Algebraic Multigrid (AMG)

Multigrid Methods and Parallel Computing

Klaus Stüben
Fraunhofer Institute SCAI
Schloss Birlinghoven
53754 St. Augustin, Germany
Klaus.Stueben@scai.fraunhofer.de

2nd International FEFLOW User Conference
Potsdam, September 14-16, 2009
What Are We Talking About?
Large-Scale Applications

- Fluid Dynamics
- Groundwater modeling
- Process- & Device-Simulation
- Casting and Molding
- Circuit Simulation
- Oil reservoir simulation
- Electrochemical Plating
Bottleneck: Linear Equation Solver

Huge but sparse „elliptic“ systems

\[ \sum_{j=1}^{N} a_{ij} u_j = f_i \quad (i = 1, ..., N) \quad \text{or} \quad Au = f \]

Calculation repeated many times over
- to follow the time evolution
- to resolve non-linearities
- for history matching
- for optimization processes

Scalability requires
- Locally “operating” numerical methods
- Exploitation of origin and nature of the discrete linear systems

required: "scalable" solvers,
Work = O(N)

hierarchical solvers:
- multigrid,
- algebraic multigrid
Bottleneck: Linear Equation Solver

Additional aspects affecting numerical performance (besides size)

- extremely unstructured or locally refined grids, highly heterogeneous
**Bottleneck: Linear Equation Solver**

Additional aspects affecting numerical performance (besides size)
- extremely unstructured or locally refined grids, highly heterogeneous
- not optimally shaped elements

**Basin flow model:**
- pentahedral prismatic mesh
- several faulty zones
- vertical distortion of the prisms
Bottleneck: Linear Equation Solver

Additional aspects affecting numerical performance (besides size)

- extremely unstructured or locally refined grids, highly heterogeneous
- not optimally shaped elements
- extreme parameter contrasts, discontinuities and anisotropies

High parameter contrast:
Permeabilities vary by 7 orders of magnitude
Bottleneck: Linear Equation Solver

Additional aspects affecting numerical performance (besides size)

- extremely unstructured or locally refined grids, highly heterogeneous
- not optimally shaped elements
- extreme parameter contrasts, discontinuities and anisotropies
- non-symmetry and indefiniteness
- multiple (varying) degrees of freedom, ...

Requested: solver should

- be fast (scalable)
- be general
- require low memory
- be easy to integrate

Nothing is for free!
Compromise?
(Algebraic) Multigrid
Geometric versus Algebraic Multigrid

Classical one-level approach
Geometric versus Algebraic Multigrid

Geometric Multigrid (GMG)

Restriction

$\Omega^h$ $\Omega^{2h}$ $\Omega^{4h}$ $\Omega^{8h}$

Direct solution

Interpolation

$S^h_{2h}$ $S^h_{4h}$ $S^h_{8h}$

$I^h_{2h}$ $I^{2h}_{4h}$ $I^{4h}_{8h}$

$\Omega$ $\Omega^{2h}$ $\Omega^{4h}$ $\Omega^{8h}$

Fraunhofer Institute
Algorithms and Scientific Computing
Geometric versus Algebraic Multigrid

Geometric Multigrid (GMG)

Basic idea: combine
- relaxation for smoothing
- coarse-grid correction

Numerically scalable

Ideally suited for
- structured meshes
- “smooth” problems

Difficult to integrate into existing software:
- no “plug-in” software
- “technical” limitations

Hardly used in industry!
Geometric versus Algebraic Multigrid

Geometric Multigrid (GMG)

Algebraic Multigrid (AMG)

given linear system: $A_1 u = f$

automatically constructed as part of the AMG algorithm
Two-stage process:

Recursive setup phase $i=1,2,...$:
1. Find subset of variables representing level $i+1$
2. Construct interpolation $I_{i+1}^i$
3. Construct restriction $I_{i}^{i+1}$
4. Compute coarse-level matrix
   \[ A_{i+1} = I_{i+1}^i A_i I_i^{i+1} \] (sparse matrix product (Galerkin))

Straightforward solution phase:
- select "simple" smoother (e.g., relaxation)
- cycling as before
**Geometric versus Algebraic Multigrid**

Relies on same basic idea
- relaxation for smoothing
- coarse-"grid" correction

"Virtually" scalable

Also suited for
- unstructured meshes, 2-3D
- "non-smooth" problems

Easy to integrate into existing software:
- "plug-in" software
- requires only matrices

Very popular in industry!

---

**Algebraic Multigrid (AMG)**

Given linear system:

\[ A_1 u = f \]

Automatically constructed as part of the AMG algorithm
Durlofsky problem:

- flow from left to right
- heterogeneous conductivity (red=1 m/s, blue=10^{-6} m/s)
- steady-state
- basic mesh: 20x20, (refined by halving mesh sizes)
Scalability

Computational work per grid node

- std. PCG
- SAMG

# Elements

- 1600
- 6400
- 25600
- 102400
- 409600
- 1638400
Steady State Test Cases (USGS, MODFLOW)

Convergence Histories (MODFLOW "p1mat")

- Standard solver: CG/ILU
- iter = 662
- CPU = 489 sec
- speedup = 8.2

Convergence Histories (MODFLOW "p2mat")

- Standard solver: CG/ILU
- iter = 4474
- CPU = 1479 sec
- speedup = 63.2 (!)
Distorted Mesh Case (Wasy)

Basin flow model:
- pentahedral prismatic mesh
- a number of faulty zones
- vertical distortion of the prisms
- transient
- parameter contrast: 3 orders
Distorted Mesh Case (Wasy)

Standard one-level method:
- failed to converge below $10^{-8}$
- solution fully instable

SAMG:
- stable solution
- sufficiently accurate
Diesel injection system, ~ 4.2 Mio tetrahedrons
Aspect ratios up to 1:10,000!!
Electrochemical Machining - Extreme Aspect Ratios

Diesel injection system, ~ 4.2 Mio tetrahedrons
Aspect ratios up to 1:10,000!!
Summary of AMG

Steady state applications (elliptic)
- highly efficient
- scalable
- increased robustness
- in particular: discontinuities, anisotropies, ...

Time dependent applications
- applicable in each time step
- increased robustness
- efficiency depends on various aspects such as
  - requested accuracy per time step
  - mesh size
  - "behavior" of classical solvers
  - time discretization, in particular, time step size

How does AMG perform on parallel computers?
Parallel Computing Nowadays
Hardware

Distributed memory
• MPI (process-based)
Shared memory (multicore)

- MPI (process-based)
- OpenMP (thread-based)
- mixed MPI/OpenMP
Hybrid architectures

- mere MPI
- mixed MPI/OpenMP
Hardware

Distributed memory
- MPI (process-based)

Shared memory (multicore)
- MPI (process-based)
- OpenMP (thread-based)
- mixed MPI/OpenMP

Hybrid architectures
- mere MPI
- mixed MPI/OpenMP

Specialized hardware
- GPUs (CUDA, OpenCL)
- Cell
Parallel Computing

General goal

- combine serial scalability with scalability of parallel hardware

However, this is not trivial at all

- there is a conflict
  - really parallel solvers are usually not serially scalable
  - serially scalable solvers are usually not really parallel
- in particular, important AMG modules are not parallel
- parallelization requires changes in the algorithm/numerics
- this requires compromises
  - ensure "stable" convergence (essentially independent of # processors)
  - minimize fill-in / complexity / overhead
  - minimize communication / memory access

Classical (A)MG parallelization: MPI-based
MPI-based Parallelization

The diagram illustrates the process of restriction and interpolation in parallel computing. The grid represents the computational domain, with levels of discretization denoted by $I_h$, $S_h$, $I_{2h}$, and $I_{4h}$. The arrows indicate the direction of data flow from coarser to finer grids (restriction) and from finer to coarser grids (interpolation).

- $I_h$: Direct solution
- $I_{2h}$: Interpolation
- $I_{4h}$: Interpolation

The matrices $A_1$, $A_2$, $A_3$, and $A_4$ represent different levels of discretization, with $I_1$, $I_2$, $I_3$, and $I_4$ indicating the corresponding interpolation steps.
MPI-based Parallelization

Partitioning of 1st grid induces partitioning on all levels!

The user just provides the 1st level set of equations along with a partitioning. The setup phase has to be parallelized!
MPI-based Parallelization

Parallel setup phase:
1. Find subset of variables
2. Construct interpolation
3. Construct restriction
4. Compute coarse-level matrix

Inherently sequential, most critical in parallelization!
Various attempts have been made to construct parallel analogs (eg, LLNL, Griebel). Unfortunately, ...

Fully parallel
However: highly communication-intensive

Solution phase:
1. Smoothing
2. Restriction
3. Interpolation
4. Stepping from fine-to-coarse and back

Generally, good smoothers are serial (eg, GS, ILU)!
Compromise: hybrid smoothers (eg, GS/Jacobi)

Inherently sequential! Towards coarser levels: increasingly bad computation-communication ratio.
Various attempts have been made to improve this (eg, McCormick, Griebel). Unfortunately, ...
Performance of Algebraic Multigrid
Hybrid Architecture (MPI)

Test case:
3D Poisson problem, 27-point FE stencil

Mesh sizes:
- p0: 884,736
- p1: 1,560,896
- p2: 3,176,523
- p3: 6,331,625

Partitioning (Metis): 64 nodes connected by infiniband; 2 dual-core sockets/node (Opteron AMD)

Hardware
Used as: 64-core distributed memory hardware

or as: 64-core hybrid system
Hybrid Architecture (MPI)

Solution phase (1 core/node)

Speedup reduced: Cores within a node have to share the internal bandwidth!
Hybrid Architecture (mixed MPI/OpenMP)

Mixed MPI/OpenMP programming? Performance on 32 cores ....

OpenMP on the nodes seems advantageous:
8:4 is most efficient!

8:4
8 MPI processes with 4 threads each

16:2
16 MPI processes with 2 threads each

32:1
32 MPI processes with 1 thread each
Shared Memory (mixed MPI/OpenMP)

Shared memory (multicore)
- AMD Opteron Processor 8384, 2.7 GHz

Programming
- MPI or (straightforward!) OpenMP, mixed MPI/OpenMP

8 Quad-Core sockets
- 4 CPUs
- 8 x 64 KB L1 Cache (Instr./Data)
- 4 x 512 KB L2 Cache
- 1 x 6144 KB L3 Cache (shared)
- 1 x 16144 MB (shared)

Comparison:
(MPI processes : Threads per proc) = (1:16), (2:8), (4:4), (8:2), (16:1)

Remark: We use only 16 cores in order to investigate thread-/memory binding
Shared Memory (mixed MPI/OpenMP)

Process- / thread binding
  coh: coherent;
  scat: scattered
  sys: system-provided

Run time

Run time

sys-binding

Fraunhofer Institute Algorithms and Scientific Computing
Shared Memory (mixed MPI/OpenMP)

Process- / thread binding
coh: coherent;
scat: scattered
sys: system-provided

Run time

coh-binding

Fraunhofer Institute Algorithms and Scientific Computing
Shared Memory (mixed MPI/OpenMP)

Influence of different thread-bindings: fairly unpredictable, practically does not pay

2D, FD Poisson

3D, FE Poisson

3D, FE Elasticity
Shared Memory (OpenMP)

Conclusions:
• Pure MPI seems a reasonable choice
• Truely hybrid is faster only rarely
• Manual thread binding does not really pay
• Pure OpenMP is always slowest

Reasons:
• Sparse matrix operations
  - matrix-matrix, matrix-vector
• Lots of integer computations
  - indirect addressing, CSR data storage, ...
• Efficient cache utilization very difficult
  - for multicore much more difficult than unicore
  - not sufficient to just consider matrix-vector
  - efficient tools are required

Problem with pure OpenMP:
• Setup phase scales nicely
• Scalability of solution phase is limited
Shared Memory (OpenMP)

Conclusions:
• Pure MPI seems a reasonable choice
• Truely hybrid is faster only rarely
• Manual thread binding does not really pay
• Pure OpenMP is always slowest

Reasons:
• Sparse matrix operations
  - matrix-matrix, matrix-vector
• Lots of integer computations
  - indirect addressing, CSR data storage, ...
• Efficient cache utilization very difficult
  - for multicore much more difficult than unicore
  - not sufficient to just consider matrix-vector
  - efficient tools are required

Remedies (unicore):
• Improve cache utilization by, eg,
  - register blocking
  - pre-fetching
  - cache blocking
• Change data access,
  - improve re-usage of matrix entries, eg,
  - "wave-like" relaxation, restriction, ....
  - potential conflict with modular programming

Remedies (multicore):
• Data blocking, re-ordering
  - minimize memory bank conflicts
• Explicit thread- and/or memory bindings
  - not (yet) really reliable
  - not enough information available
  - too hardware-dependent
Conclusions

Clearly, results and conclusions strongly depend on

- hardware, network, memory access, cache hierarchy, ....
- application, in particular, discretization, grid size, sparsity, ....

Distributed memory architecture

- complex programming via MPI
- easy to achieve high scalability (if problem size per node is sufficiently large)

Hybrid architecture (nodes consisting of multicores)

- mere MPI suffers from a reduced bandwidth inside the nodes
- mixed MPI/OpenMP programming seems best (if adjusted to the architecture)

Shared memory architecture (one or more multicore sockets)

- mere MPI is generally superior to pure (straightforward!) OpenMP
- using reasonable threat-binding, mixed programming is occasionally better
- however, without substantial effort, OpenMP performance is currently limited
  - eg, in the solution phase (dominated by matrix-vector operations)
  - demanding developments still required (tools as well as algorithms)
  - dynamic decisions are actually required at run time!
Conclusions / Outlook

One should not forget numerics (solvers)!

- it is important to consider numerically "optimal" solvers (such as AMG)
- it does not make sense to use sub-optimal solvers just because they scale better

Unfortunately,

- depending on the situation, AMG may not be suitable
  - not applicable, problem too small, insufficient convergence
- there exists no solver which works always and is always efficient!
- in a time-dependent simulation, a good solver may even depend on time
- solvers should be adjusted at run time

Challenge: Dynamic and automatic decisions at run time

- automatic switching between solvers and parameters ("intelligent solver")
- automatic adjustment of parallelization strategy
Thank You!

klaus.stueben@scai.fraunhofer.de

http://www.scai.fraunhofer.de/samg

• Reports
• User Manuals
• Benchmarks