Avoiding communication in geometric multigrid

Erin C. Carson
Samuel Williams, Michael Lijewski, Nicholas Knight, Ann S. Almgren, James Demmel, and Brian Van Straalen

University of California, Berkeley, USA
Lawrence Berkeley National Laboratory, USA

PMAA, Wednesday, July 2, 2014
Talk overview

• Coarse grid solver ("bottom solver") often the bottleneck in geometric multigrid methods due to high cost of global communication

• Replacing classical solver with communication-avoiding variant can asymptotically reduce global communication

• Implementation, evaluation, and optimization of a communication-avoiding formulation of the Krylov solver routine (CA-BICGSTAB) as a high-performance, distributed-memory bottom solve routine for geometric multigrid

• Bottom solver speedups: 4.2x in miniGMG benchmark, up to 2.5x in real applications

• First use of communication-avoiding Krylov subspace methods for improving multigrid bottom solve performance
Geometric multigrid

- Numerical simulations in a wide array of scientific disciplines require solving elliptic/parabolic PDEs on a hierarchy of adaptively refined meshes
- Geometric multigrid (GMG) is a good choice for many problems
- Consists of a series of V-cycles (“U-cycles”)
  - When further coarsening becomes infeasible, solve distributed coarse grid problem
  - Other options: agglomerate and solve local coarse grid problem, switch to algebraic, etc.
- Krylov subspace methods commonly used for bottom solve routines
  - Only require approximate solve, matrix-free representation
  - GMG + Krylov method available as solver option in many available software packages (e.g., BoxLib, Chombo, PETSc, hypre)
Krylov subspace methods

• Iterative methods based on projection onto expanding subspaces
• In iteration $m$, approximate solution $x_m$ to $Ax = b$ chosen from the expanding **Krylov Subspace**:

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$$

where $r_0 = b - Ax_0$, subject to orthogonality constraints

• Main computational kernels in each iteration:
  – **Sparse matrix-vector multiplication** (SpMV) : Compute new basis vector to increase the dimension of the Krylov subspace
    • **P2P communication** (nearest neighbors)
  – **Inner products**: orthogonalization to select “best” solution
    • **MPI_Allreduce** (global synchronization)

• Examples: Conjugate Gradient (CG), Generalized Minimum Residual Methods (GMRES), Biconjugate Gradient (BICG), BICG Stabilized (BICGSTAB)
The miniGMG benchmark

• Geometric multigrid benchmark from (Williams, et al., 2012)
• Designed to mimic key computational characteristics of applications, present a challenge for exascale architectures
• Uses hybrid MPI+OpenMP
• Finite-volume discretization of variable-coefficient Helmholtz equation ($Lu = aαu − b∇ · β∇u = f$) on a cube, periodic boundary conditions
• Global 3D domain, partitioned into subdomains: one $64^3$ box per MPI process (reflects memory capacity challenges of real AMR MG combustion applications)
• Piecewise constant interpolation, GSRB smoothing in V-cycle
• When box size reduced to $4^3$, restriction terminates, BICGSTAB used as bottom solve routine
miniGMG benchmark results

- miniGMG benchmark with BICGSTAB bottom solve
- Machine: Hopper at NERSC (Cray XE6), 4 6-core Opteron chips per node, Gemini network, 3D torus
- Weak scaling: Up to 4096 MPI processes (1 per chip, 24,576 cores total)
  - $64^3$ points per process ($N = 128^3$ over 48 cores, $N = 1024^3$ over 24,576 cores)
miniGMG benchmark results

- miniGMG benchmark with BICGSTAB bottom solve
- Machine: Hopper at NERSC (Cray XE6), 4 6-core Opteron chips per node, Gemini network, 3D torus
- Weak scaling: Up to 4096 MPI processes (1 per chip, 24,576 cores total)
  - $64^3$ points per process ($N = 128^3$ over 48 cores, $N = 1024^3$ over 24,576 cores)

→ Bottom solve time dominates the runtime of the overall GMG method
→ Scales poorly compared to other parts of the V-cycle
The communication bottleneck

- Same miniGMG benchmark with BICGSTAB on Hopper
- Top: **MPI_AllReduce clearly dominates bottom solve time**
- Bottom: Increase in MPI_AllReduce time due to two effects:
  1. # iterations required by solver increases with problem size
  2. MPI_AllReduce time increases with machine scale (no guarantee of compact subtorus)
- **Poor scalability**: Increasing number of increasingly slower iterations!

---

**Graphs:**
- **Top graph:**
  - X-axis: Processes (6 threads each)
  - Y-axis: Time (seconds)
  - Graphs: Bottom Solver Time (total) and MPI_AllReduce Time (total)

- **Bottom graph:**
  - X-axis: Processes (6 threads each)
  - Y-axis: Time per Iteration in MPI_AllReduce and BiCGStab Iterations (summed over all v-cycles)
The BICGSTAB method

Given: Initial guess $x_0$ for solving $Ax = b$
Initialize $p_0 = r_0 = b - Ax_0$
Pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$
for $j = 0, 1, \ldots$, until convergence do
  $\alpha_j = (\tilde{r}, r_j) / (\tilde{r}, Ap_j)$
  $x_{j+1} = x_j + \alpha_j p_j$
  $q_j = r_j - \alpha_j Ap_j$
  Check $\|q_j\|_2 = (q_j, q_j)^{1/2}$ for convergence
  $\omega_j = (q_j, Aq_j) / (Aq_j, Aq_j)$
  $x_{j+1} = x_{j+1} + \omega_j q_j$
  $r_{j+1} = q_j - \omega_j Aq_j$
  Check $\|r_{j+1}\|_2 = (r_{j+1}, r_{j+1})^{1/2}$ for convergence
  $\beta_j = (\alpha_j / \omega_j)(\tilde{r}, r_{j+1}) / (\tilde{r}, r_j)$
  $p_{j+1} = r_{j+1} + \beta_j (p_j - \omega_j Ap_j)$
end for
The BICGSTAB method

Given: Initial guess $x_0$ for solving $A x = b$
Initialize $p_0 = r_0 = b - Ax_0$
Pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$

for $j = 0,1, \ldots$, until convergence do

\[ \alpha_j = (\tilde{r}, r_j) / (\tilde{r}, Ap_j) \]
\[ x_{j+1} = x_j + \alpha_j p_j \]
\[ q_j = r_j - \alpha_j Ap_j \]

Check $\| q_j \|_2 = (q_j, q_j)^{1/2}$ for convergence

\[ \omega_j = (q_j, Aq_j) / (Aq_j, Aq_j) \]
\[ x_{j+1} = x_{j+1} + \omega_j q_j \]
\[ r_{j+1} = q_j - \omega_j Aq_j \]

Check $\| r_{j+1} \|_2 = (r_{j+1}, r_{j+1})^{1/2}$ for convergence

\[ \beta_j = (\alpha_j / \omega_j) (\tilde{r}, r_{j+1}) / (\tilde{r}, r_j) \]
\[ p_{j+1} = r_{j+1} + \beta_j (p_j - \omega_j Ap_j) \]

end for

Inner products in each iteration require global synchronization (MPI_AllReduce)
Communication-avoiding Krylov methods

- Communication-avoiding Krylov subspace methods (CA-KSMS) can asymptotically reduce parallel latency
- First known reference: $s$-step CG (Van Rosendale, 1983)
  - Many methods and variations created since; see Hoemmen’s 2010 PhD thesis for thorough overview
- Main idea: Block iterations by groups of $s$
Communication-avoiding Krylov methods

• Communication-avoiding Krylov subspace methods (CA-KSMS) can asymptotically reduce parallel latency

• First known reference: $s$-step CG (Van Rosendale, 1983)
  – Many methods and variations created since; see Hoemmen’s 2010 PhD thesis for thorough overview

• Main idea: Block iterations by groups of $s$

• **Outer loop (communication step):**
  – Precompute Krylov basis $V$ (dimension $N$-by-$O(s)$) required to compute next $s$ iterations ($O(s)$ SpMVs, P2P communication)
  – Encode inner products using Gram matrix $G = V^T V$
    • Requires only one MPI_AllReduce to compute information needed for $s$ iterations -> decreases global synchronizations by $O(s)$!
Communication-avoiding Krylov methods

• Communication-avoiding Krylov subspace methods (CA-KSMS) can **asymptotically reduce parallel latency**

• First known reference: $s$-step CG (Van Rosendale, 1983)
  – Many methods and variations created since; see Hoemmen’s 2010 PhD thesis for thorough overview

• Main idea: Block iterations by groups of $s$

• **Outer loop (communication step):**
  – Precompute Krylov basis $V$ (dimension $N$-by-$O(s)$) required to compute next $s$ iterations ($O(s)$ SpMVs, P2P communication)
  – Encode inner products using Gram matrix $G = V^T V$
    • Requires only one MPI_AllReduce to compute information needed for $s$ iterations - $\rightarrow$ decreases global synchronizations by $O(s)$!

• **Inner loop (computation steps):**
  – Update length-$O(s)$ vectors that represent coordinates of BICGSTAB vectors in $V$ for the next $s$ iterations, use $G$ to recover inner products locally- no further communication required!
\[ G = V^T V \]
\[ G = V^T V \]

\[ \|r\|^2 = \]

\[ r^T r = r'^T G r' \]
\[ G = V^T V \]

\[ \|r\|^2 = r^T r = r'^T G r' \]

Computed locally on each processor – no communication!
The CA-BICGSTAB method

[C., Knight, Demmel. SISC 35(5), 2013.]

Given: Initial guess $x_0$ for solving $Ax = b$

Initialize $p_0 = r_0 = b - Ax_0$, pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$

Construct $(4s + 1)$-by-$(4s + 1)$ matrix $T$

for $m = 0, s, 2s, \ldots$, until convergence do

Compute $V$ w/columns a basis for $\mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m)$

Compute $[G, g] = V^T [V, \tilde{r}]$

for $j = 0, 1, \ldots, s - 1$ do

\begin{align*}
\alpha_{m+j} &= (g, r_j')/(\tilde{r}, Tp_j') \\
x_{j+1}' &= x_j' + \alpha_{m+j}p_j' \\
q_j' &= r_j' - \alpha_{m+j}Tp_j' \\
\text{Check}\ ||q_j'||_2 = (q_j', Gq_j')^{1/2} \text{ for convergence} \\
\omega_{m+j} &= (q_j', GTq_j')/(Tq_j', Tq_j') \\
x_{j+1}' &= x_{j+1}' + \omega_{m+j}q_j' \\
r_{j+1}' &= q_j' - \omega_{m+j}Tq_j' \\
\text{Check}\ ||r_{j+1}'||_2 = (r_{j+1}', Gr_{j+1}')^{1/2} \text{ for convergence} \\
\beta_{m+j} &= (\alpha_{m+j}/\omega_{m+j})(g, r_{j+1}')/(g, r_j') \\
p_{j+1}' &= r_{j+1}' + \beta_{m+j}(p_j' - \omega_{m+j}Tp_j')
\end{align*}

end for

$[p_{m+s}, r_{m+s}, x_{m+s} - x_m] = V[p_s', r_s', x_s']$

end for
The CA-BICGSTAB method

Given: Initial guess $x_0$ for solving $Ax = b$
Initialize $p_0 = r_0 = b - Ax_0$, pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$
Construct $(4s + 1)$-by-$(4s + 1)$ matrix $T$
for $m = 0, s, 2s, \ldots$, until convergence do
Compute $V$ w/columns a basis for $\mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m)$
Compute $[G, g] = V^T[V, \tilde{r}]$
for $j = 0, 1, \ldots, s - 1$ do
\[
\alpha_{m+j} = (g, r_j')/(\tilde{r}, Tp_j')
\]
\[
x_{j+1}' = x_j' + \alpha_{m+j}p_j'
\]
\[
q_j' = r_j' - \alpha_{m+j}Tp_j'
\]
Check $\|q_j\|_2 = (q_j', Gq_j')^{1/2}$ for convergence
\[
\omega_{m+j} = (q_j', GTq_j')/(Tq_j', Tq_j')
\]
\[
x_{j+1}' = x_{j+1}' + \omega_{m+j}q_j'
\]
\[
r_{j+1}' = q_j' - \omega_{m+j}Tq_j'
\]
Check $\|r_{j+1}\|_2 = (r_{j+1}', Gr_{j+1}')^{1/2}$ for convergence
\[
\beta_{m+j} = (\alpha_{m+j}/\omega_{m+j})(g, r_{j+1}')/(g, r_j')
\]
\[
p_{j+1}' = r_{j+1}' + \beta_{m+j}(p_j' - \omega_{m+j}Tp_j')
\]
end for
end for

$[p_{m+s}, r_{m+s}, x_{m+s} - x_m] = V[p_s', r_s', x_s']$
The CA-BICGSTAB method

Given: Initial guess $x_0$ for solving $Ax = b$

Initialize $p_0 = r_0 = b - Ax_0$, pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$

Construct $(4s + 1)$-by-$(4s + 1)$ matrix $T$

for $m = 0, s, 2s, \ldots$, until convergence do

Compute $V$ w/columns a basis for $\mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m)$

Compute $[G, g] = V^T [V, \tilde{r}]$

for $j = 0, 1, \ldots, s - 1$ do

\[
\alpha_{m+j} = (g, r_j') / (\tilde{r}, Tp_j')
\]

\[
x_{j+1}' = x_j' + \alpha_{m+j} p_j'
\]

\[
q_j' = r_j' - \alpha_{m+j} Tp_j'
\]

Check $\|q_j\|_2 = (q_j', Gq_j')^{1/2}$ for convergence

\[
\omega_{m+j} = (q_j', GTq_j') / (Tq_j', Tq_j')
\]

\[
x_{j+1}' = x_{j+1}' + \omega_{m+j} q_j'
\]

\[
r_{j+1}' = q_j' - \omega_{m+j} Tq_j'
\]

Check $\|r_{j+1}'\|_2 = (r_{j+1}', Gr_{j+1}')^{1/2}$ for convergence

\[
\beta_{m+j} = (\alpha_{m+j} / \omega_{m+j})(g, r_{j+1}') / (g, r_j')
\]

\[
p_{j+1}' = r_{j+1}' + \beta_{m+j} (p_j' - \omega_{m+j} Tp_j')
\]

end for

end for

$[p_{m+s}, r_{m+s}, x_{m+s} - x_m] = V [p_s', r_s', x_s']$
The CA-BICGSTAB method

[C., Knight, Demmel. SISC 35(5), 2013.]

Given: Initial guess $x_0$ for solving $Ax = b$
Initialize $p_0 = r_0 = b - Ax_0$, pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$
Construct $(4s + 1)$-by-$(4s + 1)$ matrix $T$

for $m = 0, s, 2s, \ldots$, until convergence do

Compute $V$ w/columns a basis for $K_{2s+1}(A, p_m) + K_{2s}(A, r_m)$
Compute $[G, g] = V^T[V, \tilde{r}]$

for $j = 0, 1, \ldots, s - 1$ do

$\alpha_{m+j} = (g, r_j')/(\tilde{r}, Tp_j')$
$x_{j+1}' = x_j' + \alpha_{m+j}p_j'$
$q_j' = r_j' - \alpha_{m+j}Tp_j'$
Check $\|q_j\|_2 = (q_j', Gq_j')^{1/2}$ for convergence
$\omega_{m+j} = (q_j', GTq_j')/(Tq_j', Tq_j')$
$x_{j+1}' = x_{j+1}' + \omega_{m+j}q_j'$
$r_{j+1}' = q_j' - \omega_{m+j}Tq_j'$
Check $\|r_{j+1}\|_2 = (r_{j+1}', Gr_{j+1}')^{1/2}$ for convergence
$\beta_{m+j} = (\alpha_{m+j}/\omega_{m+j})(g, r_{j+1}')/(g, r_j')$
$p_{j+1}' = r_{j+1}' + \beta_{m+j}(p_j' - \omega_{m+j}Tp_j')$

end for

$[p_{m+s}, r_{m+s}, x_{m+s} - x_m] = V[p_s', r'_s, x'_s]$

end for
Speedups for synthetic benchmark

- Time (left) and performance (right) of miniGMG benchmark with BICGSTAB vs. CA-BICGSTAB with $s = 4$ (monomial basis) on Hopper.
- At 24K cores, CA-BICGSTAB’s asymptotic reduction of calls to MPI_AllReduce improves bottom solver by 4.2x, overall multigrid solve by nearly 2.5x.
Speedups for synthetic benchmark

- Time (left) and performance (right) of miniGMG benchmark with BICGSTAB vs. CA-BICGSTAB with $s = 4$ (monomial basis) on Hopper
- Aggregate MG solve performance using CA-BICGSTAB much closer to linear in DOF/s
Benchmark timing breakdown

- Plot: Net time spent across all bottom solves at 24,576 cores, for BICGSTAB and CA-BICGSTAB with $s = 4$

![Breakdown of Bottom Solver]

- 11.2x reduction in MPI_AllReduce time (red) – BICGSTAB requires $6s$ more MPI_AllReduce’s than CA-BICGSTAB
- Less than theoretical 24x since messages in CA-BICGSTAB are larger, not always latency-limited
- P2P (blue) communication doubles for CA-BICGSTAB – Basis computation requires twice as many SpMVs (P2P) per iteration as BICGSTAB
Benchmark timing breakdown

- Plot: Net time spent across all bottom solves at 24,576 cores, for BICGSTAB and CA-BICGSTAB with $s = 4$

- **11.2x reduction in MPI_AllReduce time (red)**
  - BICGSTAB requires 6 times more MPI_AllReduce's than CA-BICGSTAB
  - Less than theoretical 24x since messages in CA-BICGSTAB are larger, not always latency-limited
Benchmark timing breakdown

- Plot: Net time spent across all bottom solves at 24,576 cores, for BICGSTAB and CA-BICGSTAB with $s = 4$
- **11.2x reduction in MPI_AllReduce time (red)**
  - BICGSTAB requires 6x more MPI_AllReduce’s than CA-BICGSTAB
  - Less than theoretical 24x since messages in CA-BICGSTAB are larger, not always latency-limited
- **P2P (blue) communication doubles** for CA-BICGSTAB
  - Basis computation requires twice as many SpMVs (P2P) per iteration as BICGSTAB

![Breakdown of Bottom Solver](image-url)
Speedups for real applications

• CA-BICGSTAB bottom-solver implemented in BoxLib (AMR framework from LBL)
• Compared GMG with BICGSTAB vs. GMG with CA-BICGSTAB for two different applications:

Low Mach Number Combustion Code (LMC): gas-phase combustion simulation

Nyx: 3D N-body and gas dynamics code for cosmological simulations of dark matter particles
Speedups for real applications

- CA-BICGSTAB bottom-solver implemented in BoxLib (AMR framework from LBL)
- Compared GMG with BICGSTAB vs. GMG with CA-BICGSTAB for two different applications:

**Low Mach Number Combustion Code (LMC):** gas-phase combustion simulation
- Up to **2.5x speedup in bottom solve; up to 1.5x in MG solve**

**Nyx:** 3D N-body and gas dynamics code for cosmological simulations of dark matter particles
Speedups for real applications

- CA-BICGSTAB bottom-solver implemented in BoxLib (AMR framework from LBL)
- Compared GMG with BICGSTAB vs. GMG with CA-BICGSTAB for two different applications:

**Low Mach Number Combustion Code (LMC):** gas-phase combustion simulation
- Up to **2.5x speedup in bottom solve; up to 1.5x in MG solve**

**Nyx:** 3D N-body and gas dynamics code for cosmological simulations of dark matter particles
- Up to **2x speedup in bottom solve, 1.15x in MG solve**
Discussion and challenges

- For practical purposes, choice of $s$ limited by finite precision error
  - As $s$ is increased, convergence slows – more required iterations can negate any speedup per iteration gained
  - Can use better-conditioned Krylov bases, but this can incur extra cost (esp. if $A$ changes b/t V-cycles); ongoing work
  - In our tests, $s = 4$ with the monomial basis gave similar convergence for BICGSTAB and CA-BICGSTAB

- Some bottom-solves are “harder” – take more iterations to converge – than others
  - Implemented “telescoping $s$” approach
    - Outer loops begin using $s = 1$, increase up in subsequent outer loops up to $s = 4$
    - Ensures that easy solves don’t incur extra costs of computing $V$ and $G = V^T V$, hard solves see asymptotic benefits
Discussion and challenges

• Timing breakdown shows we must consider tradeoffs between bandwidth, latency, and computation when optimizing for particular problem/machine

  – Blocking BICGSTAB inner products most beneficial when:
    • MPI_AllReduces in bottom solve are dominant cost in GMG solve, GMG solves are dominant cost of application
    • Bottom solve requires enough iterations to amortize extra costs (bigger MPI messages, more P2P communication)

  – CA-BICGSTAB can also be optimized to reduce P2P communication or reduce vertical data movement when computing Krylov bases (“matrix powers kernel”)
Design space and related approaches

- **Parallelizing multigrid methods**
  - Concurrent iterations, multiple coarse corrections, full domain partitioning, block factorization, etc.

- **Solving the coarse grid problem**
  - Type of solver: direct (e.g., LU), stationary iterative (e.g., Jacobi), Krylov, switch to algebraic
  - Coarse grid agglomeration - use only a subset of available processors

- **Reducing communication cost in Krylov subspace methods**
  - Pipelining: overlap nonblocking reductions with matrix-vector multiplications (Ghysels, Ashby, Meerbergen, Vanroose, 2013)
  - Tiling approach to reduce communication bottleneck in Chebyshev iteration smoothers (Ghysels, Kłosiewicz, Vanroose, 2012)
  - Overlap global synchronization points with SpMV and preconditioner application (Gropp, 2010)
  - Delayed reorthogonalization: avoid synchronization due to reorthogonalization (ADR in SLEPc) (Hernandez, Román, Tomás, 2007)
Summary and future work

• Implemented, evaluated, and optimized CA-BICGSTAB as a high-performance, distributed-memory bottom solve routine for geometric multigrid solvers
  – GMG+CABICGSTAB available as option in miniGMG, BoxLib, and CHOMBO frameworks

• Expands the design space: trade collective latency for bandwidth, trade fine-grained operations for one coarse-grained operation that expresses more parallelism

• Future work:
  – Exploration of design space for other Krylov solvers, other architectures, other applications
  – Implementation of different polynomial bases for Krylov subspace to improve convergence for higher $s$ values
  – Improve accessibility of communication-avoiding Krylov methods through scientific computing libraries and frameworks
Summary and future work

- Implemented, evaluated, and optimized CA-BICGSTAB as a high-performance, distributed-memory bottom solve routine for geometric multigrid solvers
  - GMG+CABICGSTAB available as option in miniGMG, BoxLib, and CHOMBO frameworks
- **Expands the design space**: trade collective latency for bandwidth, trade $s$ fine-grained operations for one coarse-grained operation that expresses more parallelism
Summary and future work

• Implemented, evaluated, and optimized CA-BICGSTAB as a high-performance, distributed-memory bottom solve routine for geometric multigrid solvers
  – GMG+CABICGSTAB available as option in miniGMG, BoxLib, and CHOMBO frameworks

• **Expands the design space**: trade collective latency for bandwidth, trade $s$ fine-grained operations for one coarse-grained operation that expresses more parallelism

• Future work:
  – **Exploration of design space** for other Krylov solvers, other architectures, other applications
  – Implementation of different polynomial bases for Krylov subspace to **improve convergence** for higher $s$ values
  – **Improve accessibility** of communication-avoiding Krylov methods through scientific computing libraries and frameworks
Thank you!
Email: erin@cs.berkeley.edu
References


BoxLib website. https://ccse.lbl.gov/BoxLib.


Extra Slides
Communication is expensive!

- Algorithms have two costs: **communication** and **computation**
  - **Communication**: moving data between levels of memory hierarchy (sequential), between processors (parallel)

- On modern computers, communication is expensive, computation is cheap
  - Flop time $<< 1$/bandwidth $<<$ latency
  - Communication bottleneck a barrier to achieving scalability
  - Communication is also expensive in terms of energy cost

- For scalability, we **must redesign algorithms to avoid communication**
Coarse Grid Agglomeration

- Unite subdomains onto a subset of the available processors once ratio of interior nodes to boundary nodes falls below some threshold
- Pros: Reduces communication required to perform operations at this level and lower levels
- Cons: Leaves processors idle, requires lots of data movement (scatter/gather) to perform data redistribution

- Words moved = $2 \cdot O(n^2)$ for scatter/gather, versus $\frac{\#its}{s} \cdot O(s^2 \log p)$ for MPI_AllReduce in CA coarse grid solves

- This approach could be combined with coarse grid solve — do agglomeration at some level, then coarse grid solve at this level or lower
  - Expect CA-KSMs to have less speedup over KSMs in this case, since communication in bottom solve less expensive due to more processors
    - But not clear that this would always be the winning approach in terms of overall runtime
- Best approach will depend on the parallel environment and application!
**Weak-scaled, Krylov Solvers**

- Optimize the construction of $[P,R,rt]$ to reduce DRAM data movement

**Weak-scaled, MG Bottom Solves**

- Optimize $[G,g]=[P,R]^T[P,R,r]$ to minimize collectives

**Strong-scaled Krylov Solvers**

- Optimize $[P,R,r]$ to reduce the number of MPI messages
Krylov Subspace Methods

- A Krylov Subspace is defined as 
  \[ \mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\} \]
- A Krylov Subspace Method is a projection process onto the subspace \( \mathcal{K} \) orthogonal to \( \mathcal{L} \)
  - The choice of \( \mathcal{L} \) distinguishes the various methods
  - Examples: Conjugate Gradient (CG), Generalized Minimum Residual Methods (GMRES), Biconjugate Gradient (BICG), BICG Stabilized (BICGSTAB)

For linear systems, in iteration \( m \), approximates solution \( x_m \) to \( Ax = b \) by imposing the condition

\[ x_m = x_0 + \delta, \; \delta \in \mathcal{K}_m \; \text{and} \; r_0 - A\delta \perp \mathcal{L}_m, \]

where \( r_0 = b - Ax_0 \)
Hopper

- Cray XE6 MPP at NERSC
- Each compute node that four 6-core Opteron chips each with two DDR3-1333 mem controllers
- Each superscalar out-of-order core: 64KB L1, 512KB L2. One 6MB L3 cache per chip
- Pairs of compute nodes connected via HyperTransport to a high-speed Gemini network chip
- Gemini network chips connected to form high-BW low-latency 3D torus
  - Some asymmetry in torus
  - No control over job placement
  - Latency can be as low as 1.5 microseconds, but typically longer in practice
Convergence rates with $s = 4$
## Related Work: $s$-step methods

<table>
<thead>
<tr>
<th>Authors</th>
<th>KSM</th>
<th>Basis</th>
<th>Precond?</th>
<th>Mtx Pwrs?</th>
<th>TSQR?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Van Rosendale, 1983</td>
<td>CG</td>
<td>Monomial</td>
<td>Polynomial</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Leland, 1989</td>
<td>CG</td>
<td>Monomial</td>
<td>Polynomial</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Walker, 1988</td>
<td>GMRES</td>
<td>Monomial</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Chronopoulos and Gear, 1989</td>
<td>CG</td>
<td>Monomial</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Chronopoulos and Kim, 1990</td>
<td>Orthomin, GMRES</td>
<td>Monomial</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Chronopoulos, 1991</td>
<td>MINRES</td>
<td>Monomial</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Kim and Chronopoulos, 1991</td>
<td>Symm. Lanczos, Arnoldi</td>
<td>Monomial</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sturler, 1991</td>
<td>GMRES</td>
<td>Chebyshev</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
## Related Work: \( s \)-step methods

<table>
<thead>
<tr>
<th>Authors</th>
<th>KSM</th>
<th>Basis</th>
<th>Precond?</th>
<th>Mtx Pwrs?</th>
<th>TSQR?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joubert and Carey, 1992</td>
<td>GMRES</td>
<td>Chebyshev</td>
<td>No</td>
<td>Yes*</td>
<td>No</td>
</tr>
<tr>
<td>Chronopoulos and Kim, 1992</td>
<td>Nonsymm. Lanczos</td>
<td>Monomial</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Bai, Hu, and Reichel, 1991</td>
<td>GMRES</td>
<td>Newton</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Erhel, 1995</td>
<td>GMRES</td>
<td>Newton</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>de Sturler and van der Vorst, 2005</td>
<td>GMRES</td>
<td>Chebyshev</td>
<td>General</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Toledo, 1995</td>
<td>CG</td>
<td>Monomial</td>
<td>Polynomial</td>
<td>Yes*</td>
<td>No</td>
</tr>
<tr>
<td>Chronopoulos and Swanson, 1990</td>
<td>CGR, Orthomin</td>
<td>Monomial</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Chronopoulos and Kinkaid, 2001</td>
<td>Orthodir</td>
<td>Monomial</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Many previously derived $s$-step Krylov methods

- First known reference is (Van Rosendale, 1983)

Motivation: minimize I/O, increase parallelism

Empirically found that monomial basis for $s > 5$ causes instability

Many found better convergence using better conditioned polynomials based on spectrum of $A$ (e.g., scaled monomial, Newton, Chebyshev)

Hoemmen et al. (2009) first to produce CA implementations that also avoid communication for general sparse matrices (and use TSQR)

- Speedups for various matrices for a fixed number of iterations

  - Shows that $\left( \frac{\text{time per } s \text{ iterations KSM}}{\text{time per outer iteration CA–KSM}} \right)$ can be $O(s)$
Parallel Multigrid Methods

• Concurrent iterations
  – Reduces time per multigrid iteration by performing relaxation sweeps on all grids simultaneously

• Multiple coarse corrections
  – Accelerates convergence by projecting the fine grid system onto several different coarse grid spaces

• Full domain partitioning
  – Reduces communication during refinement stage

• Block factorization
  – Use a special selection of coarse and fine points to expose parallelism
The BICGSTAB method

Given: Initial guess $x_0$ for solving $Ax = b$
Initialize $p_0 = r_0 = b - Ax_0$
Pick arbitrary $\tilde{r}$ such that $(\tilde{r}, r_0) \neq 0$
for $j = 0, 1, \ldots$ , until convergence do

\[ \alpha_j = (\tilde{r}, r_j)/(\tilde{r}, Ap_j) \]
\[ x_{j+1} = x_j + \alpha_j p_j \]
\[ q_j = r_j - \alpha_j Ap_j \]
Check $\|q_j\|^2 = (q_j, q_j)^{1/2}$ for convergence

\[ \omega_j = (q_j, Aq_j)/(Aq_j, Aq_j) \]
\[ x_{j+1} = x_{j+1} + \omega_j q_j \]
\[ r_{j+1} = q_j - \omega_j Aq_j \]
Check $\|r_{j+1}\|^2 = (r_{j+1}, r_{j+1})^{1/2}$ for convergence

\[ \beta_j = (\alpha_j/\omega_j)(\tilde{r}, r_{j+1})/(\tilde{r}, r_j) \]
\[ p_{j+1} = r_{j+1} + \beta_j (p_j - \omega_j Ap_j) \]
end for

Inner products in each iteration require global synchronization (MPI_AllReduce)
Multiplication by A requires nearest neighbor communication (P2P)
CA-BICGSTAB derivation: Basis change

Suppose we are at some iteration \( m \). What are the dependencies on \( r_m, p_m, \) and \( x_m \) for computing the next \( s \) iterations?

By induction, for \( j = \{0, 1, \ldots, s - 1\} \)

\[
p_{m+j+1}, r_{m+j+1}, x_{m+j+1} - x_m \in \mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m)\\
q_{m+j} \in \mathcal{K}_{2s}(A, p_m) + \mathcal{K}_{2s-1}(A, r_m)\\
p_{m+j} \in \mathcal{K}_{2s-1}(A, p_m) + \mathcal{K}_{2s-2}(A, r_m)
\]

Let \( P \) and \( R \) be bases for \( \mathcal{K}_{2s+1}(A, p_m) \) and \( \mathcal{K}_{2s}(A, r_m) \), respectively.

For the next \( s \) iterations \( (j = \{0, 1, \ldots, s - 1\}) \),

\[
r_{m+j+1} = [P, R]r'_{j+1} \quad p_{m+j+1} = [P, R]p'_{j+1}\\
x_{m+j+1} - x_m = [P, R]x'_{j+1} \quad q_{m+j} = [P, R]q'_{j}
\]

i.e., length-\((4s + 1)\) vectors \( r'_{j+1}, p'_{j+1}, x'_{j+1}, \) and \( q'_{j} \) are coordinates for the length-\(N\) vectors \( r_{m+j+1}, p_{m+j+1}, x_{m+j+1} - x_m, \) and \( q_{m+j} \) respectively, in bases \([P, R]\).
CA-BICGSTAB derivation: Coordinate updates

The bases $P, R$ are generated by polynomials satisfying 3-term recurrence represented by $(4s + 1)$-by-$(4s + 1)$ tridiagonal matrix $T$ satisfying

$$A[P, 0_{N, 1}, R, 0_{N, 1}] = [P, R]T$$

where $P, R$ are $P, R$ resp. with last columns omitted

Multiplications by $A$ can then be written:


Update BICGSTAB vectors by updating their coordinates in $[P, R]$:

$$x'_{j+1} = x'_j + \alpha_{m+j} p'_j$$
$$q'_j = r'_j - \alpha_{m+j} Tp'_j$$
$$x'_{j+1} = x'_{j+1} + \omega_{m+j} q'_j$$
$$r'_{j+1} = r'_j - \omega_{m+j} Tq'_j$$
$$p'_{j+1} = r'_{j+1} + \beta_{m+j} (p'_j - \omega_{m+j} Tp'_j)$$

Each process stores $T$ locally, redundantly compute coordinate updates
CA-BICGSTAB derivation: Inner products

Last step: rewriting length-$N$ inner products in the new Krylov basis. Let


\[ g = [P, R]^T \tilde{r} \]

where the “Gram Matrix” $G$ is $(4s + 1)$-by-$(4s + 1)$ and $g$ is a $(4s + 1)$ vector. (Note: can be computed with one MPI_AllReduce by $[P, R]^T [P, R, \tilde{r}]$).

Then all the dot products for $s$ iterations of BICGSTAB can be computed locally in CA-BICGSTAB using $G$ and $g$ by the relations

\[ (\tilde{r}, r_{m+j}) = (g, r'_j) \]
\[ (\tilde{r}, r_{m+j+1}) = (g, r'_j) \]
\[ (\tilde{r}, Ap_{m+j}) = (g, Tp'_j) \]
\[ (q_{m+j}, A q_{m+j}) = (q'_j, GT q'_j) \]
\[ (A q_{m+j}, A q_{m+j}) = (T q'_j, GT q'_j) \]

Note: norms for convergence checks can be estimated with no communication in a similar way, e.g., $\|r_{m+j+1}\|_2 = (r_{m+j+1}, r_{m+j+1}) = (r'_{j+1}, Gr'_{j+1})$. 
\[ G = V^T V \]

\[
\begin{align*}
\|r\|^2 &= r^T r \\
&= r' \check{V} r'
\end{align*}
\]

Computed locally on each processor – no communication!