Reducing Communication in Parallel Algebraic Multigrid

Ulrike Meier Yang

Workshop on Novel Numerical Methods, July 29-31, 2013
Motivation: Scalable Solvers

- Multigrid solvers are essential components of LLNL simulation
- Multigrid solvers are optimal (O(N) operations)
- Scalable $\rightarrow$ faster simulations $\rightarrow$ better science!
Motivation: Scalable Solvers

- Multigrid solvers are essential components of LLNL simulation
- Multigrid solvers are optimal (O(N) operations)
- Scalable → faster simulations → better science!

- Loss of Scalability on computers with less efficient networks
- Concerns about scalability on future architectures
Algebraic Multigrid (AMG)

- iterative method for solving $Ax=b$
- commonly used as a preconditioner

**Setup phase:**

- Select coarse “grids”
- Define interpolation
  \[ P^{(m)}, m = 1, \ldots \]
- Define restriction
  \[ R^{(m)} = P^{(m)T} \]
- Define coarse-grid operators:
  \[ A^{(m+1)} = P^{(m)T} A^{(m)} P^{(m)} \]

**Solve phase:**

- iterative method for solving $Ax=b$
- commonly used as a preconditioner
Performance degradation caused by increased communication complexity on coarser grids!
Various Efforts to Reduce Communication in AMG

- Use of redundancy and agglomeration on coarser grids (Gahvari, Gropp, Jordan, Schulz, Yang)
- Additive AMG methods (Vassilevski, Yang)
Redundant Coarse Grid Solves

Now use parallel AMG on k cores for each redundant coarse solve
Redundant Coarse Grid Solves

Now use parallel AMG on $k$ cores for each redundant coarse solve

- Illustration of chunk data distribution using 4 chunks
- Cores in each chunk perform the same operations
- Communication across chunks
- Allows better use of cache
- Allows smart choice of cores
- Use performance model to determine when to switch
Performance model for the AMG solve cycle

- Model parameters: $\alpha$ latency, $\beta$ inverse bandwidth
- Take architectural features into account with penalties
  - **Distance of communication**: add time per hop $\gamma$
  - **Lower effective bandwidth**: let $m$ = # msgs, $l$ = # links, $B_{HW}$ = hardware bandwidth, $B_{MPI}$ = MPI bandwidth
    - multiply $\beta$ by $(B_{HW}/B_{MPI} + m/l)$
  - **Multicore penalties**: let $c$ = # cores per node, $P_i$ = # active processes on level $i$, $P$ = # processes
    - Multicore latency penalty: multiply $\alpha$ by $cP/P$
    - Multicore distance penalty: multiply $\gamma$ by $cP/P$
Inclusion of Performance Model into AMG

- Substantial speedups on Hera for solving various 3D (30 x 30 x 30 points/core) and 2D (150 x 150 points/core) problems:

<table>
<thead>
<tr>
<th>3D Problems</th>
<th>64 Cores</th>
<th>512 Cores</th>
<th>4096 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>7pt Laplace</td>
<td>1.36</td>
<td>2.74</td>
<td>2.56</td>
</tr>
<tr>
<td>27pt Stencil</td>
<td>1.62</td>
<td>2.47</td>
<td>2.40</td>
</tr>
<tr>
<td>Convection-diffusion</td>
<td>1.24</td>
<td>2.04</td>
<td>2.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2D Problems</th>
<th>64 Cores</th>
<th>256 Cores</th>
<th>1024 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>9pt Laplace</td>
<td>1.43</td>
<td>1.28</td>
<td>1.40</td>
</tr>
<tr>
<td>Rotated anis. (0.01, 30°)</td>
<td>1.49</td>
<td>1.40</td>
<td>1.55</td>
</tr>
</tbody>
</table>

- Here, communication was kept on-node during redundant phase
Conclusions/Future Work

- We have enhanced data gathering approach to AMG through
  1. Placing data in ways that localize communication
  2. Using a performance model to automate the switching decision

- Modeling:
  • Develop to the point where model predicts ideal MPI/OpenMP mix (it’s not there yet)

- Data redistribution algorithm:
  • Redundant vs. nonredundant data gathering
  • Other communication localizations
  • Model can guide these

- Performance predictions:
  • Find a strategy for predicting performance on more complicated problems, not just a simple test problem
Additive AMG

- Originally invented in the 80’s to increase parallelism in AMG (Greenbaum, 1986, BPX, Bastian, Hackbusch, Wittum, 1998, …)
- Generally leads to increased number of iterations
- But: shows potential for reduced communication
Multiplicative Multigrid

- Smooth
- Restrict
- Smooth
- Prolong
- Smooth
- Restrict
- Smooth
- Prolong
- Smooth
- Restrict
- Smooth
- Restrict
- Solve

Additive Multigrid

Perform in parallel

- Smooth
- Restrict
- Smooth
- Restrict
- Smooth
- Restrict
- Smooth
- Restrict
- Smooth
- Solve
- Prolong
- Prolong
- Prolong
- Prolong
Opportunity for Performance Improvement

Performance profile of AMG solve cycle for 64 MPI tasks on Hera

**computation**

**idle time**

**MPI calls**

**Cannot take advantage of parallelism, but ...**

Most communication generated by coarse grid operators, not by interpolation or restriction

too little computation compared to communication on coarse grid, prohibiting overlap
Opportunity for Performance Improvement

Performance profile of AMG solve cycle for 64 MPI tasks on Hera

- **computation**
- **idle time**
- **MPI calls**

**Cannot take advantage of parallelism, but …**

- Most communication generated by coarse grid operators, not by interpolation or restriction
- too little computation compared to communication on coarse grid, prohibiting overlap

Combine communication

- smooth
- smooth
- smooth
- **solve**
Another definition of an AMG V-cycle

- Create hierarchy of operators:
  - Matrices $A_k$, $k = 0, \ldots, \ell$
  - Interpolation matrices $P_{k+1}^k$, $k = 0, \ldots, \ell - 1$
  - Smoothers $M_k$, $k = 0, \ldots, \ell$

- A multiplicative V-cycle can now be defined by recursion as follows:
  - $B_{\ell} = A_{\ell}^{-1}$
  - $B_k = \overline{M}_k^{-1} + (I - M_k^{-T}A_k)P_{k+1}^k B_{k+1}(P_{k+1}^k)^T (I - A_k M_k^{-1})$, for $k < \ell$ with $\overline{M}_k = M_k(M_k + M_k^T - A_k)^{-1}M_k^T$
Additive AMG V-cycles

- Form smoothed interpolation
  \[ \bar{P}^{k}_{k+1} = (I - M_k^{-T} A_k) P^{k}_{k+1} \]

- Form inverse of symmetrized smoother
  \[ \Lambda_k = \bar{M}_k^{-1} = M_k^{-1} + M_k^{-T} - M_k^{-T} A_k M_k^{-1} \]

- Define composite interpolation for \( j > k \)
  \[ \bar{P}^{k}_j = \bar{P}^{k}_{k+1} \bar{P}^{k-1}_k \ldots \bar{P}^{j-1}_j \]

- Then
  \[ B_k = \Lambda_k + \sum_{j=0}^{\ell} \bar{P}^{k}_j \Lambda_j (\bar{P}^{k}_j)^T \quad \text{and} \]

  \[ B_0 = \sum_{j=0}^{\ell} \bar{P}^{0}_j \Lambda_j (\bar{P}^{0}_j)^T \]
Proposition 1:
If \( B_0 = \sum_{j=0}^{\ell} P_j^0 \Lambda_j (P_j^0)^T \) is a convergent method, and \( \Lambda_j \) s.p.d. with \( v^T \Lambda_j v \geq v^T \Lambda_{j'} v \) for all \( v \), then \( \bar{B}_0 = \sum_{j=0}^{\ell} P_j^0 \Lambda_j (P_j^0)^T \) is also convergent.

Corollary:
If \( \Lambda_j = M_j^{-1} \) fulfills the conditions above, the simplified method
\[
\bar{B}_0 = \sum_{j=0}^{\ell} P_j^0 M_j^{-1} (P_j^0)^T
\]
is also convergent.
### Multiplicative AMG

\[ r_0 = b - A_0 x_0 \]

For \( k = 1, \ldots, l \), \( x_k = 0 \)

For \( k = 0, \ldots, l - 1 \)

\[ x_k := x_k + M_k^{-1}(r_k) \]

\[ r_{k+1} = r_k + (P_{k+1}^k)^T (r_k - A_k x_k) \]

\[ x_l := x_l + M_l^{-1}(r_l) \]

\[ x_l := x_l + M_l^{-T} (r_l - A_l x_l) \]

For \( k = l - 1, \ldots, 0 \)

\[ x_k := x_k + P_{k+1}^k x_{k+1} \]

\[ x_k := x_k + M_k^{-T} (r_k - A_k x_k) \]

### Mult-Additive AMG

\[ r_0 = b - A_0 x_0 \]

For \( k = 1, \ldots, l \), \( x_k = 0 \)

For \( k = 0, \ldots, l - 1 \)

\[ r_{k+1} = r_k + (\bar{P}_{k+1}^k)^T r_k \]

For \( k = 0, \ldots, l \)

\[ x_k := x_k + M_k^{-1}(r_k) \]

\[ x_k := x_k + M_k^{-T} (r_k - A_k x_k) \]

For \( k = l - 1, \ldots, 0 \)

\[ x_k := x_k + \bar{P}_{k+1}^k x_{k+1} \]

Both algorithms are equivalent for \( \bar{P}_{k+1}^k = (I - M_k^{-T} A_k) P_{k+1}^k \)

(Vassilevski)
Use of Jacobi Smoothing

- Jacobi smoother suitable, since it is cheap

\[ M_k = \tilde{D}_k , \]
we consider here \( \ell^1 \)-Jacobi with \( \tilde{d}_{ii}^k = \sum_{m=0}^{n_k} |a_{im}^k| \)

- Note that

\[ x_k := x_k + M_k^{-1}(r_k) , \quad x_k := x_k + M_k^{-T}(r_k - A_k x_k) \]
equivalent to

\[ x_k := x_k + \Lambda_k r_k \text{ with } \Lambda_k = M_k^{-T}(M_k + M_k^T - A_k)M_k^{-1} \]

\[ \Lambda_k = 2\tilde{D}_k^{-1} - \tilde{D}_k^{-1}A_k\tilde{D}_k^{-1} \Rightarrow \text{nnz}(\Lambda_k) = \text{nnz}(A_k) \]

- Smoothing portion can be expressed as 1 large MatVec: \( x := x + \Lambda r \)

- **Simplified mult-additive method**: replace \( \Lambda_k \) by \( \tilde{D}_k^{-1} \)
  (convergent if mult-additive method converges, Proposition 1)
Comparison of multiplicative and mult-additive version

Assume $M_k$ requires no communication

\[ r_0 = b - A_0 x_0 \]

For $k = 1, \ldots, l$, $x_k = 0$

For $k = 0, \ldots, l - 1$

\[ x_k := x_k + M_k^{-1}(r_k) \]
\[ r_{k+1} = r_k + (P_{k+1}^k)^T(r_k - A_k x_k) \]

\[ x_l := x_l + M_l^{-1}(r_l) \]
\[ x_l := x_l + M_l^{-T}(r_l - A_l x_l) \]

For $k = l - 1, \ldots, 0$

\[ x_k := x_k + P_{k+1}^k x_{k+1} \]
\[ x_k := x_k + M_k^{-T}(r_k - A_k x_k) \]
Comparison of multiplicative and simplified version

Assume $M_k$ requires no communication

\[ r_0 = b - A_0 x_0 \]

For \( k = 1, \ldots, l \), \( x_k = 0 \)

For \( k = 0, \ldots, l - 1 \)

\[
\begin{align*}
    x_k &:= x_k + M_k^{-1} (r_k) \\
    r_{k+1} &:= r_k + (P_{k+1}^k)^T (r_k - A_k x_k)
\end{align*}
\]

\[ x_l := x_l + M_l^{-1} (r_l) \]
\[ x_l := x_l + M_l^{-T} (r_l - A_l x_l) \]

For \( k = l - 1, \ldots, 0 \)

\[
\begin{align*}
    x_k &:= x_k + P_{k+1}^k x_{k+1} \\
    x_k &:= x_k + M_k^{-T} (r_k - A_k x_k)
\end{align*}
\]
Cost comparison for AMG as preconditioner per cycle

<table>
<thead>
<tr>
<th></th>
<th>multiplicative</th>
<th>additive</th>
<th>mult-additive</th>
<th>simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>$\sum_{k=0}^{l-1} [\text{nnz}(A_k) + \text{nnz}(P_{k+1}^k)]$</td>
<td>$\sum_{k=0}^{l-1} [\text{nnz}(A_k) + \text{nnz}(P_{k+1}^k) + \text{nnz}(A_0)]$</td>
<td>$\sum_{k=0}^{l-1} [\text{nnz}(A_k) + \text{nnz}(\overline{P}_{k+1}^k) + \text{nnz}(A_0)]$</td>
<td>$\sum_{k=0}^{l-1} [\text{nnz}(A_k) + \text{nnz}(\overline{P}_{k+1}^k)]$</td>
</tr>
<tr>
<td>MatVecs</td>
<td>$\sum_{k=0}^{l-1} [2^{MV}(A_k) + 2^{MV}(P_{k+1}^k)]$</td>
<td>$\sum_{k=0}^{l-1} [MV(A_k) + 2^{MV}(P_{k+1}^k)]$</td>
<td>$\sum_{k=0}^{l-1} [MV(A_k) + 2^{MV}(\overline{P}_{k+1}^k)]$</td>
<td>$\sum_{k=0}^{l-1} [2^{MV}(\overline{P}_{k+1}^k)]$</td>
</tr>
</tbody>
</table>

Cost of $\overline{P}_{k+1}^k$ needs to be determined

Note: For # messages replace $\sum_{k=0}^{l-1} MV(A_k)$ by $MV(\Lambda)$!
Consider memory, #flops, #messages, amount of data communicated

3D 7pt Laplace problem, 50x50x50 per process,

best BoomerAMG option (1 level agg. coarsening, HMIS, distance-2 interpolation truncated to max 4 weights per row)

Change factors for mult-additive/multiplicative: (< 1 is good)

<table>
<thead>
<tr>
<th>#procs</th>
<th>Memory</th>
<th>flops</th>
<th>#msg</th>
<th>Data sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2.203</td>
<td>1.013</td>
<td>0.712</td>
<td>0.670</td>
</tr>
<tr>
<td>512</td>
<td>2.206</td>
<td>1.016</td>
<td>1.169</td>
<td>0.688</td>
</tr>
<tr>
<td>4096</td>
<td>2.207</td>
<td>1.012</td>
<td>1.655</td>
<td>0.727</td>
</tr>
</tbody>
</table>

Can achieve improvement using truncated $\overline{P}_{k+1}$
Effect of truncation of $\overline{P}_{k+1}^k$, 4096 cores, 7pt and 27pt stencil

Values per cycle, starting at level 0

- **Improvement factor** vs. **truncation factor**

Values per cycle, starting at level 0

- **Improvement factor** vs. **Max number of elements per row**
Effect of truncation of $\overline{P}_{k+1}^k$, 4096 cores, 7pt and 27pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 7-pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 7-pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 7-pt stencil

![Diagram showing the effect of additive strategies at coarser levels.](image)
Effect of starting additive strategies at coarser levels, 4096 cores, 7-pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 7-pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 27-pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 27-pt stencil
Effect of starting additive strategies at coarser levels, 4096 cores, 27-pt stencil
Additive V-cycle
Weighted Jacobi

Multiplicative V-cycle
Weighted Jacobi

Additive V-cycle
Weighted Jacobi

Mult-Additive V-cycle
with interpolation
truncated to at most
8 elements per row
Weighted Jacobi
Mult-additive AMG using hybrid (ℓ¹-) Gauss-Seidel

- What happens if we use hybrid (ℓ¹)-Gauss-Seidel?

- Now $M_k = \tilde{L}_k = \begin{pmatrix} & \vdots & \\ \end{pmatrix}$

- → dense blocks
  \[ \Lambda_k = \tilde{L}_k^{-T} + \tilde{L}_k^{-1} - \tilde{L}_k^{-T} A_k \tilde{L}_k^{-1} \]

- Still allows combining messages on all levels

- → denser smoothed interpolation
  \[ P_{k+1}^k = (I - \tilde{L}_k^{-T} A_k)P_{k+1}^k \]

- Can use truncation, but what about convergence?

- Can use smaller block sizes, → reduces memory requirements and increases comm/comp overlap

\[
\begin{align*}
    r_0 &= b - A_0 x_0 \\
    \text{For } k &= 1, \ldots, l, \ x_k = 0 \\
    \text{For } k &= 0, \ldots, l - 1 \\
    r_{k+1} &= r_k + (P_{k+1}^k)^T r_k \\
    \text{For } k &= 0, \ldots, l \\
    x_k &:= x_k + \tilde{L}_k^{-1} (r_k) \\
    x_k &:= x_k + \tilde{L}_k^{-T} (r_k - A_k x_k) \\
    \text{For } k &= l - 1, \ldots, 0 \\
    x_k &:= x_k + P_{k+1}^k x_{k+1}
\end{align*}
\]
Some remarks on flops, # messages and amount of data sent

- # messages and amount of data sent is comparable to the ℓ¹-Jacobi case
- Generally uses more flops per cycle than with Jacobi smoother, however saves storage of Λ
Multiplicative V-cycle with L1-Gauss-Seidel

Additive V-cycle with L1-Gauss-Seidel

Mult-additive V-cycle, smoothed interpolation with truncation factor 0.025 with L1-Gauss-Seidel block size 20
Solve times on Hera, LLNL Linux cluster, AMG-PCG, 7pt 3D Laplace, 50x50x50 points / core
Laplace, quadratic elements, ~ 29,000 dofs per core
Solve times on Hera, AMG-PCG, 3D problem with jumps,

Sphere, ~ 56,000 dofs per core

- mult.gs
- maP8.0.gs20
- add.4.gs
- mult.j
- maP8.0.j
- smaP8.0.j

seconds

64 640

no of cores
Comments

- Good results on a machine with fairly slow DDR Infiniband fat tree network thanks to decreased communication, better communication-computation overlap and decreased flops
- We expect the method to perform better than multiplicative AMG on future Exascale machines
- But what about a current machine with fast network and many cores?
Experiments on Vulcan, BG/Q

Solve times - 7pt

- mult-gs
- mult-lj1
- ma-0-lj1
- sma-0-lj1

no of cores

seconds

500 5000 50000
Experiments on Vulcan, BG/Q

Solve times - 7pt

Solve times - 27pt
27pt-stencil, BG/Q, 50x50x50 grid points per core

**no of iterations - 27pt**

- **cycle times - 27pt**

---

[Graph showing the relationship between the number of iterations and the number of cores for various algorithms.]

- **mult-gs**
- **mult-lj1**
- **ma-0-lj1**
- **sma-0-lj1**

---

[Graph showing the relationship between cycle times and the number of cores for various algorithms.]
Conclusions/Future Work

- Mult-additive AMG with Jacobi or small-block hybrid GS smoother and truncated smoothed interpolation can significantly reduce data movement across processes.

- Approach leads to generally smaller cycle times than multiplicative approach with Jacobi or even hybrid GS.

- Even larger time and communication savings can be achieved using the simplified mult-additive approach in spite of increased no. of iterations.

- Most time savings can be achieved starting the approach at the finest level.

- Investigate efficient implementation of setup phase.

- Investigate combinations of various communication reducing approaches (including non-Galerkin approaches).
Thank you!

- This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Partial support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U. S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/ High Energy Physics/ Fusion Energy Sciences/ Nuclear Physics) and by Applied Mathematics Program, DOE ASCR.