Novel Solver Algorithms for Nearly Singular Linear Systems Arising in Combustion Modelling

Presentation originally given at 2022 SIAM Conference on Parallel Processing for Scientific Computing, Wednesday, February 23, 2022

Paul Mullowney, NREL and Stephen Thomas, AMD
Arielle K. Carr, Lehigh University
Katarzyna Swirydowicz, PNNL
Marcus Day and Lucas Esclapez, NREL

Abstract

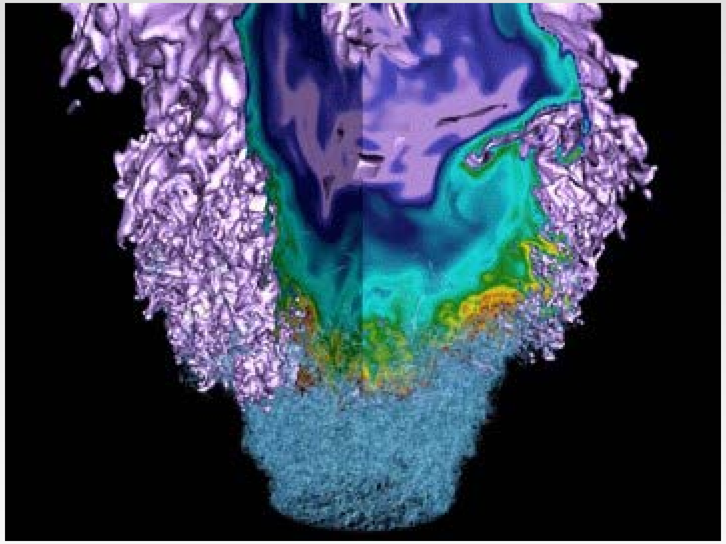
Direct Numerical Simulations of realistic combustion devices are extremely challenging due to the wide separation of scales in the simulation, for example an internal combustion (IC) engine chamber, and the flame thickness of a high-pressure flame. The PeleLMeX solver uses adaptive mesh refinement (AMR) to evolve multi-species reacting flows in the low Mach number limit at the Exascale and relies on an embedded boundary (EB) approach to represent complex geometries. In that framework, the EB geometries often give rise to very small cut-cells along the boundary, which translate into extreme ill-conditioning of the pressure-projection, with eigenvalues that span 15-16 orders of magnitude. In this talk, we focus on the case of a typical IC piston bowl geometry for which we present on a novel approach towards solving these nearly singular linear systems with ILU-based, C-AMG smoothers on massively parallel architectures. In particular, we use scaling and equilibration algorithms to handle the non-normality of the upper triangular factors. This enables us to approximate the highly sequential triangular solve algorithm, embedded in the AMG smoothing-solve phase, with Jacobi iterations. This approximation can be written as a convergent Neumann series whose terms are composed of highly parallel sparse matrix vector multiplications. The result is an algorithm that substantially decreases setup and solve time, compared to state-of-the-art, for these challenging linear systems.

Outline

- Combustion Modelling with PeleLMeX
- ILU Smoothers for C-AMG
    - Jacobi Iterations for approximating Sparse Triangular Solve
    - Handling Non-Normal Triangular Factors
- Computational Results
    - PeleLMeX
    - Deeper dive into performance gains
- Conclusions and Future Work

PeleLMeX: a low Mach number reactive flows solver at the ExaScale

- Key features:
  - Solve the multi-species Navier-Stokes equations in the low-Mach number limit
  - Detailed chemistry and transport based on standard CHEMKIN format
  - Spectral deferred correction (SDC) to couple slow and fast physics-chemical processes while enforcing the low Mach velocity divergence constraint
  - Embedded boundary method to include complex geometry
  - AMR supported by the open-source AMReX library
  - Ported to the next generation of GPU-accelerated supercomputers



Turbulent low swirl burner hydrogen-air flame

Available on github *https://amrex-combustion.github.io/PeleLMeX/*

PeleLMeX: a low Mach number reactive flows solver at the ExaScale

- A linear solver intensive algorithm:
  - In the low Mach number limit, the velocity divergence is constrained by the diffusion/reaction processes occurring in the gaseous mixture
  - A fractional-step approach is employed:

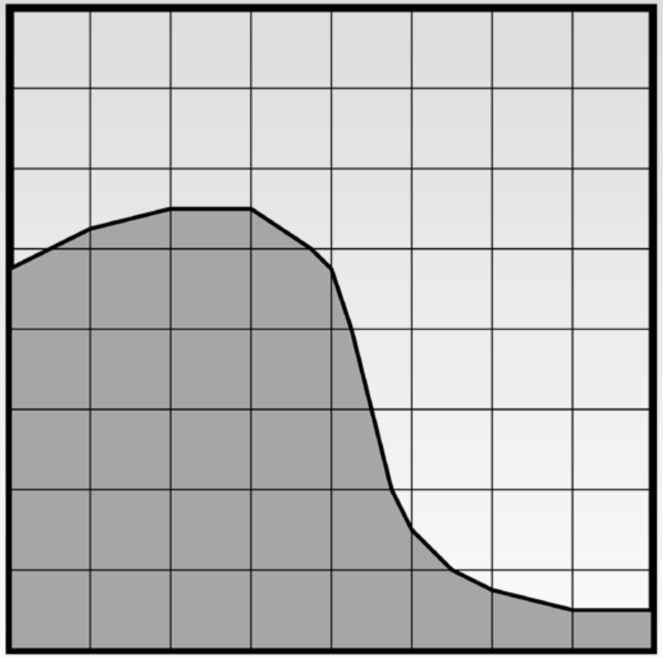
1. t^{n+1*} velocity U^{n+1*} is predicted using the momentum equation
2. A projection step is then performed to enforce the divergence constraint, solving the linear system:

$$\overbrace{\mathcal{L}^p}^{\text{FE approx. of } \nabla \cdot \frac{1}{\rho}\nabla} \phi = \underbrace{\mathcal{D}}_{\text{nodal divergence of cell-centered data}} \left(U^{n+1*} + \Delta t \overbrace{\mathcal{G}}^{\substack{\text{cell centered gradient} \\ \text{of nodal data}}} \underset{\text{dynamic pressure}}{\pi^{n-1/2}}\right) - \overset{\text{divergence constraint}}{S^{n+1}}$$

3. Update velocity: $U^{n+1} = U^{n+1*} - \dfrac{1}{\rho}\mathcal{G}\phi$

PeleLMeX: a low Mach number reactive flows solver at the ExaScale

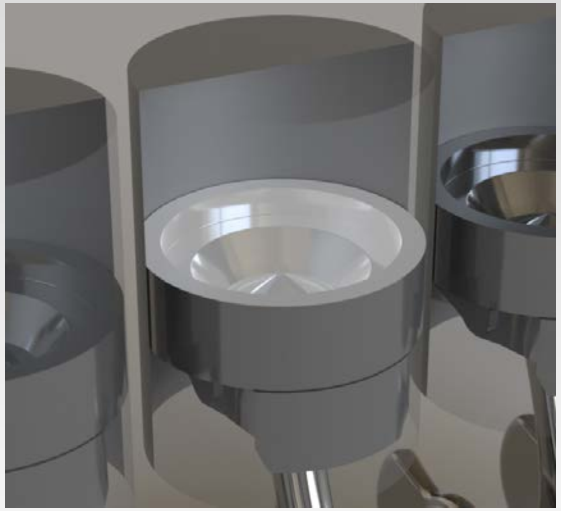
- Complex geometries are represented using an embedded geometry (EB) approach:
  - The (uniform) rectangular mesh is cut by the irregular shape of the computational domain
  - This technique can produce arbitrarily small cut cells in the domain, possibly raising robustness and stability issues for classical finite volume scheme (CFL type conditions)
  - Additionally, the modifications of the linear operator stencils can lead to particularly ill-conditioned systems



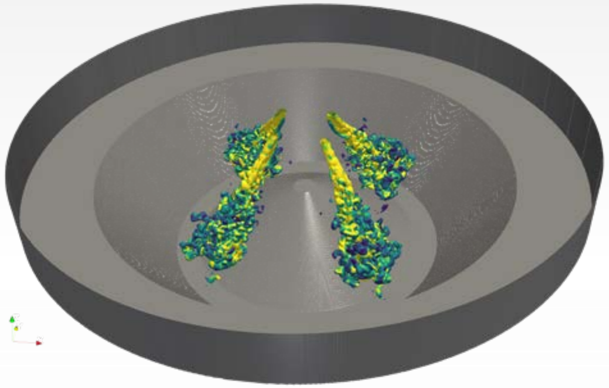
Schematic of the EB approach used in AMReX

Challenge problem: reactivity-controlled compression ignition

- Performing direct numerical simulation of multi-injection ignition behavior in representative engine conditions:
  - At scale, on ½ of the Frontier ExaScale platform (>4000 MI250X AMD GPUs)
  - Typical engine conditions: complex multi-species mixture (~50 chemical species, ~300 reactions) at high pressure and temperature



CAD of a typical ICE configuration



4 turbulent $C_{12}H_{26}$ jets into the piston-bowl geometry (PeleLMeX)

- Including a realistic representation of an internal combustion engine (ICE) piston bowl geometry
- Spatial resolution requirements leads ~50B DoF, concentrated near the highly reactive areas

ILU(k), ILUT Error Smoothers in C-AMG

- Consider the solution of the linear system $Ax = b$, with A sparse, symmetric, highly ill-conditioned.
- When solving using C-AMG, the incomplete LU factorization can be employed as a smoother.
- Direct sparse triangular solves are comparatively slow on GPUs.
- We instead consider an iterative approach [1,2] with scaled factors to solve the triangular systems at each of the (pre/post) error smoothing steps.
- With A ill-conditioned, we may consider equilibration techniques, but can result in highly nonnormal factors.

[1] Chow E, Patel A. “Fine-Grained Parallel Incomplete LU Factorization” SIAM SISC 2015.
[2] Anzt H, Chow E, Dongarra J. “Iterative sparse triangular solves for preconditioning” European conference on parallel processing 2015.

Iterative Solution of Triangular Solves

- Jacobi iteration for solving $Ax = b$, define the iteration matrix $G = I \ - D^{-1}A$ and iterate as

$$x^{(k+1)} = Gx^{(k)} + D^{-1}b \qquad (1)$$

- For the upper triangular solve, we compute $Ux = D^{-1}b$, $U = I + \ U_s$, where U_s is strictly upper triangular (analogous solve for $L = I + L_s$).

- Jacobi diverges for nonnormal matrices, such as upper and lower triangular matrices.

- Note also that we obtain Neumann series since, for $f \ = D^{-1}b$, (1) can be rewritten as

$$\begin{aligned} x^{(k+1)} &= f \ - \ U_s f + U_s^2 f - \cdots + (-1)^k U_s^k f \\ &= (I \ - \ U_s + U_s^2 - \cdots + \left(-1)^k U_s^k\right) f \\ &= (I - U_s)^{-1} f \end{aligned}$$

Mitigating Nonnormality in the Jacobi Iteration

- At each level of the V-cycle we can mitigate divergence using row and column scaling on L and U directly.[1]
- Ruiz scaling [3] or simply the incomplete LDU factorization (row scaling) can be applied.
- In doing so, we minimize the *departure from normality* of the triangular factor.

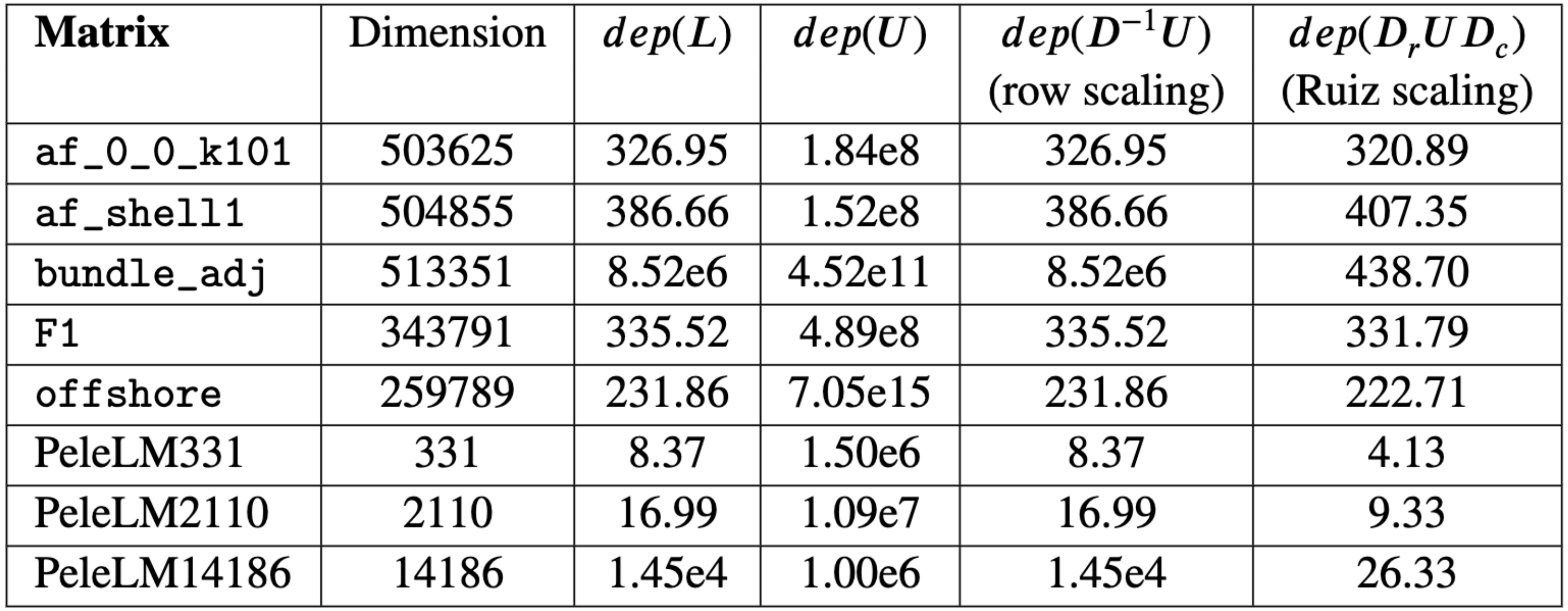
$$dep(B) = \sqrt{||B||_F^2 - ||D||_F^2},$$

where D is the diagonal matrix containing the eigenvalues of B.

[1] For the applications we consider, we need only scale U, but this strategy can be used for both L and U.

[3] Knight PA, Ruiz D, Uçar B. “A symmetry preserving algorithm for matrix scaling” SIAM Journal on matrix analysis and applications. 2014

Results

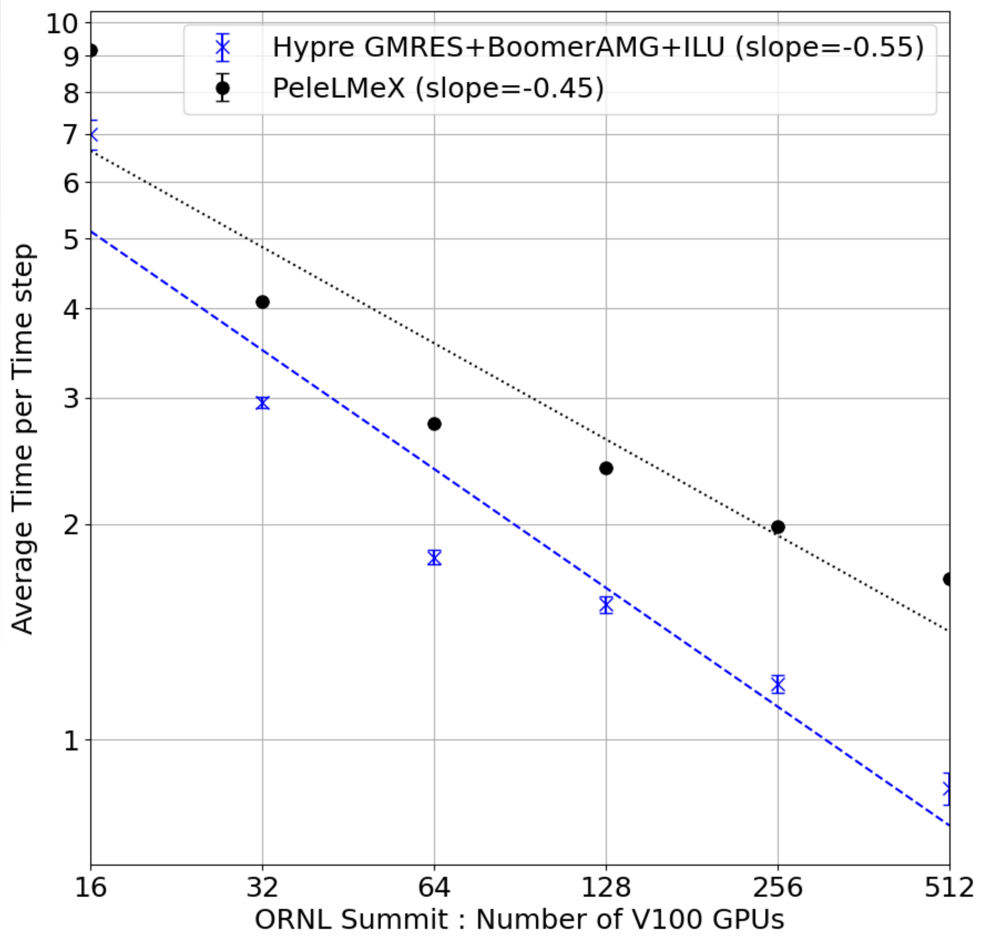


Matrix	Dimension	$dep(L)$	$dep(U)$	$dep(D^{-1}U)$ (row scaling)	$dep(D_r U D_c)$ (Ruiz scaling)
af_0_0_k101	503625	326.95	1.84e8	326.95	320.89
af_shell1	504855	386.66	1.52e8	386.66	407.35
bundle_adj	513351	8.52e6	4.52e11	8.52e6	438.70
F1	343791	335.52	4.89e8	335.52	331.79
offshore	259789	231.86	7.05e15	231.86	222.71
PeleLM331	331	8.37	1.50e6	8.37	4.13
PeleLM2110	2110	16.99	1.09e7	16.99	9.33
PeleLM14186	14186	1.45e4	1.00e6	1.45e4	26.33

Departure from normality when applying ILU(0) to several matrices before scaling (columns 3 and 4) and after Ruiz scaling (column 5) and the incomplete LDU (column 6).

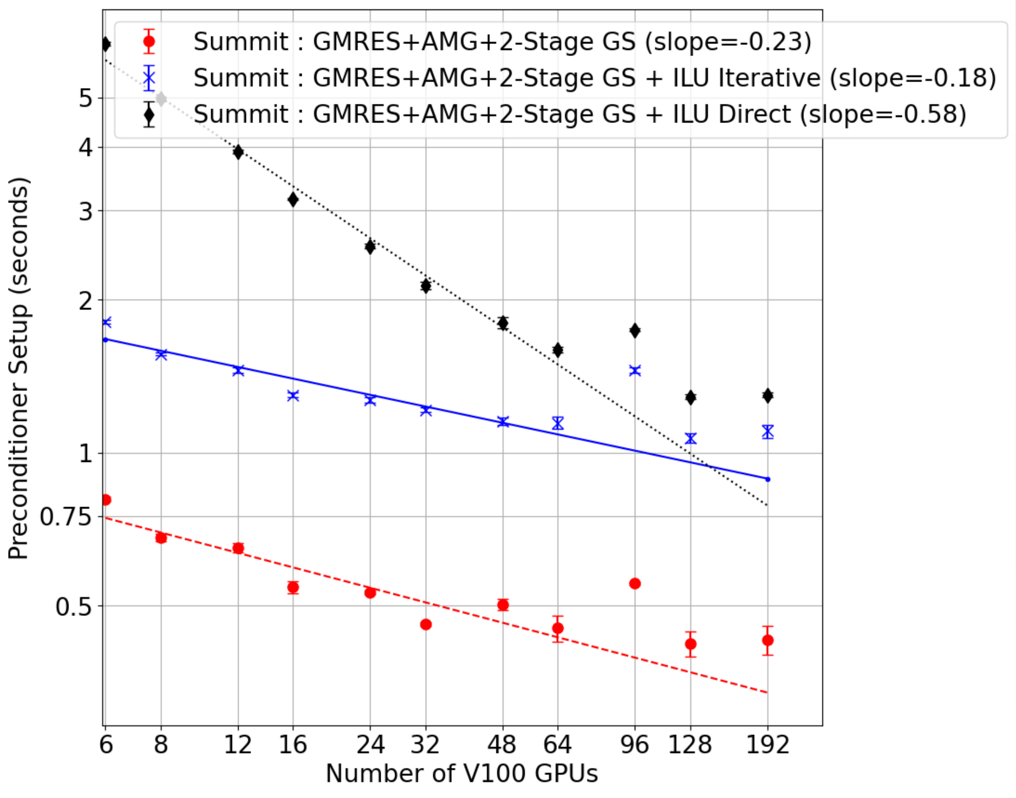
First five come from SuiteSparse Matrix Collection and last three are extracted from PeleLM (with N = 331, 2110, and 14186).

ORNL Summit : PeleLMeX Application Performance



- Solver applied to Nodal Projection
  - 1 Level of AMR (mesh refinement) with a mesh size of 512x512x256.
  - EB volume fraction 1.88e-3
  - Solver is applied at the finest level (no AMReX MLMG coarsening)
  - 100 time steps
- HYPRE solve :
  - ILU0 applied to finest level of BoomerAMG
  - GMRES + BoomerAMG + ILU0 on diagonal block
- Hypre solve is a sizeable fraction of the total time per time step
  - ~12 million unknowns per solve

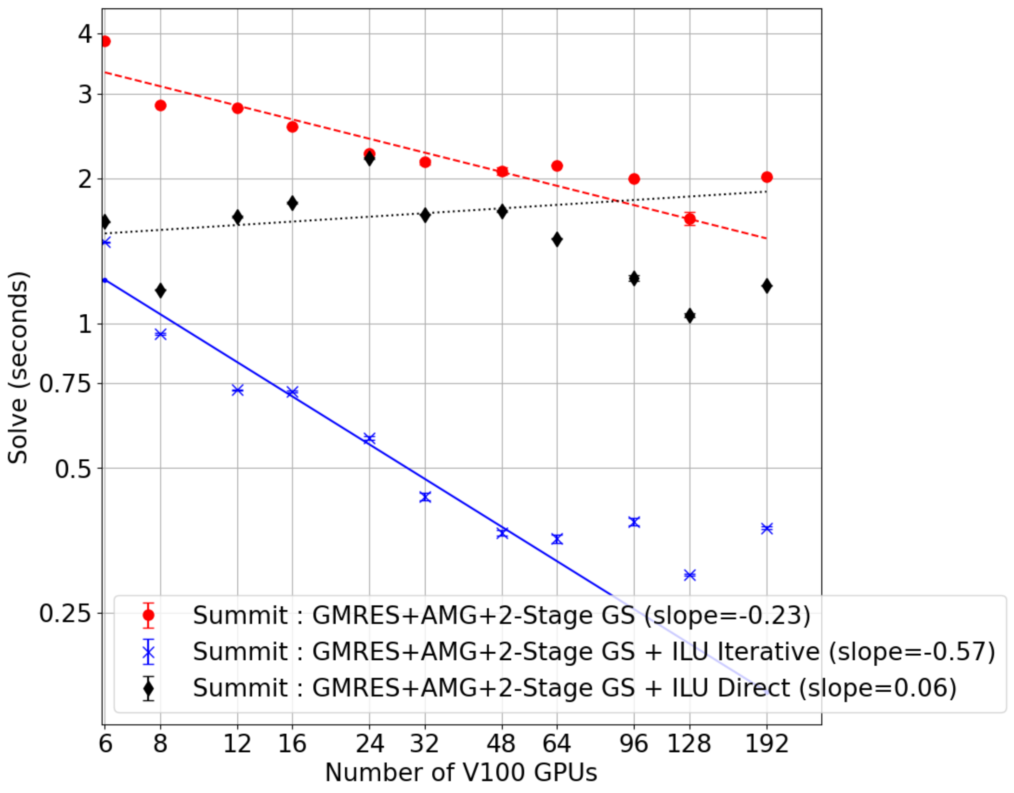
ORNL Summit : Preconditioner Setup

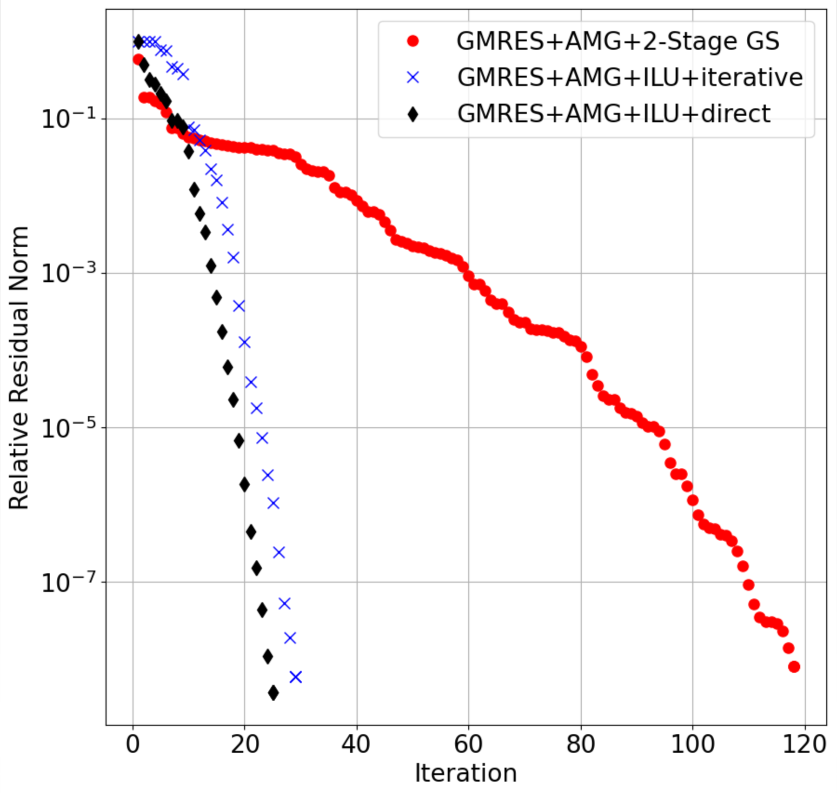


- ILU0 applied to finest level of BoomerAMG
- Two-Stage Gauss-Seidel applied to all other levels [1]
- Solver performance is for an optimized set of parameters
  - Robust search over Boomer AMG parameter space (1000s of trials)
  - 13 Lower, 5 Upper Jacobi Iterations
    - Row scaling applied to U
- Use of Jacobi iteration removes the expensive solve analysis stage of the preconditioner setup

[1] Mullowney et al. “Preparing an Incompressible-Flow Fluid Dynamics Code for Exascale-Class Wind Energy Simulations” SC21.

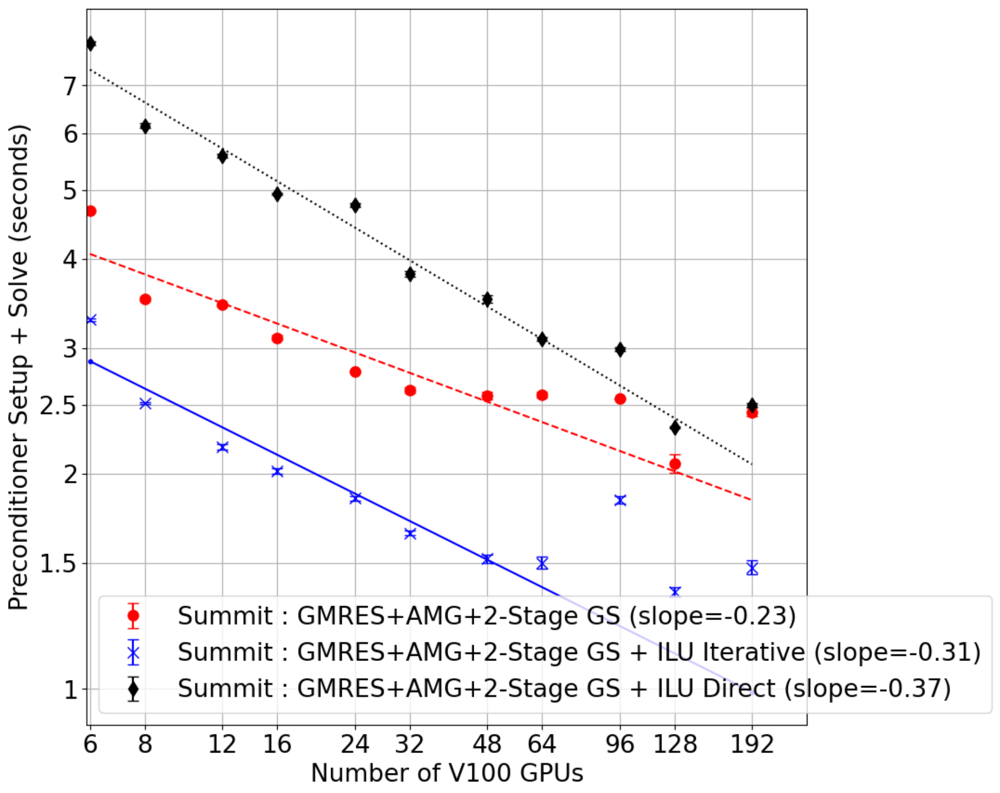
ORNL Summit : Solve





For the PeleLMeX matrices, ILU0-preconditioned solvers are more effective at driving down the convergence of the GMRES solver.

ORNL Summit : Total Time



Effectiveness of ILU0-preconditioned solver (reduced iterations and lower preconditioner setup) results in a significant reduction in time to solution.

Porting to Crusher

- Hypre ILU Algorithms were coded towards Nvidia architectures
  - Heavy reliance on UVM for ILU(k) and ILUT algorithms as well as reordering algorithms
  - In the latter case, this includes RCM host implementations and some other key code components related to Schur Compliment based solves.

- Nvidia Data Types were embedded in algorithm APIs
  - csrsvILU0_t, cusparseSolvePolicy_t, … data types were passed in many methods
  - Porting to Crusher required generalizing these interfaces. Hardware specific functions calls at key places:

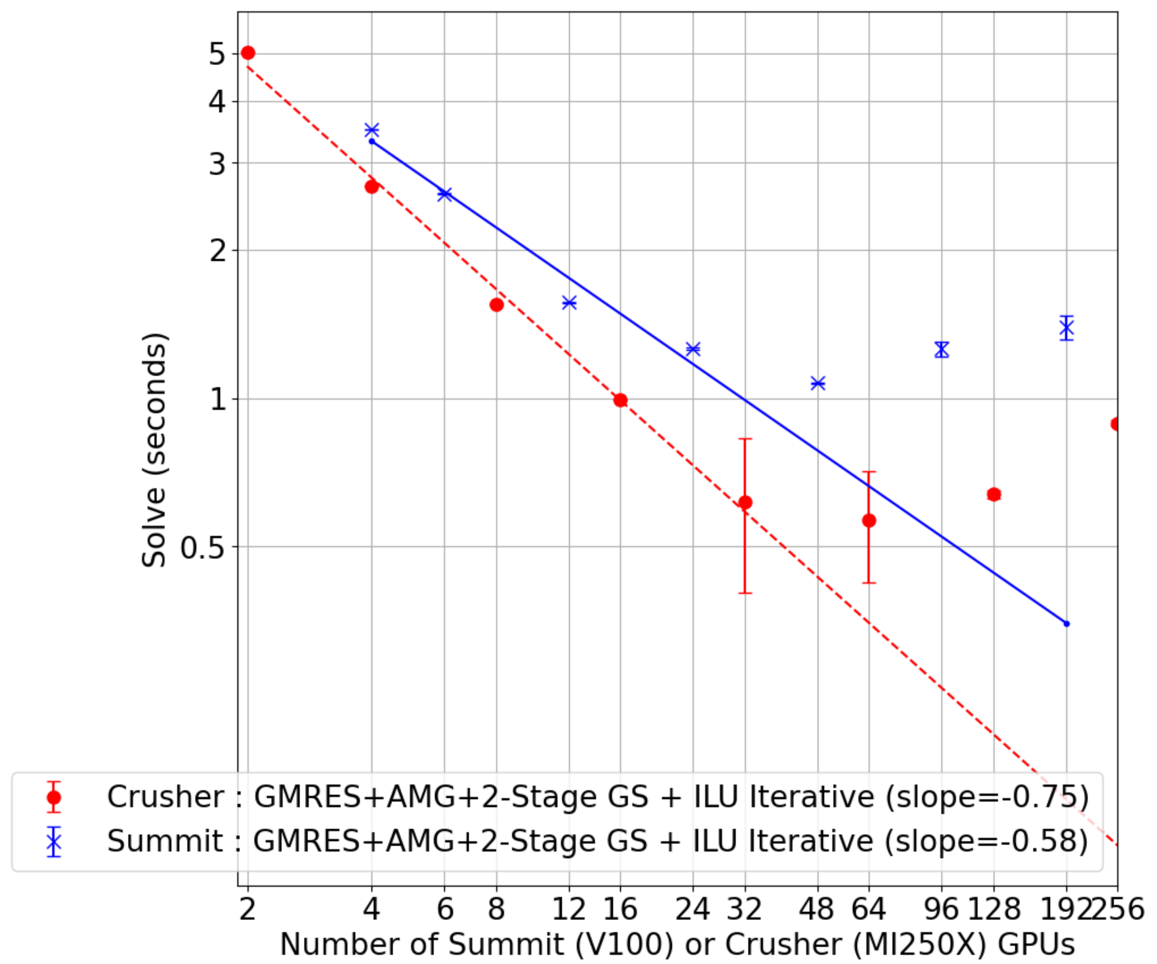
    ILU0 factorization (Rocpsarse)

    Forward and Backward triangular solve (Rocsparse)

    Jacobi Iterative triangular solve written in Cuda (**hipcc** did all the heavy lifting)

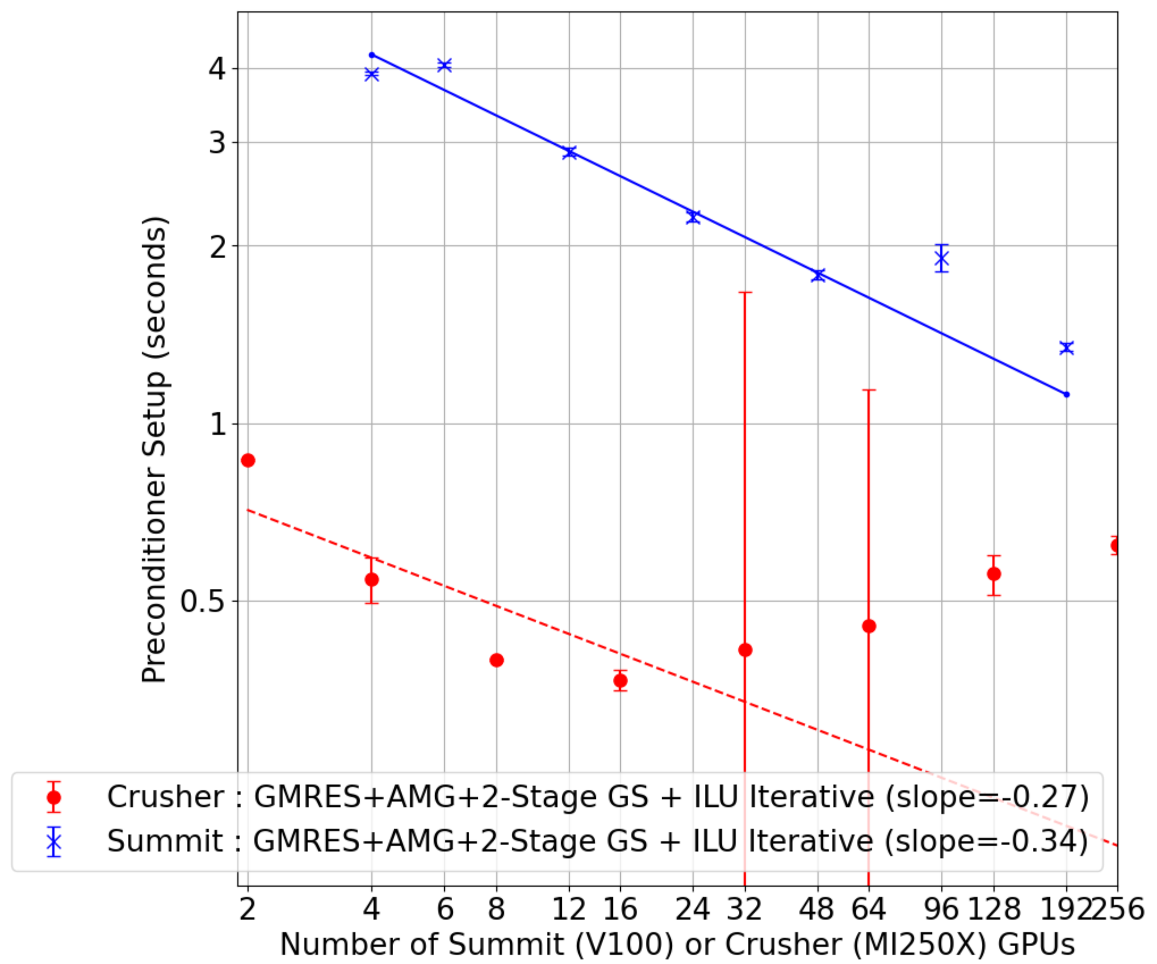
- Porting was done in 1-2 weeks to get minimal implementation that ran successfully on Crusher
  - Block-Jacobi ILU0

Crusher vs Summit : Solve Performance



- For small numbers of GPUs, Crusher solve throughput is roughly equivalent to Summit with 50% more GPUs
  - 4 MI250 ~ 6 V100s
  - 8 MI250 ~ 12 V100s
- Strong scaling is the point at which additional compute hardware no longer provides performance gain.
  - ~400k DoFs/Crusher
  - ~300k DoFS/Summit
  - At the strong scaling limit, Crusher is roughly 40-50% faster
- Significant optimization potential remains
  - Optimization of SpMVs with L/U
  - Hardware level optimization

Crusher vs Summit : Preconditioner Setup Performance

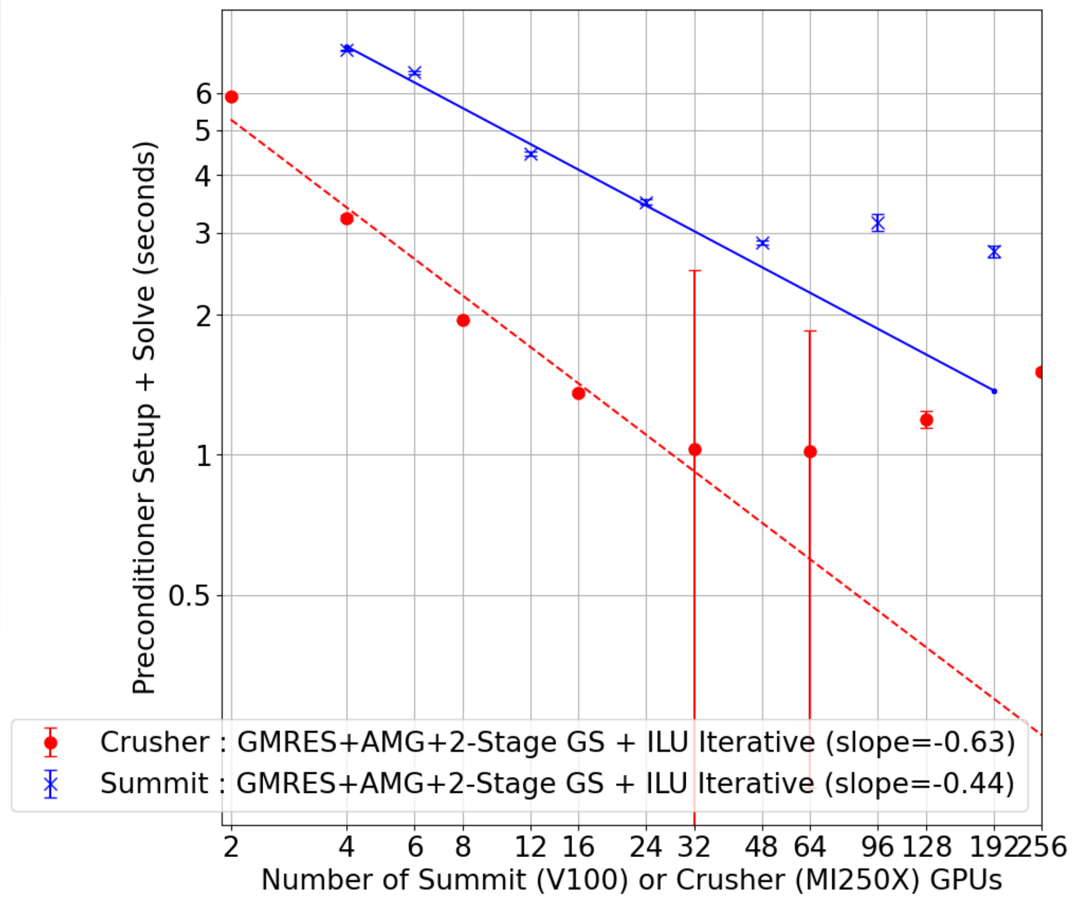


- Preconditioner setup costs included full BoomerAMG setup plus ILU0 factorization at level 0.

- Rocsparse ILU0 factorization is substantially faster than Cusparse equivalent. This accounts for much of the difference between these plots
  - Other key factors include AMG RAP calculation (SGEMM)

- The cause of the large error bars is not known
  - Each data point is an average of 5 trials. Sometimes, an individual run generates poor data.

Crusher vs Summit : Total Performance



Crusher provides impressive performance gains over Summit for block-ILU0, C-AMG, GMRES linear solvers!

Conclusions and Future Work

- Jacobi iterations are a fast and powerful alternative to preconditioned AMG solves that require sparse triangular solve
  - Fast Implementation based on SpMV
  - For U solve, Ruiz or Row scaling required to get a convergent series
  - Also effective with Schur-complement based ILU smoother algorithms
- We’ve explored other matrices from the UFlorida Sparse Matrix collection
  - Often able to find a set of ILU/AMG parameters that yield a faster solve
  - In most cases, the additional preconditioner setup costs (ILU factorization) resulted in a slower time to solution.
  - Why was it so successful for PeleLMeX matrices?
- Faster factorization (ILU(k) and ILUT) could make this more generally useful
  - These run on Nvidia architectures with Unified Memory
- Next steps: Get this running in the PeleLMeX