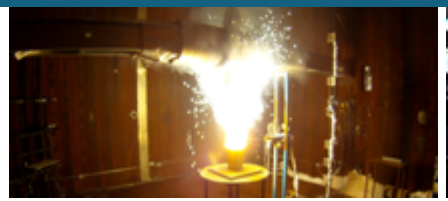
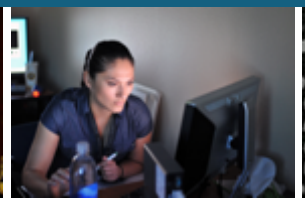
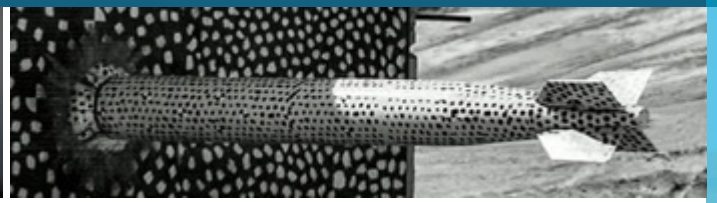
SAND2021-6164C

Sandia National Laboratories

# Polynomial Preconditioned GMRES for GPU Computing







Jennifer Loe and Erik Boman
with Siva Rajamanickam, Christian Glusa, and Ichi Yamazaki

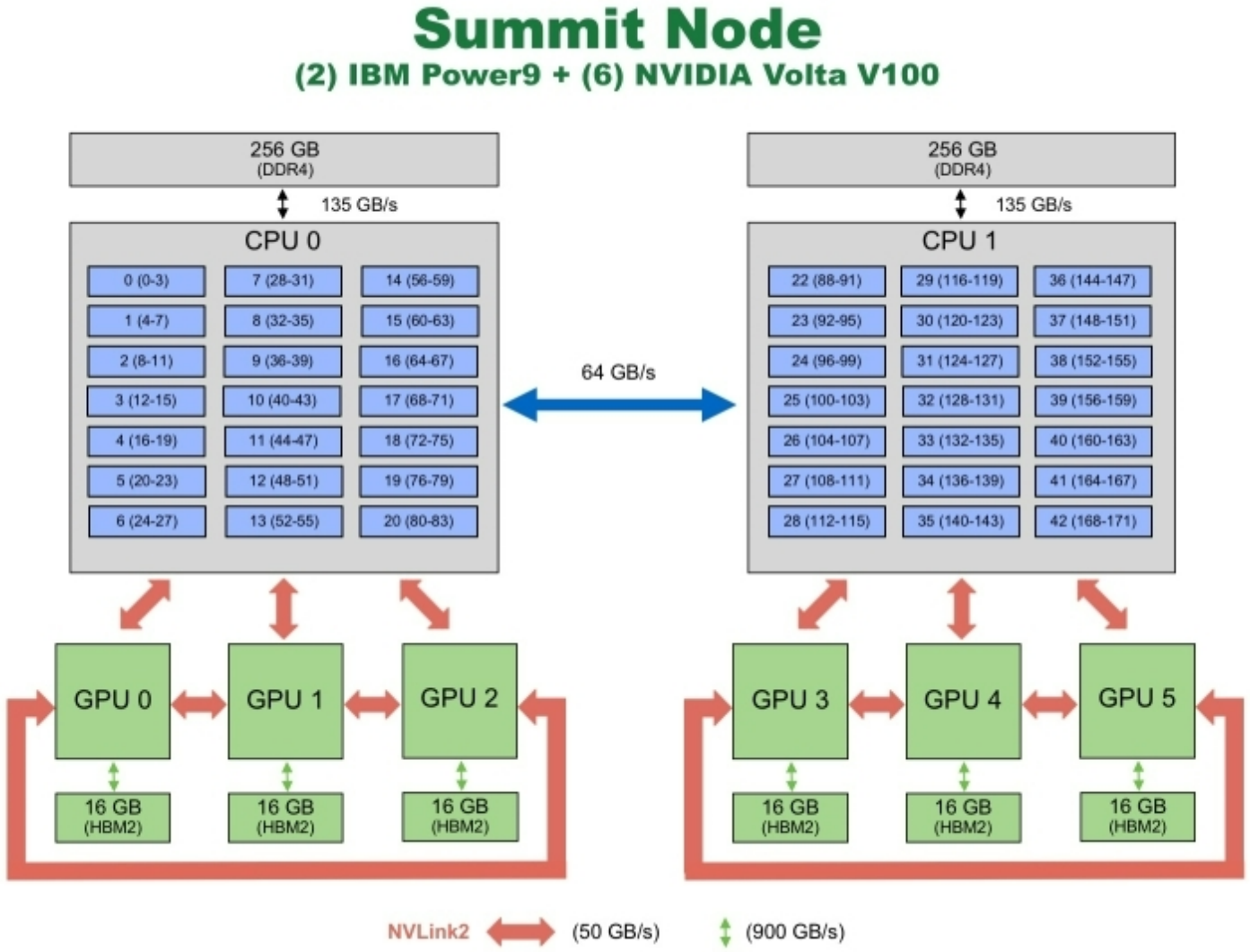
EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

# Why do we care about GPU computing?

- Parallel computing terms:
  - CPU- Heavyweight cores, just a few
  - GPU- Very weak cores, many of them
- Supercomputer Summit has 4608 compute nodes, each with 6 NVIDIA V100 GPUs. Not using GPUs -> not using 97% of Summit's compute power!
- Future exascale computers (Aurora, Frontier, El Capitan) will also be built relying upon GPUs.
- **BIG Question: Can polynomial preconditioning scale to large problems on multiple GPUs?**




https://www.olcf.ornl.gov/summit/

# So how does polynomial preconditioning perform on Summit?

???


- Tests are just beginning!
  - **TODAY: Preliminary results; Algorithmic scaling.**

# What is polynomial preconditioning?

For solving large, sparse linear systems $Ax = b$.

Right-preconditioned problem is $AMM^{-1}x = b$.

Choose $M = p(A)$, where $M$ approximates $A^{-1}$.

We choose p(A) to be the GMRES polynomial.

p(A) can be constructed with a few initial steps of GMRES.

$$p(A) = \sum_{k=1}^{d} \frac{1}{\theta_k} \left(I - \frac{1}{\theta_1}A\right)\left(I - \frac{1}{\theta_2}A\right)\cdots\left(I - \frac{1}{\theta_{k-1}}A\right)$$

Where $\theta_i$ are Harmonic Ritz values from the initial GMRES run.

- See polynomial details in:
  - Loe, Jennifer and Morgan, Ron. *Toward Efficient and Stable Polynomial Preconditioning for GMRES* https://arxiv.org/abs/1911.07065
  - Jennifer Loe, Erik Boman, and Heidi Thornquist. *Polynomial Preconditioned GMRES in Trilinos: Practical Considerations for High-Performance Computing* https://doi.org/10.1137/1.9781611976137.4

# Why use polynomial preconditioning? Why does it have potential for High-Performance Computing?

- Works for general non-symmetric matrices.
- Can compute p(A) for entire matrix A; no need to divide A into subproblems.
- Matrix-free implementation.
- Stable at high degrees via added roots. (See Friday's talks!)
- Can be used to accelerate standard preconditioning.
- Use more SpMVs (local communication) rather than dot products from orthogonalization (global communication).
- Can reduce solve time, iteration count, and SpMVs:
  - Example: Matrix **ML_Geer** from Janna collection: nonsymmetric poroelastic structure problem, n=1.5 million
  - Using GMRES(100) run to tolerance of $10^{-8}$.

Results from ML_Geer:

| Polynomial Degree | Iterations | SpMVs | Solve Time |
|---|---|---|---|
| 20 | 12897 | 260500 | 3214 |
| 40 | 1487 | 61580 | 731.5 |
| 60 | 472 | 29570 | 346.7 |
| 80 | 200 | 16970 | 197 |

# Linear Solvers in Trilinos:


- Trilinos:
  - Large numerical software library
  - Open-source: www.github.com/trilinos
  - Many contributors; primarily developed at Sandia National Labs
  - Packages for: linear solvers, nonlinear solvers, eigensolvers, multigrid preconditioning, mesh partitioning, factorization-based preconditioning, optimization, time integration, automatic differentiation, and more!
- Belos: Krylov Linear Solvers package in Trilinos:
  - All linear algebra kernels are abstracted through “adapter” interface.
  - Uses Epetra, Tpetra, or custom linear algebra kernels
- Kokkos and Kokkos Kernels:
  - Portable parallel linear algebra.
  - Performant BLAS kernels for GPU (single node).
  - Tpetra = Kokkos + MPI

# Implementation of Polynomial Preconditioning in Trilinos:

- Implemented in Belos -> Can be used with any linear algebra backend.
- Use as a stand-alone iteration or as a preconditioner.
- Can be composed with other preconditioners to accelerate them.

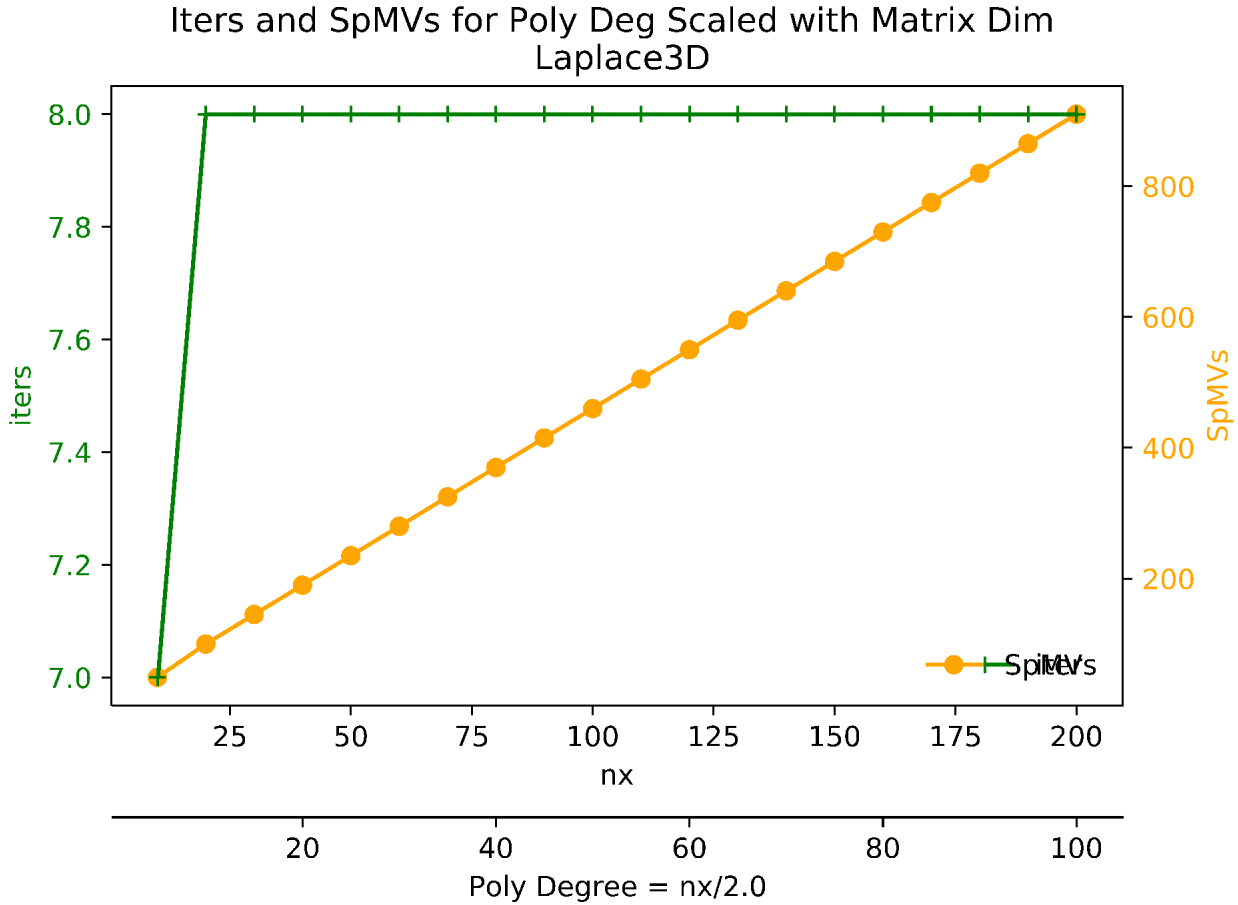
- Experiments to follow:
  - Use Kokkos as the linear algebra backend for solvers.
  - Using GMRES(50) with $x_0 = \mathbf{0}$ and $b = (1,1,1 \ldots, 1)^T$.
  - Stopping criteria: Relative residual tolerance of $10^{-8}$.
  - Tested performance on a single node with V100 GPU.

# Three questions for today's talk:

1. **Algorithmic scaling 1:** How can I use polynomial preconditioning to get a constant number of iterations as I refine the mesh for my PDE problem?
2. **Algorithmic scaling 2:** What happens to GMRES convergence if I fix the polynomial degree and keep refining the PDE mesh anyway?
3. **Multiprecision Applications:** What makes polynomial preconditioning a good candidate for mixed-precision computing?

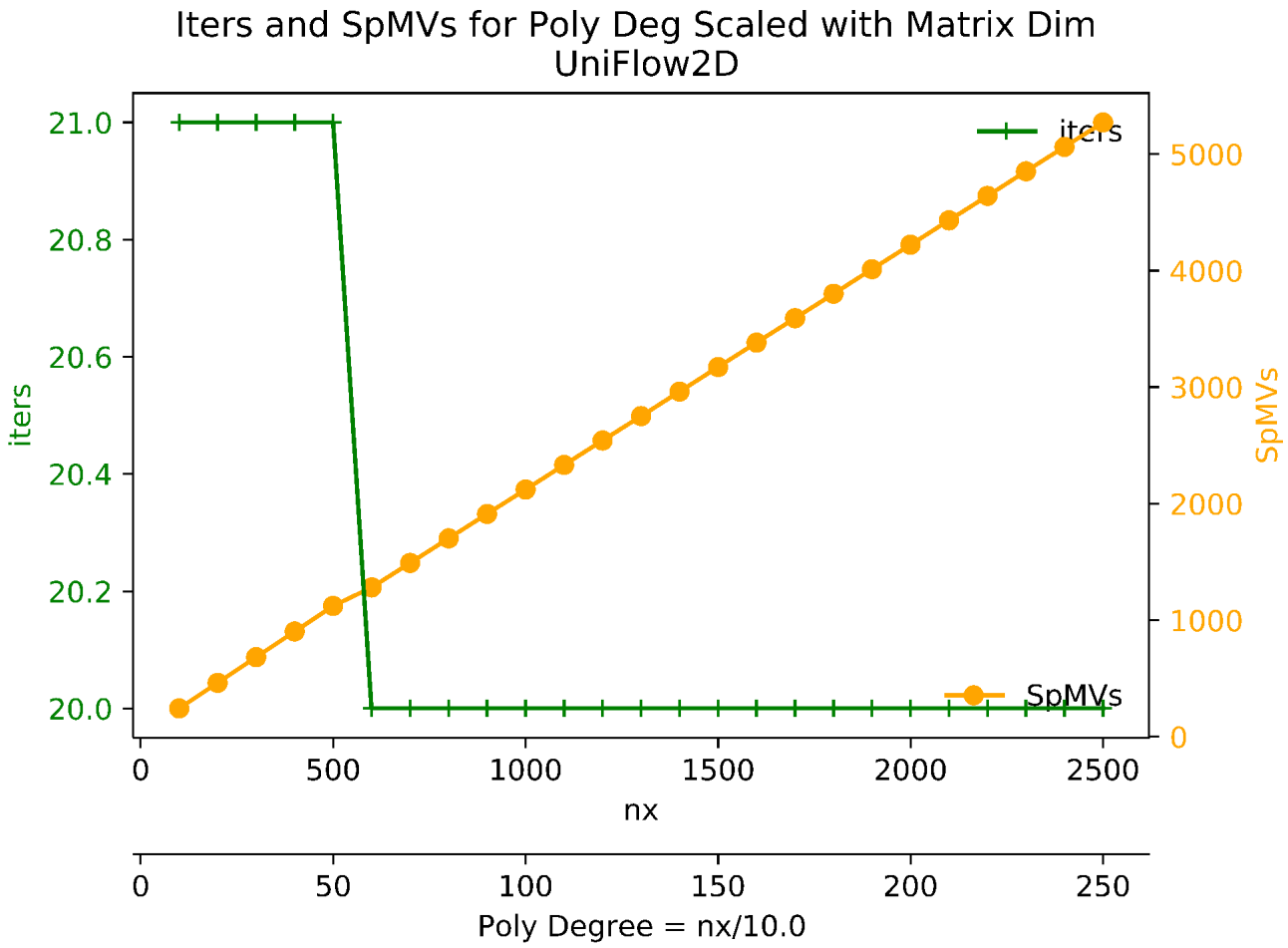
# How to get a constant number of iterations:

- Say PDE mesh has $nx$ grid points in each direction.
- Finer PDE mesh:
  - More small eigenvalues of A
  - Need higher polynomial degree to maintain same convergence.
- How much to increase polynomial degree?
- As $nx$ is doubled, double the poly degree.
- Example: 3D Laplacian
  - $n = nx * nx * nx$
- SpMVs scale linearly, while $n$ grows cubically.



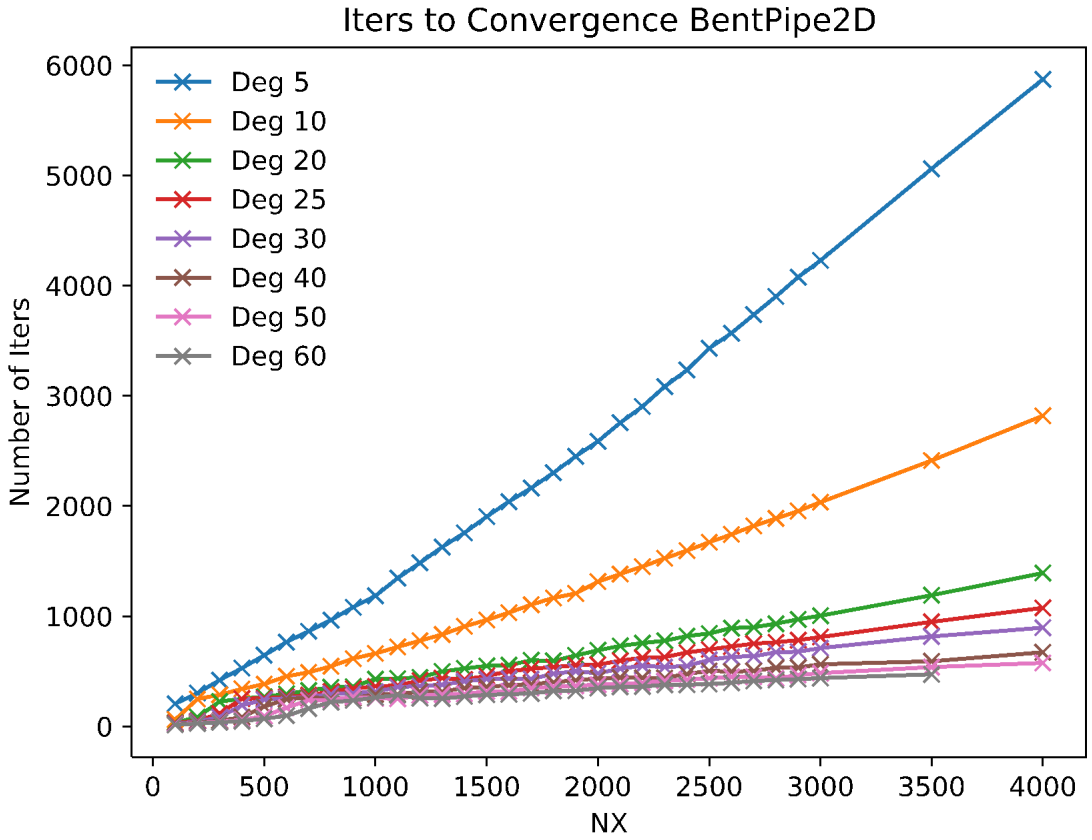
# Another example of near-constant iterations:

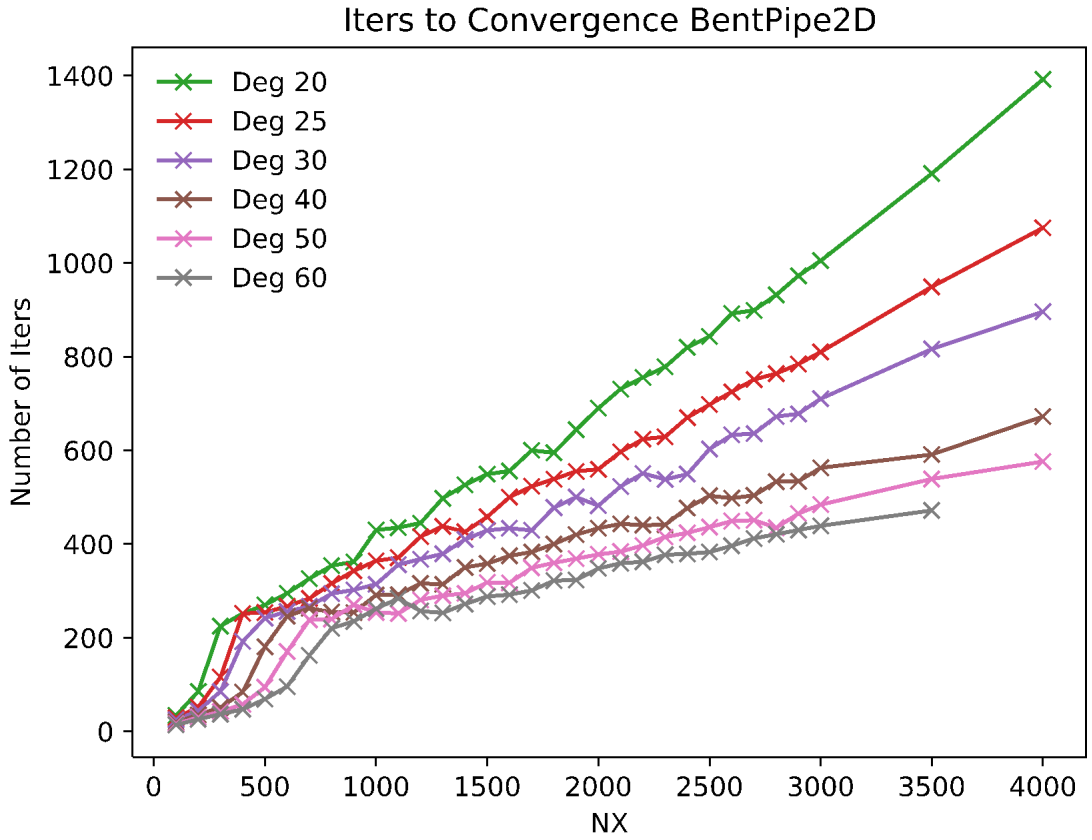
- Matrix = UniFlow2D
- Convection-diffusion, highly nonsymme
- $n = nx * nx$
- Again, As $nx$ is doubled, double the polynomial degree.
- 20 to 21 iterations
- As $n$ grows quadratically, SpMV count grows linearly.



# When you fix the polynomial degree:

- For fixed poly degree, iteration count grows roughly linearly as $nx$ grows.
- So iteration count and SpMV count grow roughly like $\sqrt{n}$.



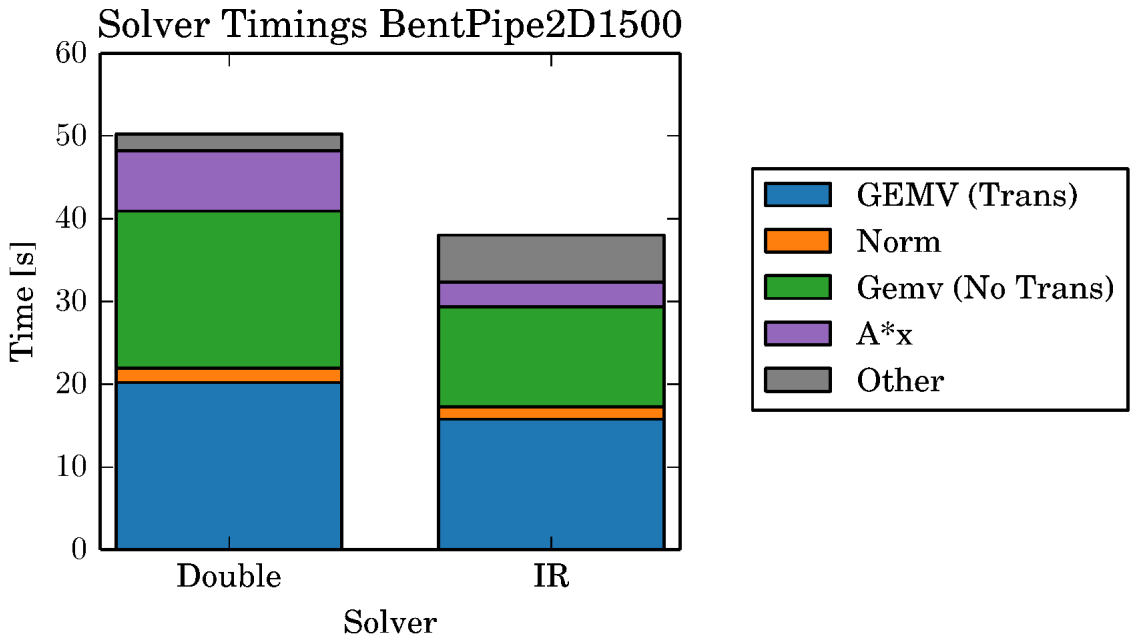




- **Rough conjecture:** We can find a "high enough" polynomial degree so that the iteration count needed for GMRES convergence scales (almost) linearly with respect to $nx$.

# Multi-precision GMRES:

**Algorithm 1** Iterative Refinement with GMRES Error Correction

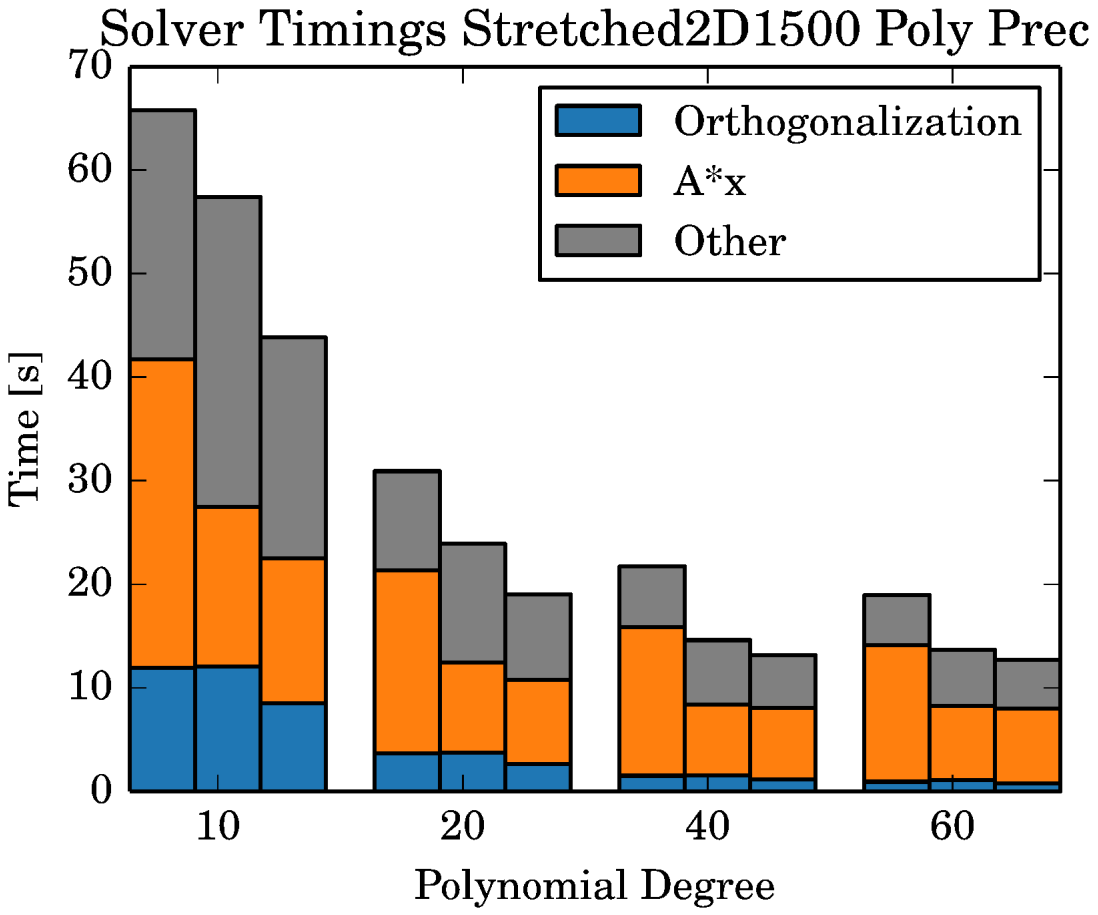
1: $r_0 = b - Ax_0$ [double]
2: **for** $i = 1, 2, \ldots$ until convergence: **do**
3: Use GMRES($m$) to solve $Au_i = r_i$ for correction $u_i$ [single]
4: $x_{i+1} = x_i + u_i$ [double]
5: $r_{i+1} = b - Ax_{i+1}$ [double]
6: **end for**




- GMRES-IR is the preferred GMRES solver for multi-precision computing. (See Carson, Higham, Anzt, Linquist and many others.)
- Our experiments showed 2.5x speedup in SpMV kernel from double to single precision.
- Speedup for A*x comes from better use of the L2 cache in the GPU- "perfect caching" for vector x in single precision.
- Opportunity for polynomial preconditioning!

| | GMRES double | GMRES IR | Speedup |
|---|---|---|---|
| **Total time:** | **50.26** | **38.03** | **1.322** |
| Ortho: GEMV Trans | 20.20 | 15.78 | 1.280 |
| Ortho: GEMV No Trans | 19.01 | 12.10 | 1.571 |
| Ortho (norm) | 1.71 | 1.49 | 1.152 |
| A*x | 7.33 | 2.95 | 2.484 |

# Polynomial Preconditioning in multiple precisions:




**Test Problem:**

- 2D Laplacian, Stretched Grid
- n = 2.25 million
- GMRES(50) to stopping tolerance of $10^{-10}$

**Three solves compared:**

1. GMRES double w/ double precision polynomial. (left)
2. GMRES double w/ single precision polynomial. (middle)
3. GMRES-IR (GMRES with Iterative Refinement) w/ single precision polynomial. (right)

- (Solve times do not include preconditioner creation.)

**Takeaways:**

- Single precision preconditioning improves solve time up to 30% over double precision preconditioning.
- Polynomial preconditioning shifts main expense to SpMV rather than dense orthogonalization kernels.

# Future Questions to Answer:

- **BIG Question: Can polynomial preconditioning scale to large problems on multiple GPUs?**
- How does polynomial preconditioning perform in true weak and strong scaling experiments?
- Does reducing orthogonalization in favor of more SpMVs really pay off on an HPC machine like Summit?
- How does polynomial preconditioning compare to other preconditioners (e.g. Jacobi, ILU, AMG) at large-scale?
- Can we create high enough degree polynomials for life-size physics problems? Is it best to divide problem into subdomains and/or combine with other preconditioners?

Thanks to Christian Glusa, Ichi Yamazaki, and Siva Rajamanickam for their help in this work!

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

# More Polynomial Preconditioning Answers coming up Friday!

- How to make the GMRES polynomial stable by adding extra roots?
- How to give your preconditioner a new speedup: Compose it with a polynomial!
- Ritz values, eigenvectors, and the Arnoldi iteration: Polynomial preconditioning is for eigenvalue problems, too!

All this and more!

**Friday 11:00am CST**

**MS 80** – Recent Advances in Polynomial Preconditioning for Linear Algebra Problems

Thanks to co-authors Ron Morgan and Mark Embree for their help in work leading up to this project.