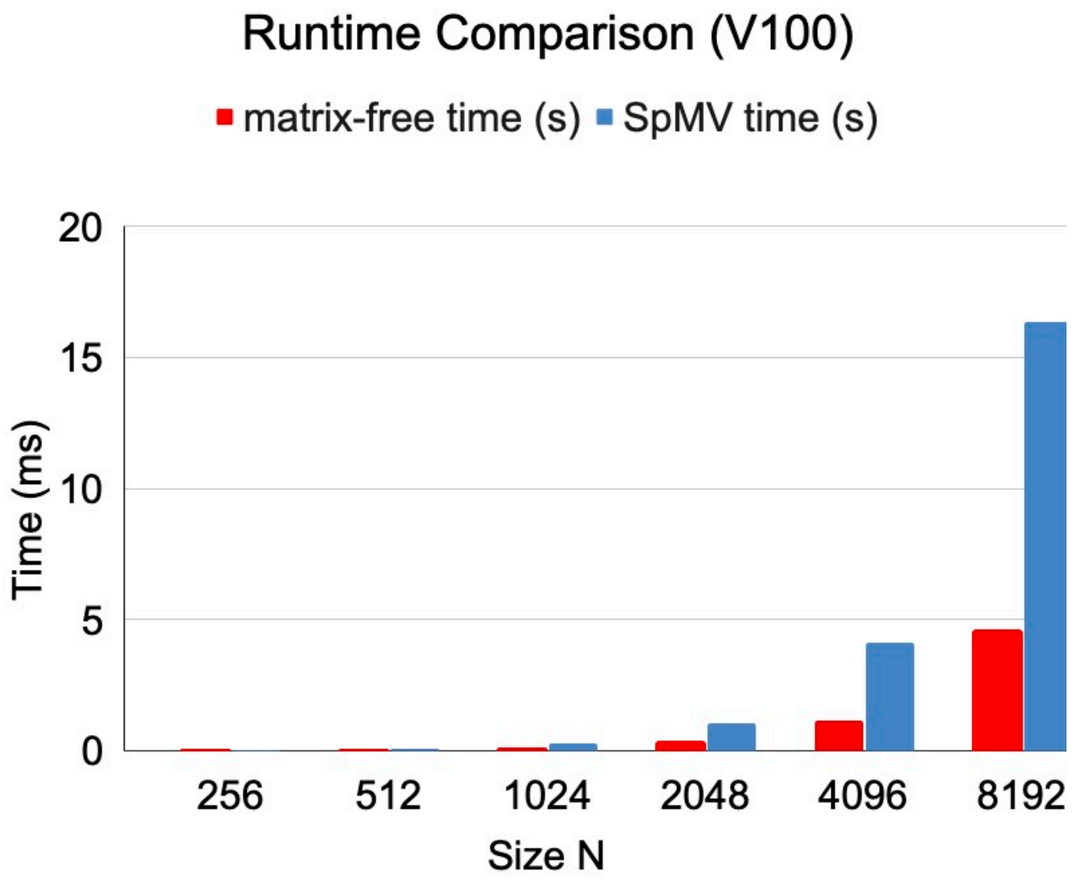
Matrix-free stencils for SBP-SAT method

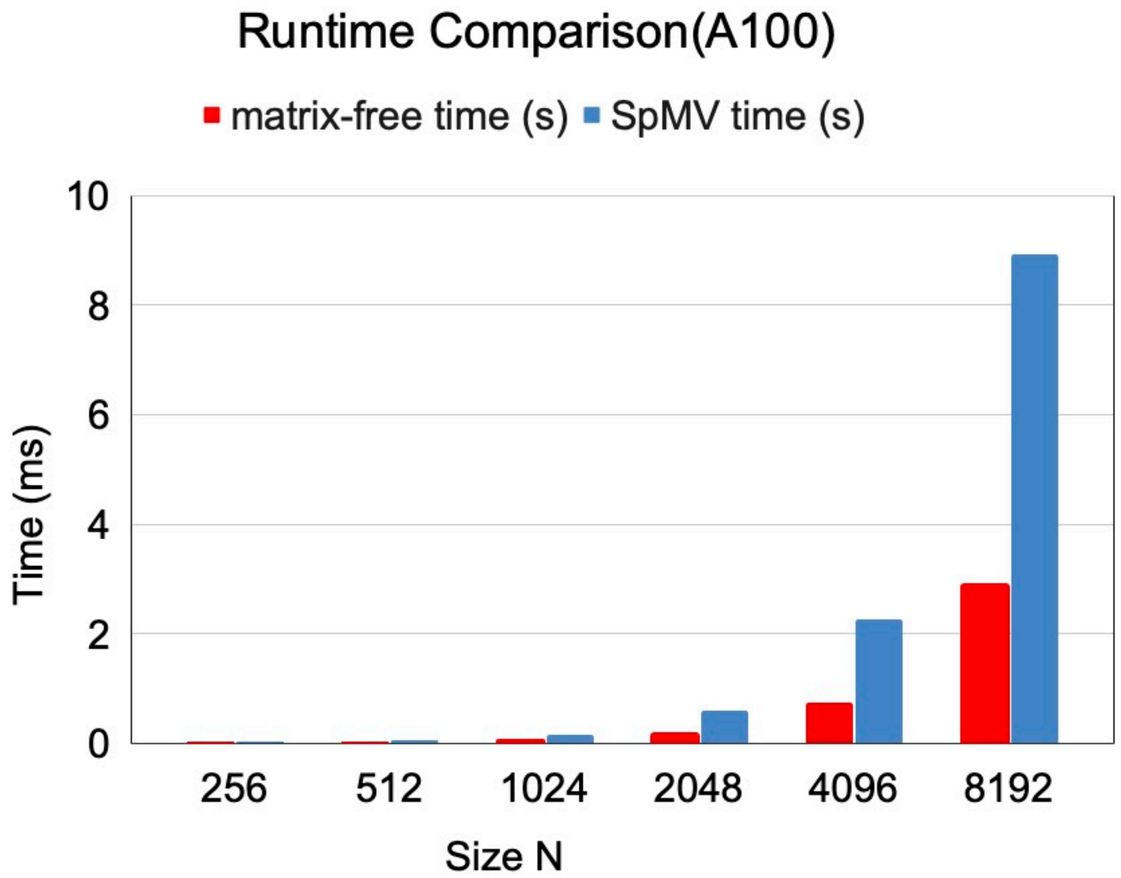
- Idea: Write separate calculations for domain boundaries to avoid race condition

Matrix-free GPU kernel Action of matrix-free A for west boundary (face 1).

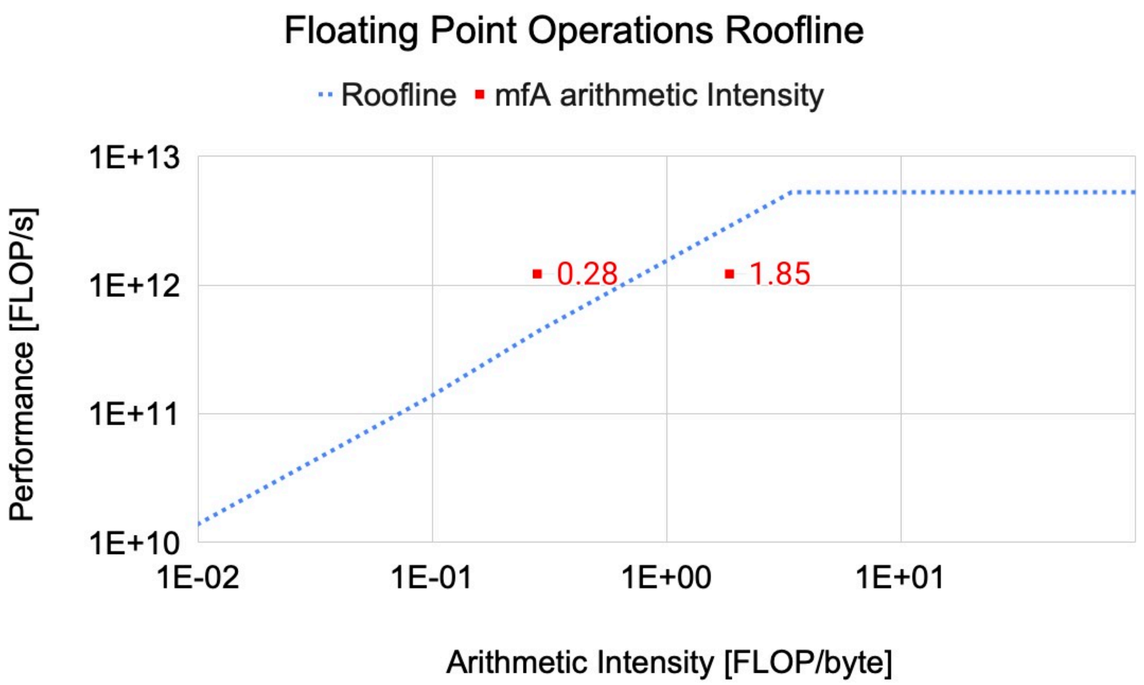
if $2 \leq i \leq N$ and $j = 1$ ▷ interior west nodes
  odata[g] = $\left(M_{rr}^{int} + M_{rs}^{int} + M_{sr}^{int} + M_{sr}^{int} + C_1^{int}\right)$ (idata) ▷ apply boundary M and C stencils
  odata[g+1] = C_1^{int}(idata) ▷ apply interior C stencil
  odata[g+2] = C_1^{int}(idata) ▷ apply interior C stencil
end
if $i = 1$ and $j = 1$ ▷ southwest corner node
  odata[g] = $(M_{rr}^{sw} + M_{rs}^{sw} + C_1^{sw})$ (idata) ▷ apply southwest partial M and C stencils
  odata[g+1] = C_1^{sw}(idata) ▷ apply southwest interior boundary C stencil
  odata[g+2] = C_1^{sw}(idata) ▷ apply southwest interior boundary C stencil
end
if $i = N + 1$ and $j = 1$ ▷ northwest corner node
  odata[g] = $(M_{rr}^{nw} + M_{rs}^{nw} + C^{nw})$ (idata) ▷ apply northwest partial M and C stencils
  odata[g+1] = C^{nw}(idata) ▷ apply northwest interior boundary C stencil
  odata[g+2] = C^{nw}(idata) ▷ apply northwest interior boundary C stencil
end

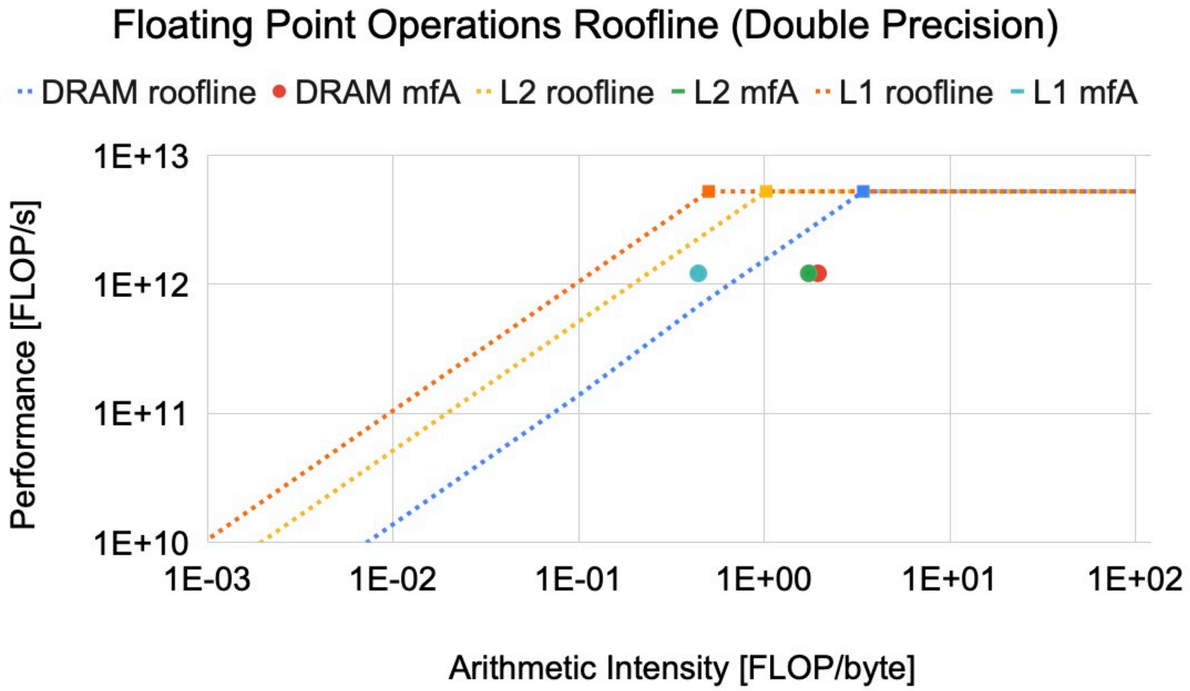
Matrix-free vs SpMV





Roofline model





Multigrid Method

Two-Grid Correction Scheme

$$\mathbf{v}^h \leftarrow MG(\mathbf{v}^h, \mathbf{f}^h).$$

- Relax ν_1 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ on Ω^h with initial guess $\mathbf{v}^h$.
- Compute the fine-grid residual $\mathbf{r}^h = \mathbf{f}^h - A^h\mathbf{v}^h$ and restrict it to the coarse grid by $\mathbf{r}^{2h} = I_h^{2h} r^h$.
- Solve $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ on Ω^{2h}.
- Interpolate the coarse-grid error to the fine grid by $\mathbf{e}^h = I_{2h}^h \mathbf{e}^{2h}$ and correct the fine-grid approximation by $\mathbf{v}^h \leftarrow \mathbf{v}^h + \mathbf{e}^h$.
- Relax ν_2 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ on Ω^h with initial guess $\mathbf{v}^h$.

A Two-level Multigrid Scheme from A Multigrid Tutorial 2nd Edition (Briggs, 2000)

Geometric or Algebraic Multigrid ?

How to form coarse grid operator A^{2h}

- Geometric Multigrid: Forming A^{2h} directly on coarser grid similarly to A^h-> SBP-SAT compatibility issue?
- Algebraic Multigrid / Galerkin Coarsening $A^{2h} = I_h^{2h} A^h I_{2h}^h$
- SBP-preserving interpolation operators:
  $I_h^{2h} = (H_{2h} \otimes H_{2h})^{-1} \left(I_{2h}^h\right)^T (H_h \otimes H_h)$, where I_{2h}^h is the standard prolongation operator (**Ruggiu, Andrea A** et al. 2018, **Briggs** et al. 2000)

- We choose **Geometric Multigrid**
  - Galerkin Coarsening or Algebraic Multigrid can assemble coarse grid automatically (“plug-in” solver), however
    - Writing different kernels for each grid level + compilation cost for matrix-free GPU kernels
    - Increased memory requirement if matrix-explicit methods are used
  - Interpolation operators in Geometric Multigrid
    - SBP-SAT method is rendered SPD by the multiplication of the H matrix that contains grid information, the residual for each level is “scaled” residual
    - When interpolating the residuals, the SBP-preserving restriction operators need to be further modified with grid information. $\tilde{I}_h^{2h} = (H_{2h} \otimes H_{2h}) I_h^{2h} (H_h \otimes H_h)^{-1}$, which is essentially letting $\tilde{I}_h^{2h} = \left(I_{2h}^h\right)^T$
    - us
- Using **Richardson Iteration** $x^{k+1} = x^k + \omega_k(b - Ax^k)$ and optimal $\omega_k = \dfrac{2}{\lambda_{max} + \lambda_{min}}$ for each level. To determine for eigenvalues for very large grid, we **extrapolate** eigenvalues from smaller grids
- 5 relaxation steps for pre-smoothing, post-smoothing and smoothing on the coarsest grid, Multi-level multigrid to avoid the cost of direct solve on CPU

Multigrid Preconditioned CG

Table 3: Iterations and time to converge for $N = 2^{10}$ using 1 smoothing step in PETSc PAMGCG with V cycle (first three rows) vs. our MGCG using Richardson's iteration as smoother (last row)

mg_levels_ksp_type	mg_levels_pc_type	iters	time
chebyshev	sor	18	4.105 s
	jacobi	22	3.382 s
	bjacobi	17	3.945 s
richardson	sor	18	3.581 s
	jacobi	49	3.729 s
	bjacobi	16	3.729 s
cg	sor	17	4.081 s
	jacobi	23	3.849 s
	bjacobi	16	3.971 s
richardson	none	11	0.086 s

Table 4: Iterations and time to converge for $N = 2^{10}$ using 5 smoothing steps in PETSc PAMGCG with V cycle (first three rows) vs. our MGCG using Richardson's iteration as smoother (last row)

mg_levels_ksp_type	mg_levels_pc_type	iters	time
chebyshev	sor	10	10.76 s
	jacobi	14	10.20 s
	bjacobi	9	10.58 s
richardson	sor	9	10.13 s
	jacobi	DV	9.24 s
	bjacobi	8	10.28 s
cg	sor	9	10.47 s
	jacobi	13	10.54 s
	bjacobi	8	10.45 s
richardson	none	8	0.069s

Table 7: Time to perform a direct solve after LU factorization on CPUs vs PCG on GPUs. We report time in seconds and iterations to converge. For AmgX, we report setup + solve time. For our MGCG, setup time is negligible. "ns" is short for the number of smoothing steps. GPU results are tested on A100.

N	Direct Solve	AmgX (ns = 1)	AmgX (ns = 5)	SpMV-MGCG (ns = 5)	MF-MGCG (ns = 5)
2^{10}	0.912 s	(0.0319 s + 0.0243 s) / 25	(0.0321 s + 0.0435 s) / 17	7.019E-2 s / 8	2.851E-2 s / 8
2^{11}	6.007 s	(0.086 s + 0.161 s) / 55	(0.086 s + 0.311 s) / 38	0.158 s / 7	0.0605 s / 7
2^{12}	22.382 s	(0.310 s + 0.235 s) / 24	(0.323 s + 0.488 s) / 15	0.564 s / 7	0.207 s / 7
2^{13}	134.697 s	(1.334 s + 1.643 s) / 24	(1.217 s + 1.865 s) / 16	5.028 s / 7	0.865 s / 7

Conclusions

- Matrix-Free GPU kernels for SBP operators not only save memory, but can be much faster than SpMV kernels
- We adapted SBP-preserving interpolation operators to make it compatible with the SBP-SAT scheme in geometric multigrid.
- Geometric multigrid can be a very effective preconditioner for the SBP-SAT method. The multigrid algorithm can also be implemented matrix-free.
- MGCG outperforms direct solve for large problem with much lower memory requirement

Can we optimize the performance even more?

- Mixed precision strategy (made easier in matrix-free):
  - GPUs have much higher performance with single precisions. Using single precision can easily make GPU kernels run (at least) 2 times faster.
  - If single precision is used only for the MG preconditioner, the increase in overall PCG iteration counts is negligible.
- Using higher-order matrix-free smoothers (second-order Richardson, Chebyshev iteration)
- Replacing V-cycle multigrid with F-cycle multigrid to reduce the cost
- Aggressive coarsening: interpolation for more than 1-level gives additional reduction in complexity and increased scalability (e.g. BoomerAMG) (+ Machine Learning)
- When solving systems of equations like earthquake cycle simulation, using interpolated initial guesses reduce iteration counts significantly (+ Machine Learning)