SAND2013-5548C

Exceptional service in the national interest

Sandia National Laboratories

Scaling Challenges in Multigrid Solvers

Jonathan Hu

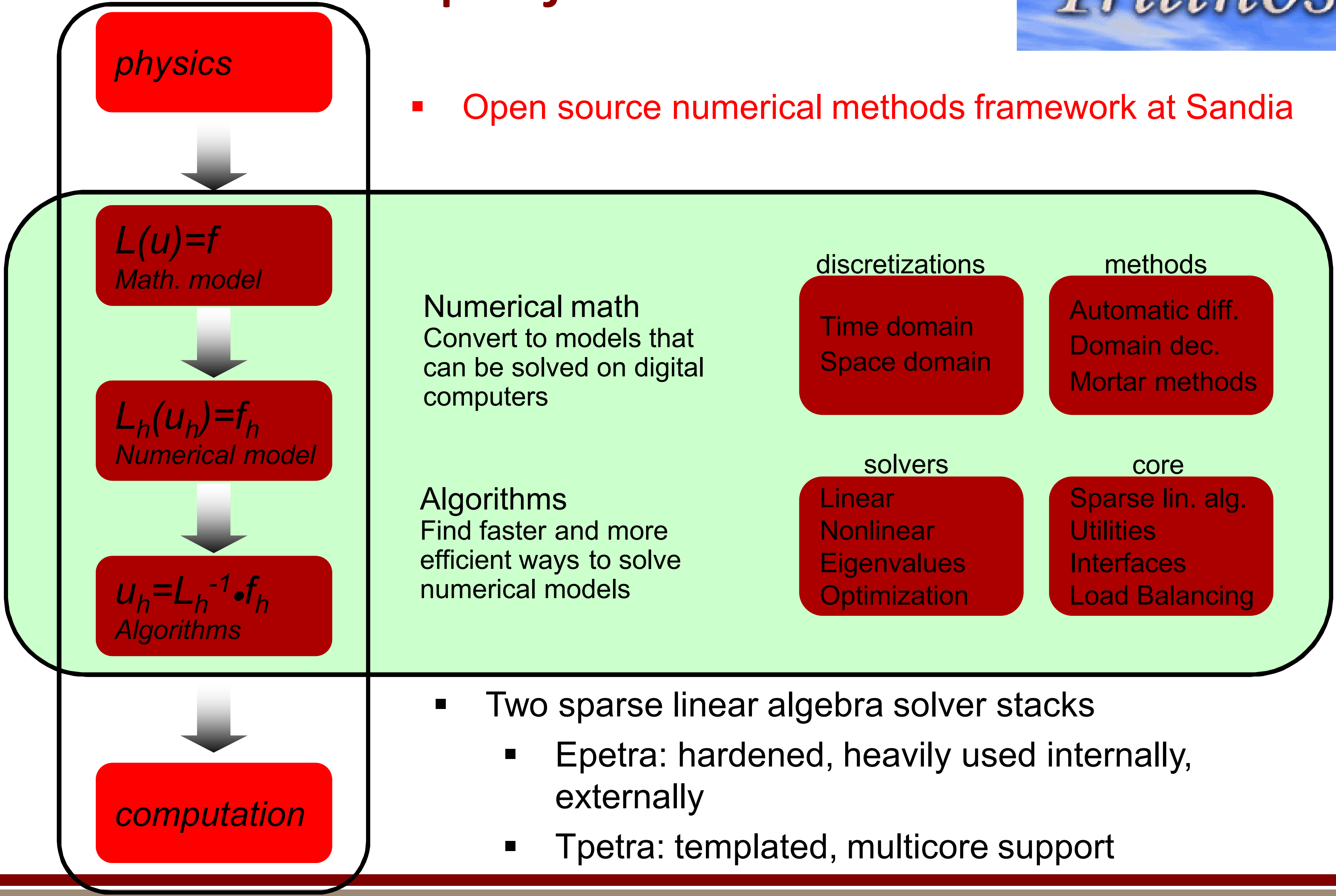
SIAM Annual Meeting, June 8-12, 2013

Outline

- Background / Motivation
  - The Trilinos project
  - Motivating application
  - MueLu, a multigrid framework
- Challenges
  - Multigrid setup costs
  - Design flexibility versus performance
- Numerical Results
- Conclusions

The Trilinos project

- Open source numerical methods framework at Sandia



Numerical math
Convert to models that can be solved on digital computers

Algorithms
Find faster and more efficient ways to solve numerical models

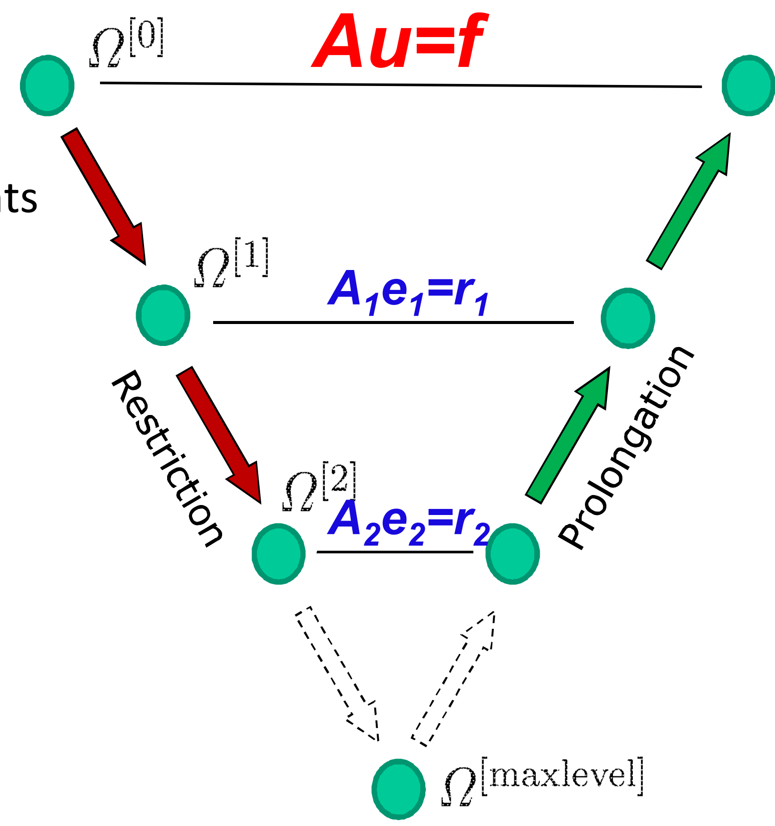
- Two sparse linear algebra solver stacks
  - Epetra: hardened, heavily used internally, externally
  - Tpetra: templated, multicore support

Trilinos and Sierra integration

- Sierra is well-known Sandia engineering simulation framework
  - Regularly uses the Trilinos Epetra stack
  - Moving to Tpetra is opportunity to exploit multicore architectures
- Trilinos templated Tpetra stack includes:
  - Sparse linear algebra, Krylov methods, multigrid methods, iterative solvers, sparse direct methods, load balancing methods
- Sierra/Trilinos integration project:
  - Deploy Trilinos templated Tpetra linear algebra/solver stack in Sierra
  - Demonstrate that representative simulations with Tpetra stack are competitive with hardened Epetra solvers
- Required significant effort throughout Tpetra stack
- This talk focuses on the algebraic multigrid part of the integration

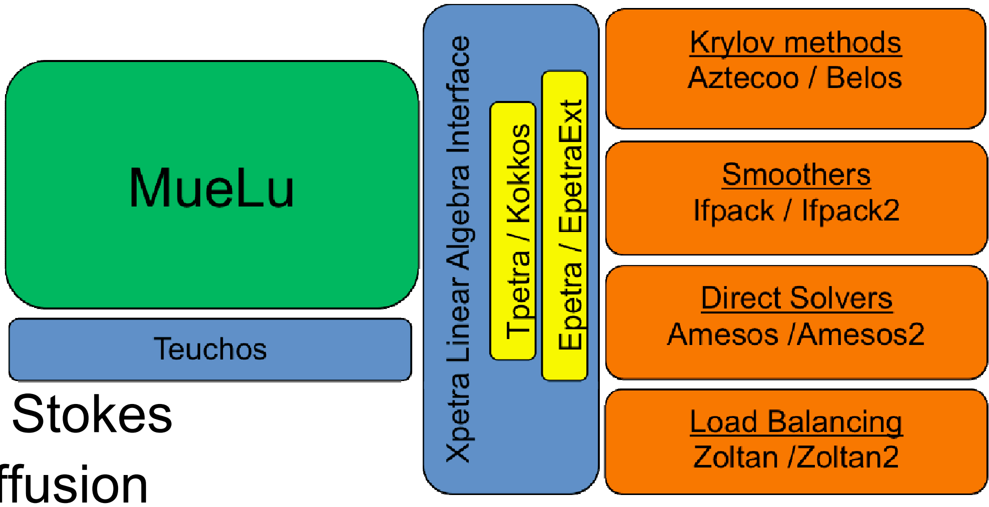
Algebraic multigrid (AMG)

- Scalable method for preconditioning elliptic systems
- Two main components
  - Smoothers
    - Approximate solves on each level
    - “Cheaply” reduces particular error components
    - On coarsest level, smoother = A_i^{-1} (usually)
  - Grid Transfers
    - Moves data between levels
    - Must represent error components that smoothers can’t reduce
- Algebraic Multigrid (AMG)
  - AMG generates grid transfers
  - AMG generates coarse grid A_i’s



MueLu multigrid framework

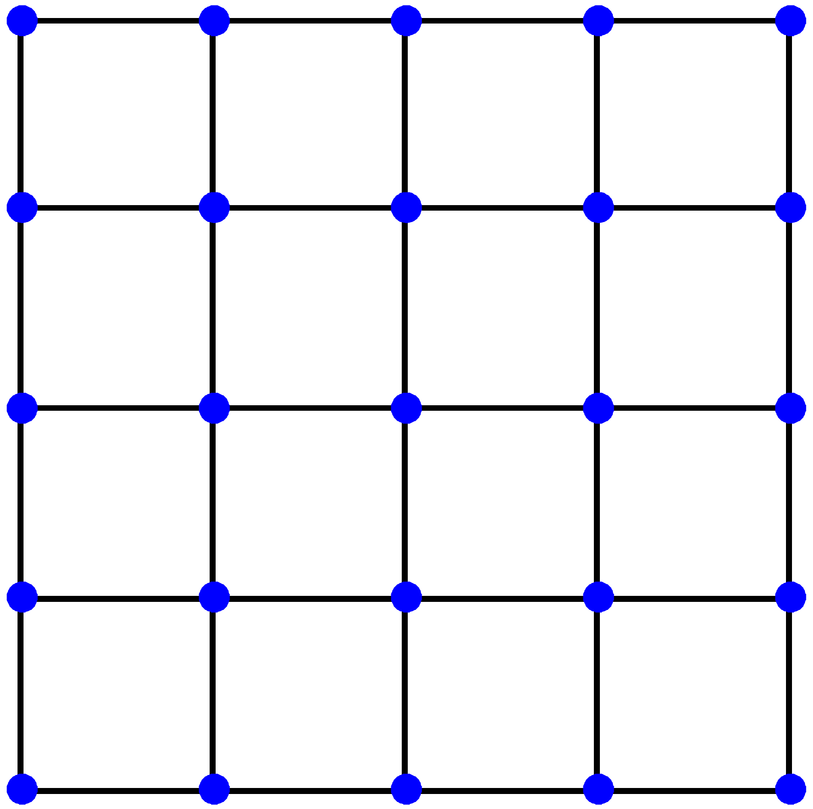
- Part of Tpetra templated solver stack
- Supports variety of scalar and ordinal types
- Support for modern many-core architectures
  - Via Tpetra/Kokkos layers
- Extensible
  - Smoothed aggregation AMG
  - Energy minimization (AP talk)
  - Helmholtz solvers (PT talk)
  - Geometric multigrid for Navier Stokes
  - PG-SA AMG for convection diffusion
- Supports preconditioner reuse to reduce setup expense
  - All or parts of preconditioner (AP talk)



Challenges in parallel AMG

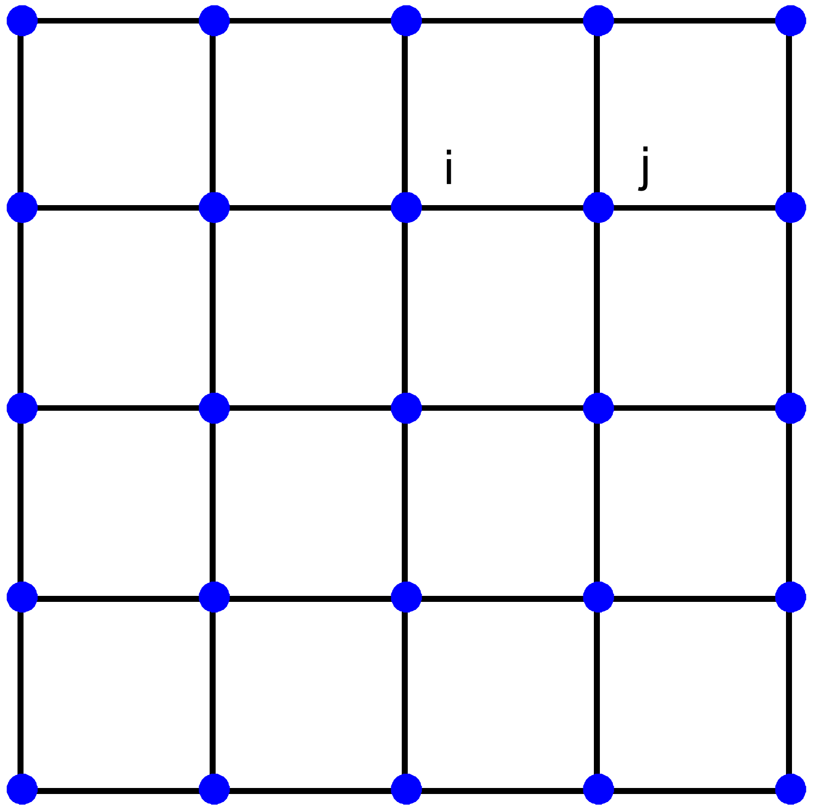
- Nontrivial setup costs
  - Aggregation (matrix graph coarsening)
  - Galerkin triple matrix product to form coarse grid operators
  - Matrix transpose
- Load balance and data movement
  - Multigrid hierarchy consists of many sparse matrices of different sizes and densities
  - Setup and apply sensitive to data distribution and movement
- Tension between design flexibility and performance

AMG coarsening by aggregation



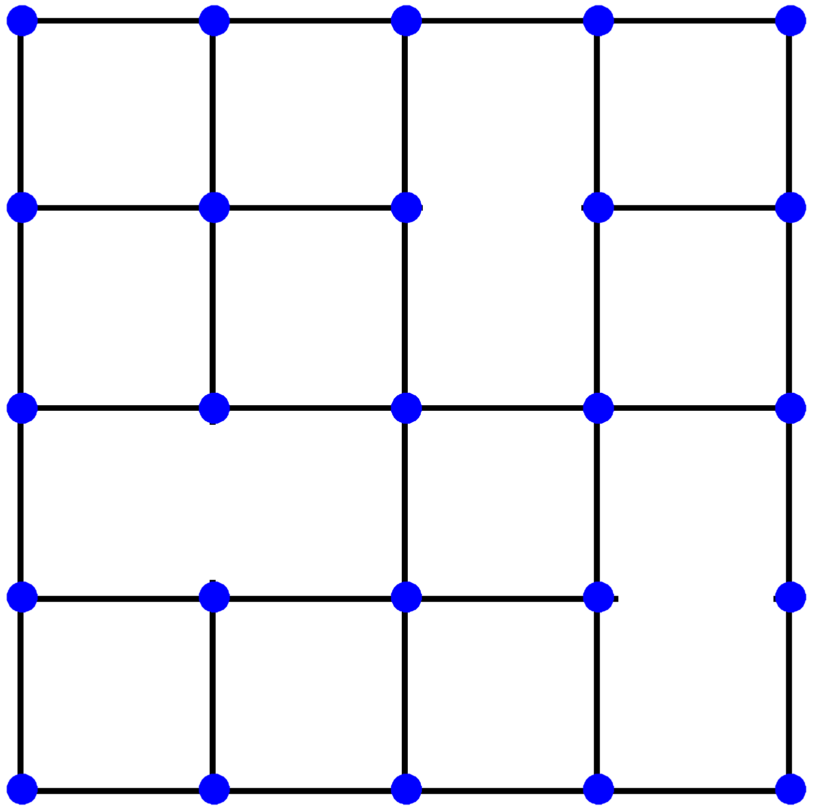
- Coarsen matrix connectivity graph
  - Nodes = diagonal entries
  - Edges = off-diagonal entries
- A coarse DOF is formed by set of fine DOFs called *aggregate*

AMG coarsening by aggregation



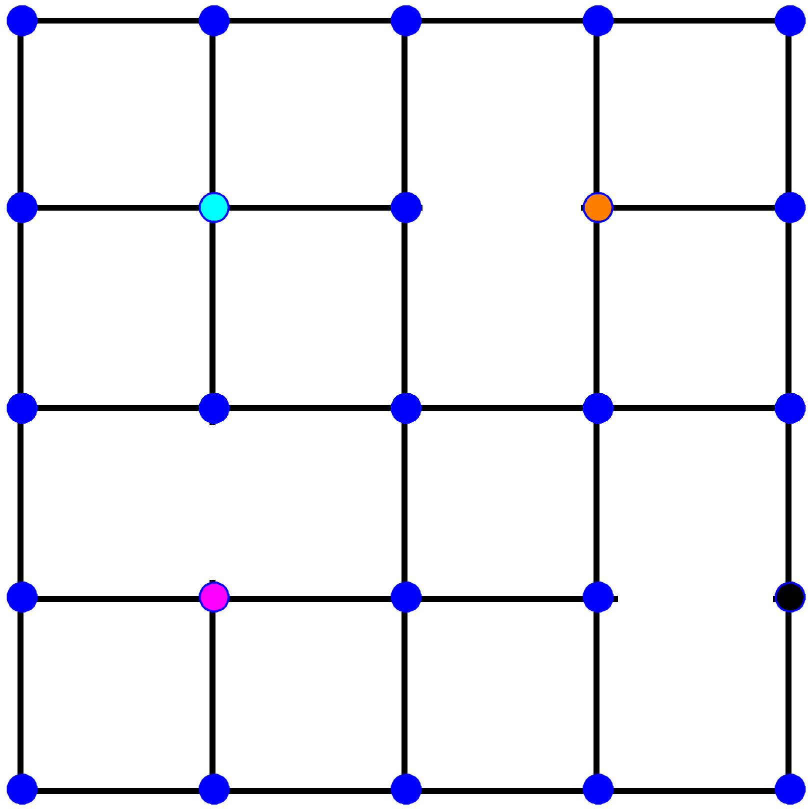
- Two DOFs i and j are connected if a_{ij} is nonzero

AMG coarsening by aggregation



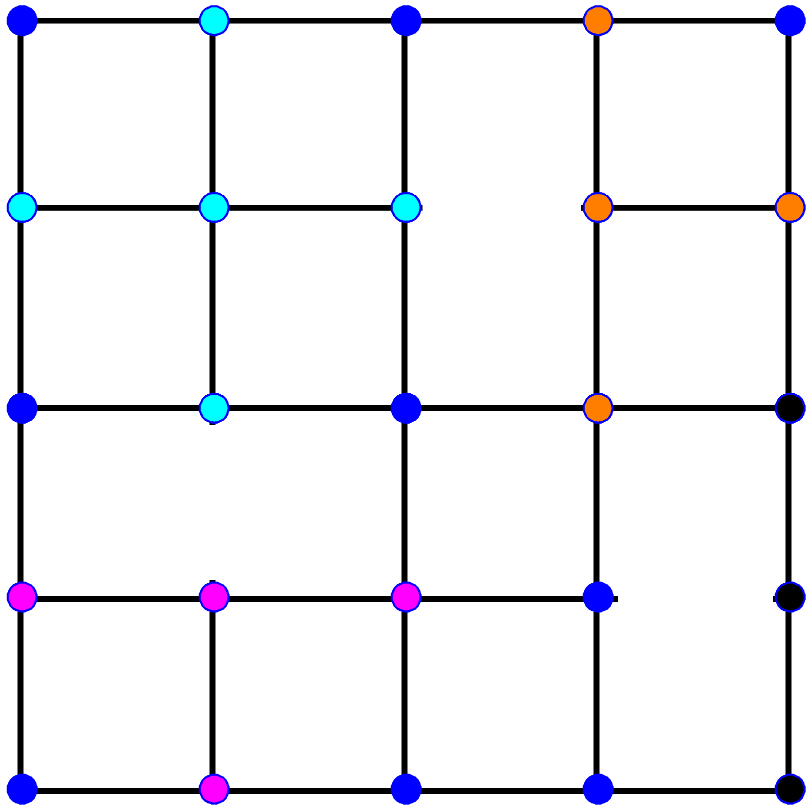
- Two DOFs i and j are connected if a_{ij} is nonzero
- Graph is pruned by dropping too-small connections
- Based on strength of connection (SoC) measure, e.g.,
  - $|a_{ij}|^2 > \varepsilon \, |(a_{ii})(a_{jj})|$

AMG coarsening by aggregation



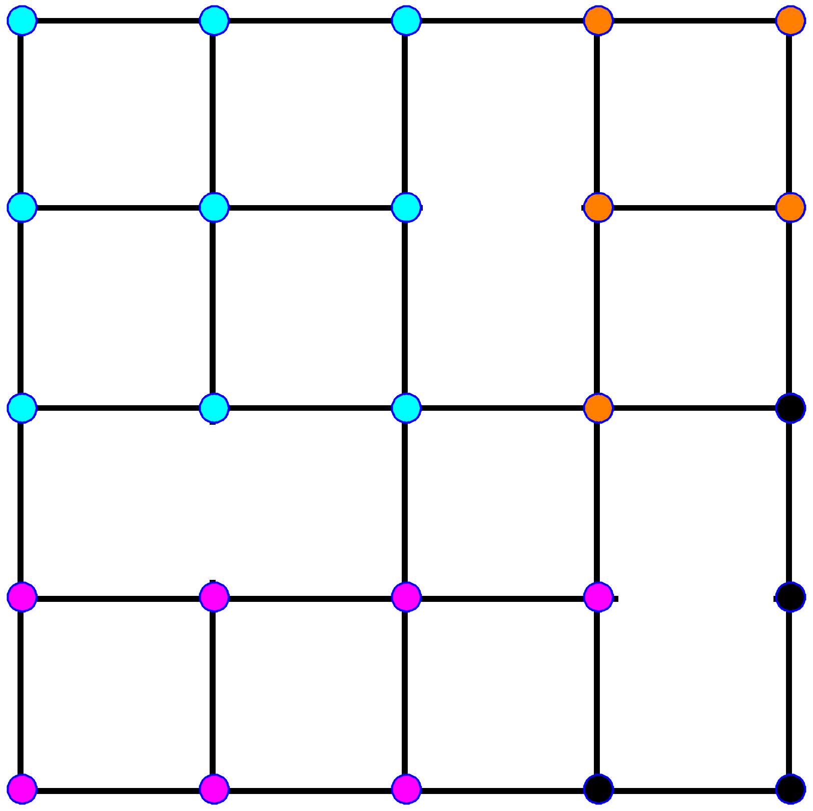
- Choose distance-3 points as root nodes to seed aggregates

AMG coarsening by aggregation



- Add all immediate neighbors to root point’s aggregate

AMG coarsening by aggregation



- Cleanup phase
  - Add remaining nodes to aggregates, or
  - Create new aggregates

Aggregation: parallel performance considerations

- Initially considered scheme that allows aggregates to include DOFs from multiple processors
  - Pro: robust, avoids too-small aggregates
  - Con: Requires global synchronizations that do not scale
- Current approach: aggregate only on core (local, no communication)

Aggregation: parallel performance considerations

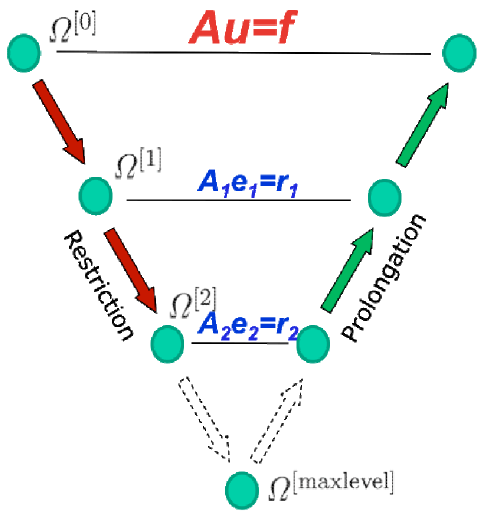
- For scalar PDE and no dropping, aggregation can operate on graph of existing matrix
- Existing graph is insufficient if
  - SoC leads to dropped connections
  - System of PDEs and want to keep DOFs @ mesh node together
    - Requires coalescing multiple DOFs to single graph node
- Build a 2nd parallel graph?
  - Just a few lines of code
  - But almost as expensive as building matrix itself

Aggregation: parallel performance considerations

- For scalar PDE and no dropping, aggregation can operate on graph of existing matrix
- Existing graph is insufficient if
  - SoC leads to dropped connections
  - System of PDEs and want to keep DOFs @ mesh node together
    - Requires coalescing multiple DOFs to single graph node
- Build a 2nd parallel graph?
  - Just a few lines of code
  - But almost as expensive as building matrix itself
- Solution
  - Build local version of graph
    - Dropping requires no communication
  - If need global or off-core information, use original graph's parallel communicator

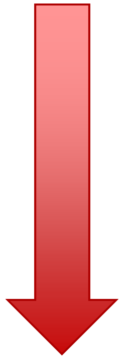
Work load and communication

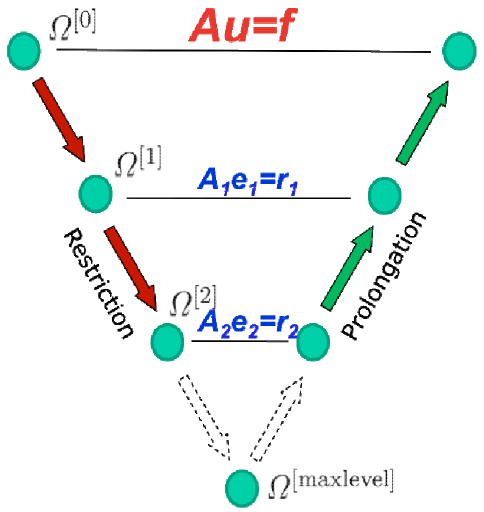
- AMG hierarchy is sequence of sparse operators
  - Decreasing rows, #nonzeros
  - Decreasing workloads per core during setup and apply
  - Increasing communication to computation ratio in parallel



Work load and communication

- AMG hierarchy is sequence of sparse operators
  - Decreasing rows, #nonzeros
  - Decreasing workloads per core during setup and apply
  - Increasing communication to computation ratio in parallel

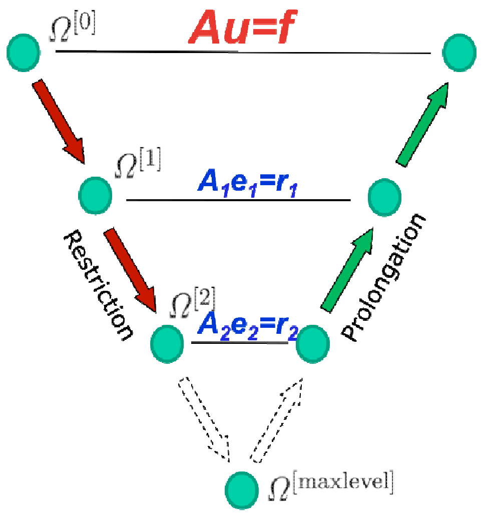




- Last two issues imply communication can dominate on coarser levels.
  - Leads to high preconditioner apply cost

Work load and communication

- Three options for dealing with decreasing workload/core, increasing comm. to comp. ratio
  1. Stop coarsening prematurely (must solve large parallel sparse system)
  2. Replicate coarse grid problem across cores (no idle cores)
  3. Move coarse grid problem to core subset (leaving some cores idle)

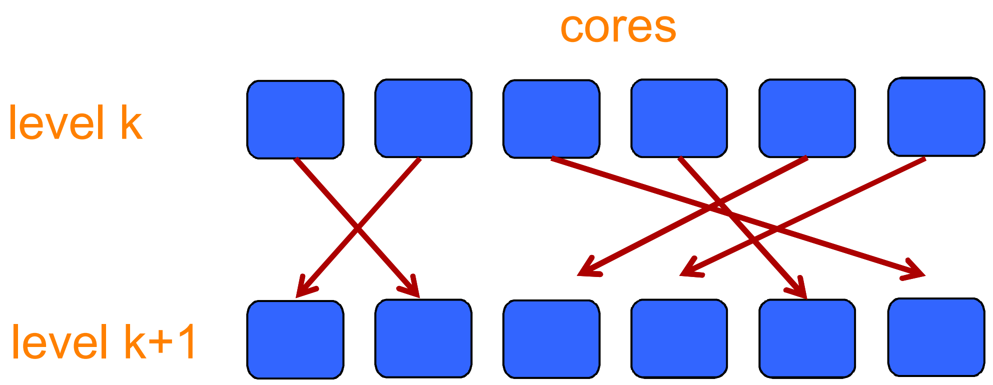


Work load and communication

- Three options for dealing with decreasing workload/core, increasing comm. to comp. ratio
  1. Stop coarsening prematurely (must solve large parallel sparse system)
  2. Replicate coarse grid problem across cores (no idle cores)
  3. Move coarse grid problem to core subset (leaving some cores idle)
- Our approach:
  - Coarse grid matrices A_k moved to subset of cores if
    - #nonzeros/core becomes sufficiently imbalanced or
    - Work/core drops below given threshold
  - Zoltan2 calculates new partition numbering, either by recursive coordinate bisection (RCB) or multijagged (MJ) partitioning
  - MueLu does data migration

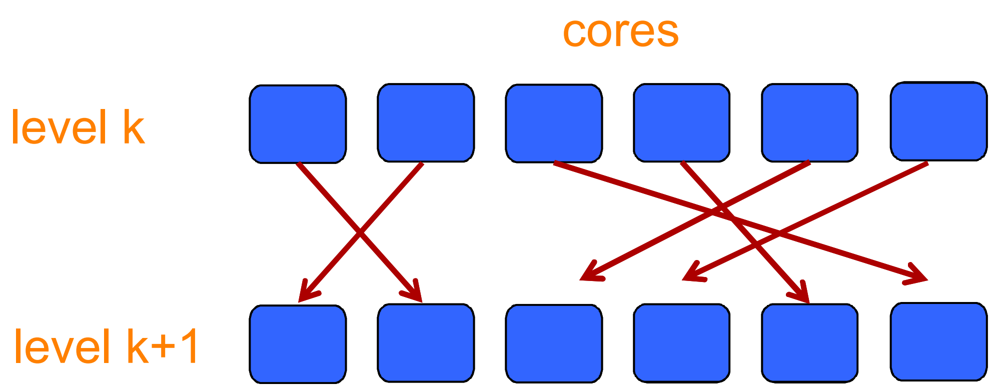
Minimizing data movement between cores

- Good load balance alone is insufficient, however
  - RCB and MJ partitioning algorithms are not diffusive, meaning data distribution may be very different on two consecutive AMG levels



Minimizing data movement between cores

- Good load balance alone is insufficient, however
  - RCB and MJ partitioning algorithms are not diffusive, meaning data distribution may be very different on two consecutive AMG levels



- Poor partition placement => unnecessary data movement in
  - Rebalancing data (setup)
  - Doing matrix matrix multiply (setup)
  - Applying restriction (apply)
  - Applying prolongation (apply)

Minimizing data movement between cores

- Our approach
  - Zoltan2 calculates partition # that each DOF belongs to
  - MueLu assigns partition to core that has most DOFs in that partition
- Phase 1
  - Each core k creates vector of nonzero weights $\{w_k(i)\}$, where $w_k(i)$ = #DOFs on core k in partition i.
  - Core k assigned partition i if $w_k(i) \geq w_j(i)$ over all cores j.
  - Ties broken randomly.
  - Repeat, with cores owning partitions dropped out.
- Phase 2
  - Remaining partitions assigned to lowest free PIDs.
- In practice
  - Often only a few rounds of Phase 1 necessary
  - #rounds in Phase 1 is capped

Optimization of sparse linear algebra kernels

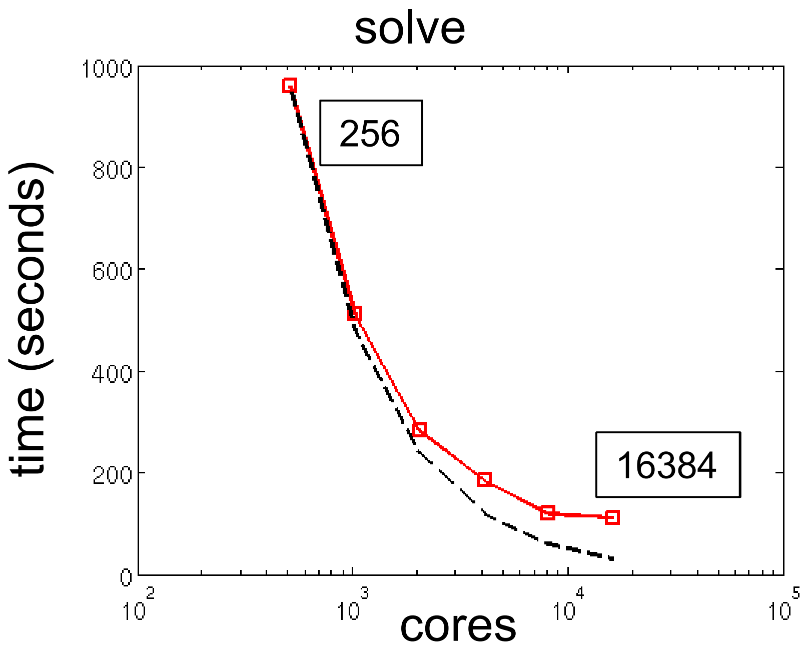
- AMG setup requires
  - Matrix matrix multiply
  - Transpose
- Matrix matrix multiply
  - Highly tuned parallel variant of Gustavson algorithm
  - Forgoes transpose variants (e.g., AB^T) for speed
  - Short-circuits standard matrix fillComplete
    - Global to local indexing, column maps, intercore comm. pattern
- Transpose
  - For symmetric problems, $R = P^T$
  - Transpose local matrix first
  - Short circuits standard matrix fillComplete
- Haven't mentioned many other sparse linear algebra performance improvements

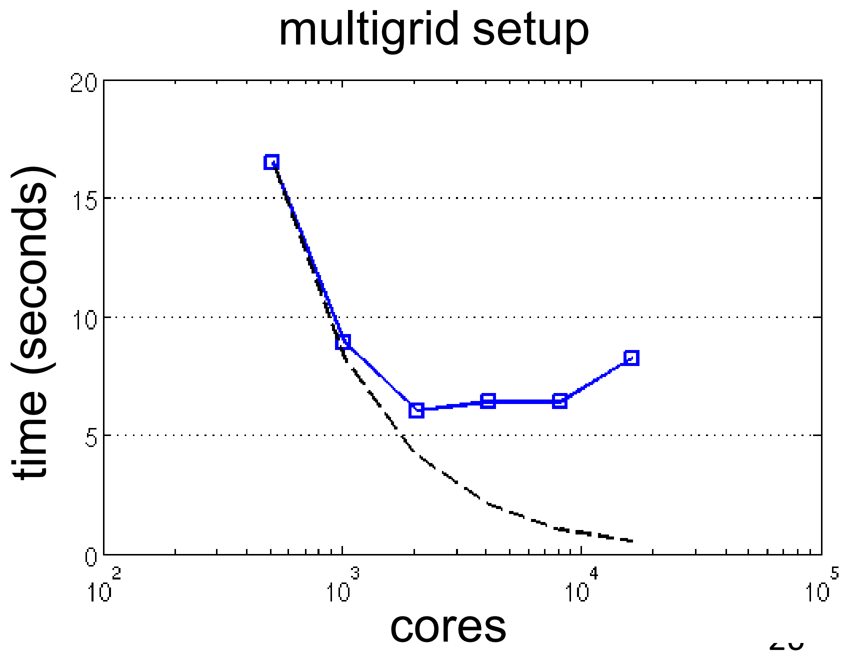
Design versus performance

- Aggregation
  - Using native Tpetra graph
    - Cleaner design, faster development
    - Setup too expensive for our needs
  - SoC initially designed to be separate methods
    - User can provide own custom SoC measures
    - Issue – must be called for every DOF, which is slow
- Matrix matrix multiply
  - Not fully general (no transpose mode)
  - Supports just AB use case that we care about

Numerical results

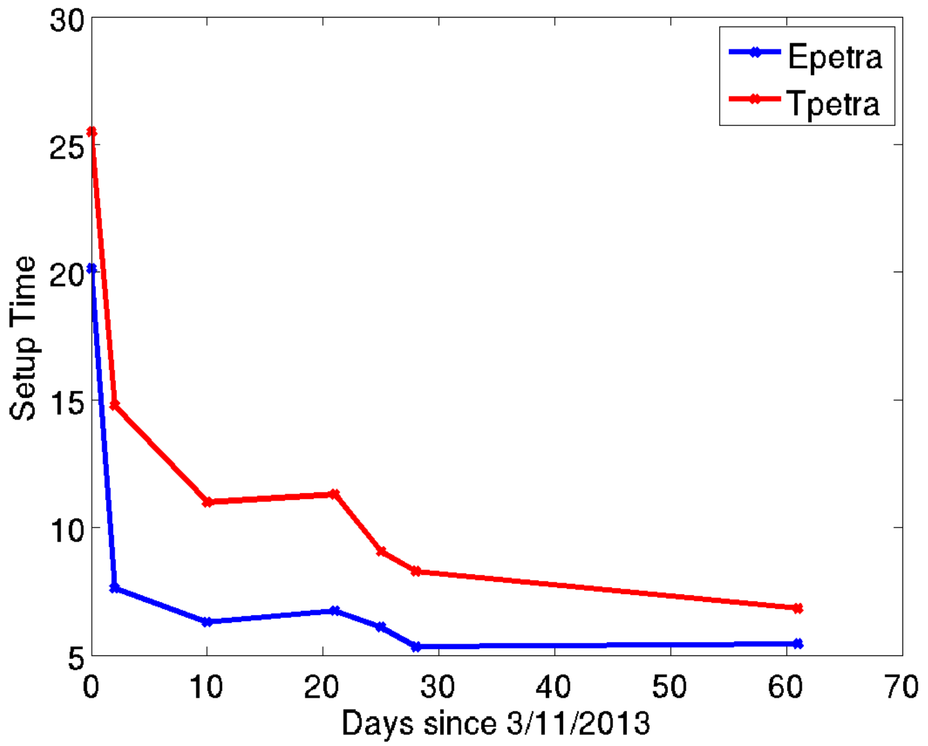
- Strong scaling on Cray XE6
- 3D pressure solve within Sierra simulation
- Semi-structured mesh, 140m DOF matrix
- GMRES preconditioned by aggregation-based AMG
- Stopping criterion $|r_{final}| / |r_0| < 1e\text{-}4$





MueLu Setup Performance Over Time

- Synthetic constant coefficient 3D Poisson system on structured mesh.
- Weak scaling, ~100K DOFs per core.
- NERSC Cray XE6
- Epetra (blue) version has fully optimized MMM and lightweight matrix fillCompletes



Run on 5184 cores of hopper.nersc.gov.
Plot courtesy of C. Siefert.

Ongoing and future work

- Translate improvements in Epetra matrix matrix multiply to Tpetra library
- Deploy expert matrix fillComplete methods in solvers
- Further coarsening optimizations
- Reuse of information between setup phases when possible
- Refine application interfaces

Conclusions

- Important to reduce parallel communication through AMG stack
- Heavy dependence on underlying sparse matrix kernels
- Tradeoffs between design and performance
  - Graph representation in aggregation
  - SoC criterion
  - MMM
- Improvements required collaborative effort among Trilinos and Sierra developers
- For more information on Trilinos project: trilinos.sandia.gov

Acknowledgements

- ASC program support
- Sandia staff involved in Sierra/Trilinos Tpetra solver stack integration, including:
  - Matt Bettencourt, Erik Boman, Karen Devine, Stefan Domino, Travis Fischer, Mark Hoemmen, Paul Lin, Eric Phipps, Andrey Prokopenko, Siva Rajamanickam, Chris Siefert