Scalable Non-blocking Preconditioned Conjugate Gradient Methods

Paul Eller and William Gropp
University of Illinois at Urbana-Champaign
Department of Computer Science

Supercomputing 16

Motivation

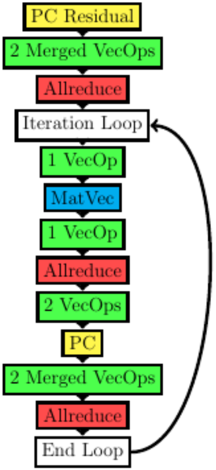
- To achieve best performance on extreme-scale systems, we need more scalable methods
- Blocking collectives are barrier to scalability
  - Increasing cost as core count increases
  - Requires synchronization across cores
- Noise causes performance variation across cores
- Methods with frequent synchronization will struggle to perform well as supercomputers grow larger

Big Picture

- What did we want to do?
  - Study preconditioned conjugate gradient (PCG) methods using non-blocking allreduces
  - Previous research answered many questions about numerical properties
  - But when do scalable PCG methods actually outperform standard PCG?
- What did we do?
  - Theoretically analyzed performance using performance models
  - Started developing software to analyze scalable algorithms
  - Found conditions where scalable PCG methods thrive
  - Developed a new method, PIPE2CG, using performance results to guide method development

Preconditioned Conjugate Gradient Method (PCG)

- Popular iterative method for solving sparse symmetric positive definite linear systems $Ax = b$
- Preconditioners needed in practice to improve convergence
- Blocking allreduce is barrier to scalability for PCG
- Need to minimize allreduce cost and avoid synchronization

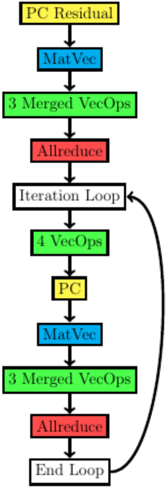


Scaling PCG

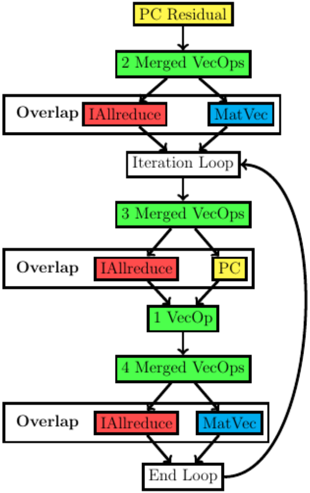
- Scalable approaches:
  - Reduce communication latency by combining multiple allreduces
  - Overlap communication and computation using non-blocking allreduces
- Potential to:
  - Hide or avoid most of allreduce cost
  - Avoid synchronization cost
- Deriving new methods:
  - Use recurrence relations to rearrange order of key kernels
  - Equivalent to PCG in exact arithmetic
  - Increases initialization and vector operations costs

Scalable PCG Methods

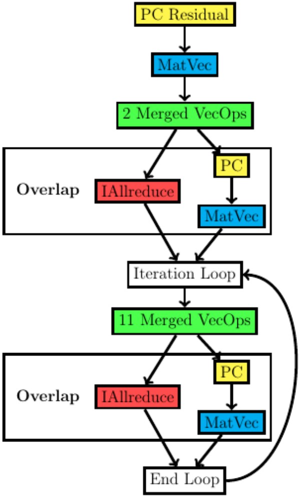
Single Allreduce PCG
D'Azevedo et al 1993



Non-blocking PCG
Gropp 2010



Pipelined PCG
Ghysels et al 2014

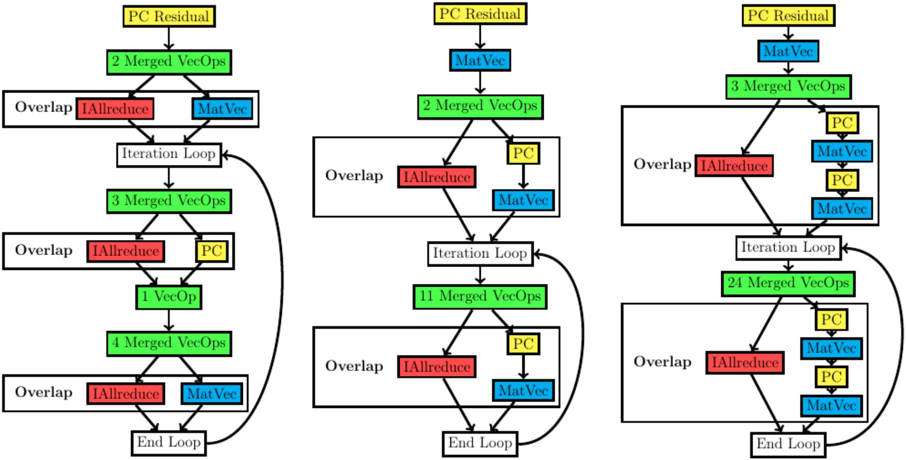


Further Pipelined PCG Methods

- PIPECG seems like reasonable starting point, but requires extra matvecs and PCs
- Instead start with three-term recurrence PCG
- PIPE2CG derived with similar process as two-term recurrence methods
- Three-term recurrences can produce less accurate residuals than two-term recurrences
- Slightly different approach will likely be needed to derive PIPE3CG, PIPE4CG, etc

PIPE2CG

Non-blocking PCG | Pipelined PCG | 2-Iteration Pipelined PCG



Other Scalable PCG Methods

- Communication avoiding Krylov solvers:
  - Potential to reduce communication at multiple levels
  - Preconditioning matrix-powers kernel is difficult
  - PIPE2CG could use matrix-powers kernel to get non-blocking communication-avoiding solver
- Other Methods:
  - Pipelined GMRES: Similar approach as Pipelined PCG methods
  - Hierarchical and nested Krylov methods
  - Enlarged Krylov methods

Merged Vector Operations

Separate VecOps		Merged VecOps

AYPX(p, beta, u);
AYPX(s, beta, w);
Dot(p, s, gamma);
	→	for(i=0; i<n; i++) {
		p[i]=u[i]+beta*p[i];
		s[i]=w[i]+beta*s[i];
		gamma+=p[i]*s[i]; }

- Rearranging PCG introduces extra vector operations
- Minimize increased cost by performing vector operations element-wise
- Reduces vector reads
- Still requires extra vector writes

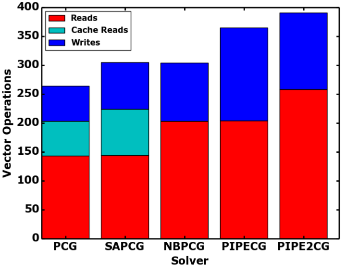


Figure: Vector reads and writes for 20 iterations of each method

Non-blocking Allreduce

- Expected Usage:
  - Starts allreduce and returns control to user
  - Use wait to verify allreduce has completed
- Problem: May execute blocking allreduce in wait

Ideal Approach	MPI_Test Approach:

`MPI_Iallreduce();`
`computation();`
`MPI_Wait();`
	`MPI_Iallreduce();`
	`for(...) {`
	`  chunk_comp();`
	`  MPI_Test(); }`
	`MPI_Wait();`

- Approaches:
  - MPI_Test() - Break computation into chunks and give MPI control of thread
  - Progress threads - Dedicate one or more threads per node to communication
  - Hardware acceleration - Execute non-blocking allreduce in hardware

Test Setup

- Experimental Setup
  - Tests run on Blue Waters, a Cray XE6/XK7, with custom PETSc solvers
  - Primarily use block-Jacobi incomplete-Cholesky preconditioner and Poisson matrices
- Benchmarking Approach
  - Use Hoefler et al 2015 and Hunold et al 2014 for guidance on producing accurate timings
  - Developed code to collect parallel timings and compute statistics
  - Produced iteration and 21-iteration test stats
  - Primarily plot median runtimes, using 99% confidence intervals for 21-iteration tests

Scalable PCG Performance Model

- Motivation:
  - Suboptimal non-blocking allreduce performance in practice
  - Performance variation at scale
- Model Setup:
  - LogGOPS model for parallel communication on Blue Waters
  - Netgauge for network parameters
  - Modified STREAM benchmark for computation parameters
- Overview of accuracy:
  - Accurately captures general solver trends
  - Improved cache, noise, and non-blocking allreduce models would improve runtime prediction accuracy

Expected Performance

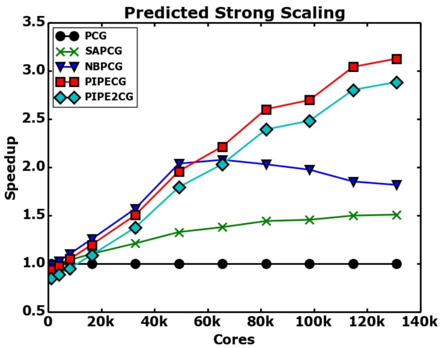


Figure: 27-point Poisson problem with 512^3 rows for 21 iteration tests

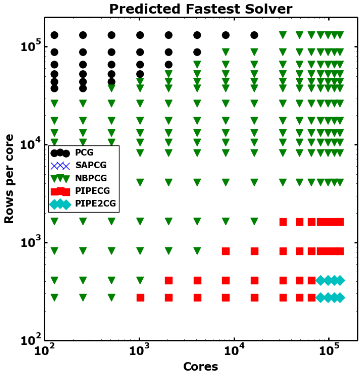


Figure: 27-point Poisson matrix for 21 iteration tests

Weak Scaling Results

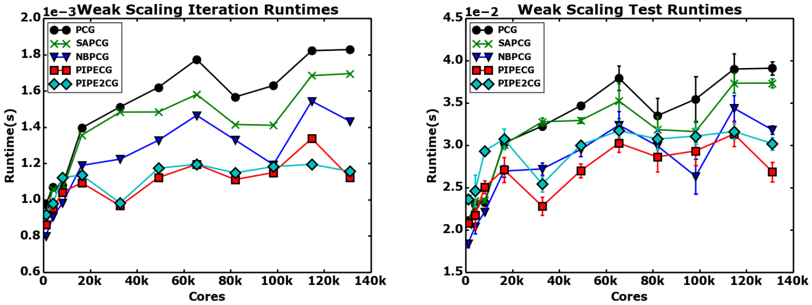


Figure: 27-point Poisson matrices with 4k rows per core

- Non-blocking solvers can hide communication costs and absorb noise to produce more consistent runtimes
- Increased initialization costs limit performance for non-blocking methods for 21-iteration tests

Strong Scaling Results

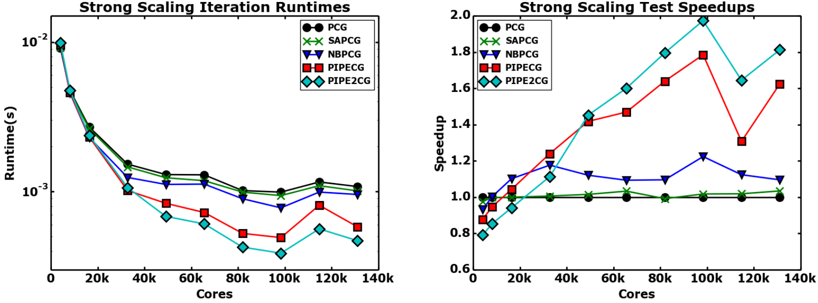
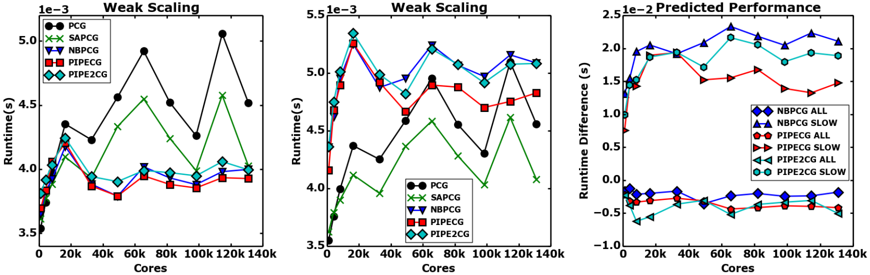


Figure: 27-point Poisson matrices with 512^3 rows

- Non-blocking methods struggle to scale further once computation cannot fully overlap communication
- Must overlap allreduce with more matrix kernels as work per core decreases and communication costs increase

Noise



(a) All cores (b) Slowest core (c) Prediction accuracy

Figure: Total runtime prediction accuracy using all cores and slowest core each iteration for 27-point Poisson matrices with 13k rows per core

Accuracy

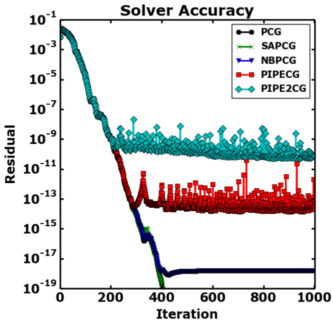


Figure: Computed residual norms for 5-point Poisson matrix with 256^2 rows

- Most tested matrices produced more accurate residuals
- Ghysels and Vanroose suggest residual replacement strategy
- Cools et al 2016 developed PIPECG method to monitor rounding error and use residual replacement as needed
- Effective for cheaper preconditioners and problems with O(1000) iteration counts

Preconditioners

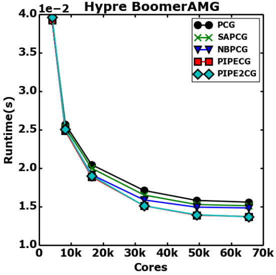


Figure: Strong scaling 27-point Poisson matrix with 512^3 rows

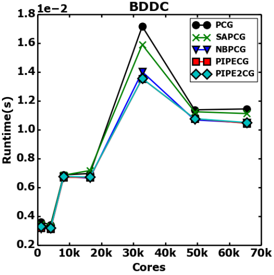


Figure: Weak scaling 3-d Laplacian matrix with 4k rows per core

- Assumed preconditioners call MPI routines enough to progress allreduce
- More detailed study needed to determine most scalable settings

Weak Scaling Poisson Matrices

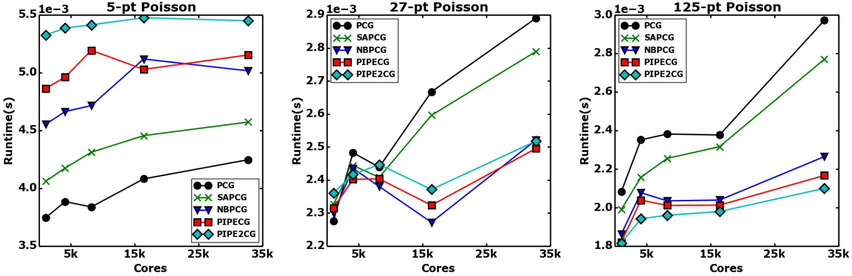


Figure: Iteration runtimes for matrices with about 220k nonzeros per process

- Matrices with more nonzeros per row can overlap allreduce with fewer rows per core, limiting overhead
- 2-d finite element matrices (9 and 18 nonzeros per row) show performance similar to 5-point Poisson matrices
- 3-d finite element matrices (80 nonzeros per row) show performance similar to 125-point Poisson matrices

Analysis

- Scalable PCG methods:
  - More efficient communication
  - Increased initialization and vector operations costs
  - Effective for simpler preconditioners
  - Some speedup for more complex preconditioners without code modification
  - Hardware accelerated collectives may improve non-blocking allreduce

How to use in practice

- Factors to consider for problems:
  - Expected iteration count
  - Matrix nonzeros per row
- Best Problems:
  - Enough iterations to converge to overcome increased initialization cost
  - More nonzeros per row in matrix

How to use in practice

- Factors to consider for method:
  - Matrix/Vector rows per core
  - Core count
  - Expected system noise
- Method Selection:
  - Not a single fastest PCG method
  - Suite of scalable solvers provides best performance
  - Use least rearranged method capable of effectively overlapping allreduce with matrix kernels
  - May need extra rows per core in noisier environments

Conclusions and Future Work

- Conclusions:
  - Can rearrange PCG to produce significant speedups at scale
  - Discovered conditions where scalable PCG methods outperform standard PCG using experiments on up to 128k cores of Blue Waters
  - Developed a new method, PIPE2CG, which outperforms other PCG methods for lower work per core and higher core counts
- Areas for Future Research:
  - Solver performance in noisy environments
  - Effectively using more complex preconditioners
  - Performance, accuracy, and usability within scientific applications
  - Non-blocking variations of other Krylov solvers
- Paper goes into much more detail on many topics

Backup Slides

Rearranging PCG

- Use recurrence relations to change vector used in key kernels
- Can then rearrange order of key kernels
- Example:
  - Start: $r_j \leftarrow r_{j-1} - \alpha_j w_j$ and $z_j \leftarrow M r_j$
  - Replace: $z_j \leftarrow M(r_{j-1} - \alpha_j w_j) \leftarrow M r_{j-1} - \alpha_j M w_j$
  - End: $s_j \leftarrow M w_j$ and $z_j \leftarrow z_{j-1} - \alpha_j s_j$
- Equivalent to PCG in exact arithetic
- Rearrangement produces additional initialization and vector operations costs

PCG

$r_0 \leftarrow b - Ax_0$
$z_0 \leftarrow Mr_0$

$\gamma_0 \leftarrow (z_0, r_0)$ ▷ 2 Merged Operations
$norm_0 \leftarrow \sqrt{(z_0, z_0)}$ ▷ Blocking Allreduce

for j=1,2,...,until convergence **do**
  if i **then** $\beta_j \leftarrow \gamma_{j-1}/\gamma_{j-2}$
  else $\beta_j \leftarrow 0.0$
  $p_j \leftarrow z_{j-1} + \beta p_{j-1}$
  $w_j \leftarrow Ap_j$
  $\delta_j \leftarrow (p_j, w_j)$ ▷ Blocking Allreduce
  $\alpha_j \leftarrow \gamma_{j-1}/\delta_j$
  $x_j \leftarrow x_{j-1} + \alpha_j p_j$
  $r_j \leftarrow r_{j-1} - \alpha_j w_j$
  $z_j \leftarrow Mr_j$
  $\gamma_j \leftarrow (z_j, r_j)$ ▷ 2 Merged Operations
  $norm_j \leftarrow \sqrt{(z_j, z_j)}$ ▷ Blocking Allreduce

- PCG contains four key kernels:
  - Matrix-vector multiply
  - Preconditioner
  - Vector operations
  - Allreduce
- PCG contains two blocking allreduces

Single Allreduce PCG (SAPCG)

$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad s_0 \leftarrow Az_0$
$\delta_0 \leftarrow (z_0, s_0) \quad \gamma_0 \leftarrow (z_0, r_0)$
$norm_0 \leftarrow \sqrt{(z_0, z_0)}$ ▷ 3 Merged Operations
Allreduce on δ_0, γ_0, $norm_0$ ▷ Blocking

for j=1,2,...,until convergence **do**
  if $j > 1$ **then** $\beta_j \leftarrow \gamma_{j-1}/\gamma_{j-2}$
  else $\beta_j \leftarrow 0.0$
  $p_j \leftarrow z_{j-1} + \beta_j p_{j-1}$
  $w_j \leftarrow s_{j-1} + \beta_j w_{j-1}$
  $\phi_j \leftarrow \delta_{j-1} - \beta_j^2 * \phi_{j-1}/\beta_{j-1}^2$
  $\alpha_j \leftarrow \beta_j/\phi_j$
  $x_j \leftarrow x_{j-1} + \alpha_j p_j$
  $r_j \leftarrow r_{j-1} - \alpha_j w_j$
  $z_j \leftarrow Mr_j \quad s_j \leftarrow Az_j$
  $\delta_j \leftarrow (z_j, s_j) \quad \gamma_j \leftarrow (z_j, r_j)$
  $norm_j \leftarrow \sqrt{(z_j, z_j)}$ ▷ 3 Merged Operations
  Allreduce on δ_j, γ_j, $norm_j$ ▷ Blocking

- Uses single blocking allreduce
- Algorithm from D'Azevedo et al 1993

Non-blocking PCG (NBPCG)

$r_0 \leftarrow b - Ax_0 \qquad z_0 \leftarrow Mr_0$
$\gamma_0 \leftarrow (z_0, r_0) \qquad norm_0 \leftarrow \sqrt{(z_0, z_0)}$ ▷ 2 Merged Ops
MPI_Iallreduce on γ_0, $norm_0$
$Z_0 \leftarrow Az_0$ ▷ Overlap Comm and Comp

for j=1,2,...,until convergence **do**
  if i **then** $\beta_j \leftarrow \gamma_{j-1}/\gamma_{j-2}$
  else $\beta_j \leftarrow 0.0$
  $p_j \leftarrow z_{j-1} + \beta_j p_{j-1}$ ▷ 3 Merged Operations
  $s_j \leftarrow Z_{j-1} + \beta_j s_{j-1} \qquad \delta_j \leftarrow (p_j, s_j)$
  MPI_Iallreduce on δ_j
  $S_j \leftarrow Ms_j$ ▷ Overlap Comm and Comp
  $\alpha_j \leftarrow \gamma_{j-1}/\delta_j \qquad x_j \leftarrow x_{j-1} + \alpha_j p_j$
  $r_j \leftarrow r_{j-1} - \alpha_j s_j$ ▷ 4 Merged Operations
  $z_j \leftarrow z_{j-1} - \alpha_j S_j$
  $\gamma_j \leftarrow (r_j, z_j) \qquad norm_j \leftarrow \sqrt{(z_j, z_j)}$
  MPI_Iallreduce on γ_j, $norm_j$
  $Z_j \leftarrow Az_j$ ▷ Overlap Comm and Comp

- Overlaps Matvec and PC each with non-blocking allreduce
- Algorithm from Gropp 2010

Pipelined PCG (PIPECG)

$r_0 \leftarrow b - Ax_0 \quad u_0 \leftarrow Mr_0 \quad w_0 \leftarrow Au_0$

$\delta_0 \leftarrow (w_0, u_0)$ ▷ 3 Merged Operations
$\gamma_0 \leftarrow (r_0, u_0) \quad norm_0 \leftarrow \sqrt{(u_0, u_0)}$

MPI_Iallreduce on δ_0, γ_0, $norm_0$ ▷ Overlap Comm
$m_0 \leftarrow Mw_0 \quad n_0 \leftarrow Am_0$ ▷ and Comp

for j=1,2,...,until convergence **do**
  if $j > 1$ **then** $\beta_j \leftarrow \gamma_{j-1}/\gamma_{j-2}$
    $\alpha_j \leftarrow \gamma_{j-1}/(\delta_{j-1} - \beta_j/\alpha_{j-1}\gamma_{j-1})$
  else $\beta_j \leftarrow 0.0 \quad \alpha_j \leftarrow \gamma_{j-1}/\delta_{j-1}$
  $z_j \leftarrow n_{j-1} + \beta_j z_{j-1}$ ▷ 11 Merged Operations
  $q_j \leftarrow m_{j-1} + \beta_j q_{j-1} \quad p_j \leftarrow u_{j-1} + \beta_j p_{j-1}$
  $s_j \leftarrow w_{j-1} + \beta_j s_{j-1} \quad x_j \leftarrow x_{j-1} + \alpha_j p_{j-1}$
  $u_j \leftarrow u_{j-1} - \alpha_j q_{j-1} \quad w_j \leftarrow w_{j-1} - \alpha_j z_{j-1}$
  $r_j \leftarrow r_{j-1} - \alpha_j s_{j-1} \quad \delta_j \leftarrow (w_j, u_j)$
  $\gamma_j \leftarrow (r_j, u_j) \quad norm_j \leftarrow \sqrt{(u_j, u_j)}$

  MPI_Iallreduce on δ_j, γ_j, $norm_j$ ▷ Overlap Comm
  $m_j \leftarrow Mw_j \quad n_j \leftarrow Am_j$ ▷ and Comp

- Overlaps Matvec and PC with single non-blocking allreduce
- Algorithm from Ghysels and Vanroose 2014

2-iteration Pipelined PCG (PIPE2CG)

$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad w_0 \leftarrow Az_0$
$\delta_0 \leftarrow (z_0, w_0) \quad \beta_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)}$ ▷ 3 Merged Operations

MPI_Iallreduce on δ_0, β_0, $norm_0$
$p_0 \leftarrow Mw_0 \quad q_0 \leftarrow Ap_0$ ▷ Overlap Comm and Comp
$c_0 \leftarrow Mq_0 \quad d_0 \leftarrow Ac_0$

for j=1,3,...,until convergence **do**
  if $j > 1$ **then**
    $\rho_{j-1} \leftarrow 1/(1 - (\gamma_{j-1}\beta_{j-1})/(\gamma_{j-2}\beta_{j-2}\rho_{j-2}))$
    $\gamma_{j-1} \leftarrow \beta_{j-2}/\delta_{j-2}$
    $\phi \leftarrow [\rho_{j-1}, -\rho_{j-1}\gamma_{j-1}, (1 - \rho_{j-1})]$
    $\delta_{j-1} \leftarrow \phi_0\phi_0\lambda_8 - 2\phi_0\phi_1\lambda_2 + 2\phi_0\phi_2\lambda_3 + \phi_1\phi_1\lambda_4 - 2\phi_1\phi_2\lambda_5 + \phi_2\phi_2\lambda_9$
    $\beta_{j-1} \leftarrow \phi_0\phi_0\lambda_0 - 2\phi_0\phi_1\lambda_8 + 2\phi_0\phi_2\lambda_7 + \phi_1\phi_1\lambda_2 - 2\phi_1\phi_2\lambda_3 + \phi_2\phi_2\lambda_9$
    $\rho_j \leftarrow 1/(1 - (\gamma_j\beta_{j-1})/(\gamma_{j-1}\beta_{j-2}\rho_{j-1}))$
    $\gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$
  else $\rho_j \leftarrow 1 \quad \gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$
  VecPipelined_PIPE2CG()

  MPI_Iallreduce on λ_0 to λ_9 ▷ Overlap Comm
  $c_j \leftarrow Mq_j \quad d_j \leftarrow Ac_j$ ▷ and Comp
  $g_j \leftarrow Md_j \quad h_j \leftarrow Ag_j$

PIPE2CG Vector Operations

$\mu_j \leftarrow 1 - \rho_j \qquad \mu_{j-1} \leftarrow 1 - \rho_{j-1}$

if $j > 1$ **then** ▷ 8 Merged Ops

$\quad x_{j-1} \leftarrow \rho_{j-1}(x_{j-2} + \gamma_{j-1} z_{j-2}) + \mu_{j-1} x_{j-3}$

$\quad r_{j-1} \leftarrow \rho_{j-1}(r_{j-2} - \gamma_{j-1} w_{j-2}) + \mu_{j-1} r_{j-3}$

$\quad z_{j-1} \leftarrow \rho_{j-1}(z_{j-2} - \gamma_{j-1} p_{j-2}) + \mu_{j-1} z_{j-3}$

$\quad w_{j-1} \leftarrow \rho_{j-1}(w_{j-2} - \gamma_{j-1} q_{j-2}) + \mu_{j-1} w_{j-3}$

$\quad p_{j-1} \leftarrow \rho_{j-1}(p_{j-2} - \gamma_{j-1} c_{j-2}) + \mu_{j-1} p_{j-3}$

$\quad q_{j-1} \leftarrow \rho_{j-1}(q_{j-2} - \gamma_{j-1} d_{j-2}) + \mu_{j-1} q_{j-3}$

$\quad c_{j-1} \leftarrow \rho_{j-1}(c_{j-2} - \gamma_{j-1} g_{j-1}) + \mu_{j-1} c_{j-3}$

$\quad d_{j-1} \leftarrow \rho_{j-1}(d_{j-2} - \gamma_{j-1} h_{j-1}) + \mu_{j-1} d_{j-3}$

$x_j \leftarrow \rho_j(x_{j-1} + \gamma_j z_{j-1}) + \mu_j x_{j-2}$

$r_j \leftarrow \rho_j(r_{j-1} - \gamma_j w_{j-1}) + \mu_j r_{j-2}$ ▷ 16 Merged

$z_j \leftarrow \rho_j(z_{j-1} - \gamma_j p_{j-1}) + \mu_j z_{j-2}$ ▷ Ops

$w_j \leftarrow \rho_j(w_{j-1} - \gamma_j q_{j-1}) + \mu_j w_{j-2}$

$p_j \leftarrow \rho_j(p_{j-1} - \gamma_j c_{j-1}) + \mu_j p_{j-2}$

$q_j \leftarrow \rho_j(q_{j-1} - \gamma_j d_{j-1}) + \mu_j q_{j-2}$

$\lambda_0 \leftarrow (z_j, w_j) \qquad \lambda_1 \leftarrow (z_j, q_j) \qquad \lambda_2 \leftarrow (z_j, w_{j-1})$

$\lambda_3 \leftarrow (p_j, q_j) \qquad \lambda_4 \leftarrow (p_j, w_{j-1}) \qquad \lambda_5 \leftarrow (z_{j-1}, w_{j-1})$

$\lambda_6 \leftarrow (z_j, r_j) \qquad \lambda_7 \leftarrow (z_j, r_{j-1}) \qquad \lambda_8 \leftarrow (z_{j-1}, r_{j-1})$

$\lambda_9 \leftarrow (z_j, z_j)$

$\delta_j \leftarrow \lambda_0 \qquad \beta_j \leftarrow \lambda_6 \qquad norm_j \leftarrow \sqrt{\lambda_9}$

Scalable PCG Methods

Method	VecOps	Allr	Overlap	Init Costs
PCG	6	2 Allr	None	1 PC
SAPCG	7	1 Allr	None	1 PC, 1 Matvec
NBPCG	8	2 Iallr	Matvec or PC	1 PC, 1 Matvec
PIPECG	11	1 Iallr	Matvec, PC	2 PC, 2 Matvec
PIPE2CG*	24	1 Iallr	2 Matvec, 2 PC	3 PC, 3 Matvec

Table: Differences between each PCG method
*PIPE2CG computes two iterations of PCG each iteration

- Custom versions of solvers are implemented in PETSc
- Matrices are stored in MPIAIJ sparse compressed row format
- Primary preconditioner is block-Jacobi incomplete-Cholesky

Matrix Experiments - Finite Element

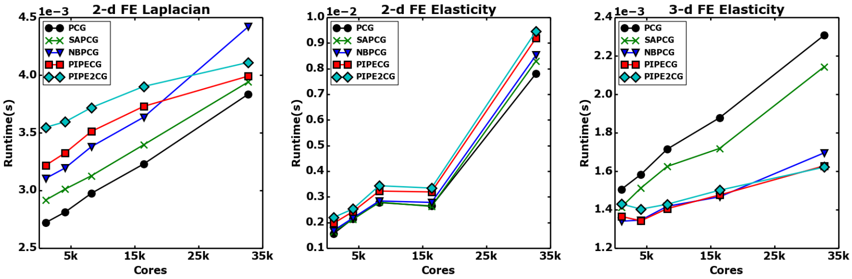


Figure: Weak scaling tests for median iteration runtimes for matrices with about 220k nonzeros per process for 2-d (9 and 18 nonzeros per row) and 3-d (80 nonzeros per row) finite element matrices

Allreduce and Vector Operations

- Allreduce:
  - Investigated time spent in MPI_Test() and MPI_Wait() compared to blocking allreduce for 27-point Poisson matrix with 4k rows per core
  - Potential for 10-20% overall speedups by further optimizing non-blocking allreduce
- Vector Operations:
  - Investigated impact of merged vs separate vector operations for 27-point Poisson matrix with 13k rows per core
  - Showed vector operation speedups of about 30% for PIPE2CG, 25% for PIPECG, and 10% for NBPCG