Algebraic Multigrid on a Dragonfly Network: First Experiences on a Cray XC30

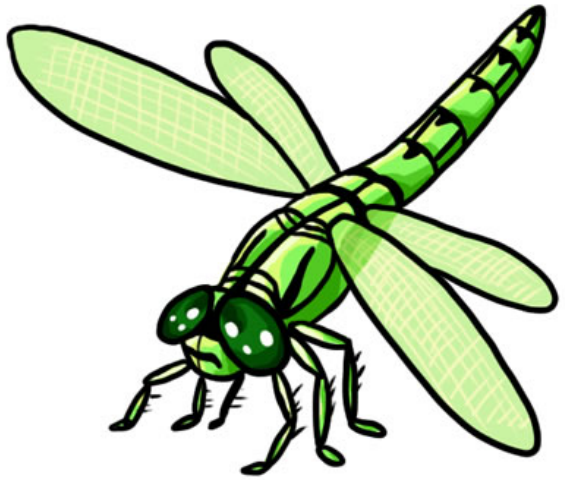
Hormozd Gahvari[1], William Gropp[2], Kirk E. Jordan[3], Martin Schulz[1], Ulrike Meier Yang[1]

[1]Lawrence Livermore National Laboratory
[2]University of Illinois at Urbana-Champaign
[3]IBM TJ Watson Research Center

November 16, 2014

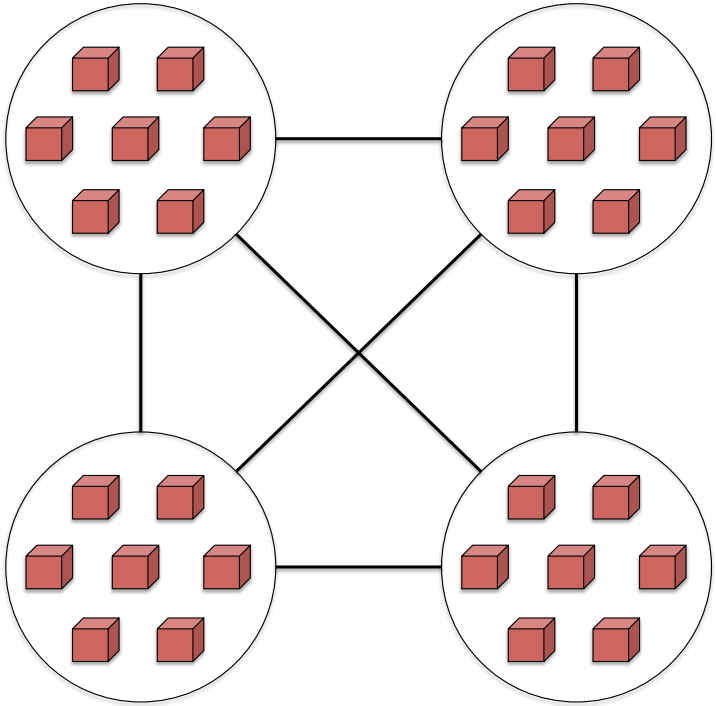
Introduction

- So we have a new interconnect topology making the rounds
- First major appearance in Cray XC30
- Questions to answer:
  - How do applications perform on this topology?
  - Advantages and disadvantages?
  - Prospects for larger future machines?
- We examine the performance of one application on an XC30 to see what insights we can gain



Dragonfly Interconnect

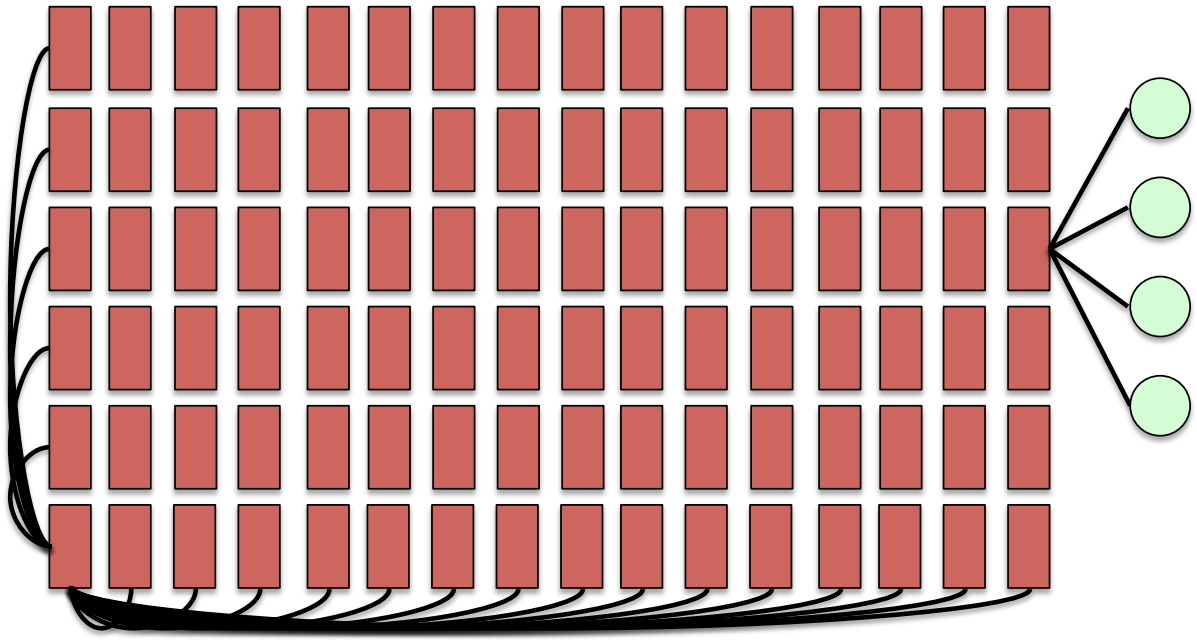
- Groups of routers fully connected by optical links:



- Routers in each group connected through their own topology

Cray XC30 Dragonfly

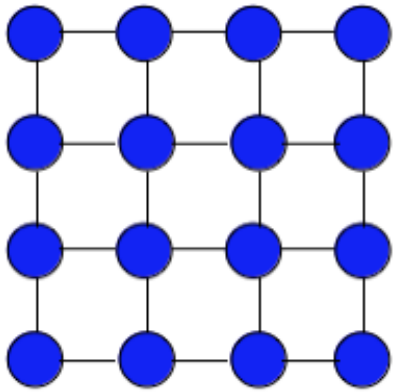
- Routers in each group arranged in a 2D grid with all-to-all links in each dimension:



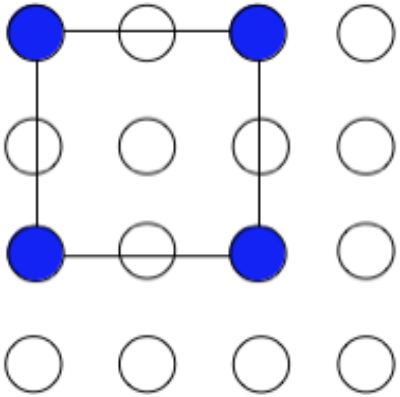
- Four nodes connected to each router

Algebraic Multigrid

Apply multigrid concept:

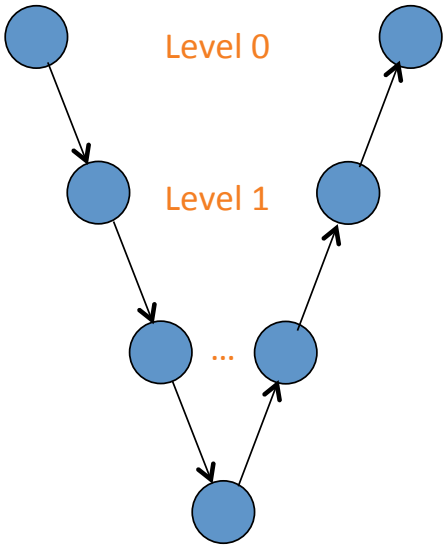


Solve original "fine" problem

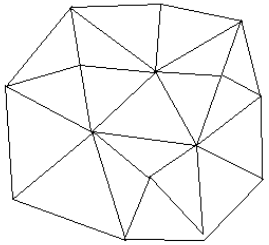


With information from smaller "coarse" problems

And cycle:



To unstructured grid problems:



Requires two phases:

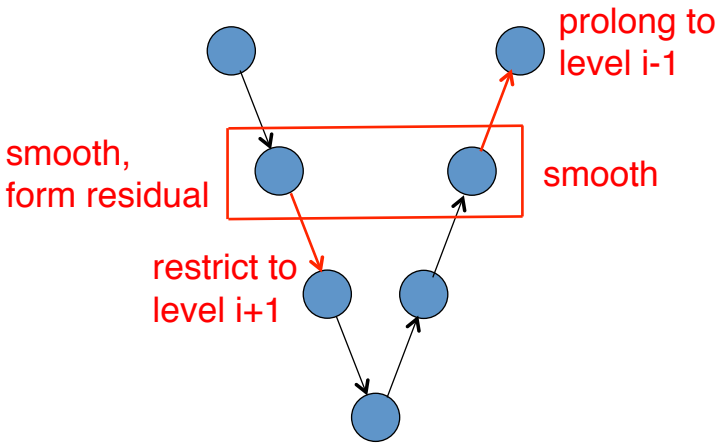
1. Setup hierarchy of grids
2. Solve problem

Performance Modeling

- In past work, developed a performance model for the AMG solve cycle
- Used it to analyze performance on several machines and guide improvements on a number of them
  - Fat-tree Linux clusters
  - Blue Gene/P, Blue Gene/Q
  - Cray XT5, XK6, XK7
- Extend work to XC30 to analyze dragonfly

Performance Modeling

- Approach: work level-by-level, with α-β model (T_{send}=α+nβ for message of length n) as baseline



- Fundamental operations at each level shown in red
- Treat each operation as MatVec with appropriate operator

- Machine parameters for network and computation rate measured using benchmarks
- Communication, computation counts are available from solver data structures

Performance Modeling

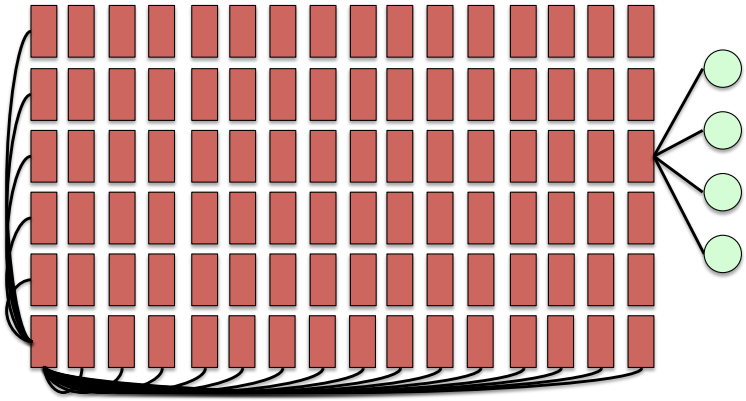
- To the baseline models, we add penalties to take architecture into account:
  - Distance of communication: introduce time per hop γ
    - Measured from worst-case, best-case latencies and global network diameter
    - Distance of diam(P) charged to each message
  - Lower effective bandwidth: multiply β by $\frac{\text{Hardware Bandwidth}}{\text{MPI Bandwidth}}$ or $\frac{\text{Hardware Bandwidth}}{\text{MPI Bandwidth}} + \frac{\text{nmsgs}}{\text{nlinks}}$, depending on machine
  - Multicore penalties:
    - c = number of cores per node
    - P_i = number of "active" processes on level i
    - Multicore latency penalty: multiply α by $\left\lceil \frac{cP_i}{P} \right\rceil$
    - Multicore distance penalty: multiply γ by $\left\lceil \frac{cP_i}{P} \right\rceil$
  - Hybrid MPI/OpenMP: if using j threads, multiply time per flop by (Mem BW for 1 thread)/(Mem BW for j threads)

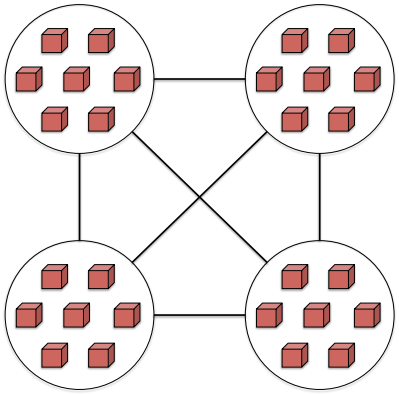
Extending to Dragonfly

- # of links and distance for penalties are computed based on the interconnect
- Torus networks: compute from geometry
- Fat-trees: distance = going up and down the tree, # links = midpoint between best and worst cases
- What to do for dragonfly?

Extending to Dragonfly

- Message distance: use maximum shortest path between nodes
- On XC30:

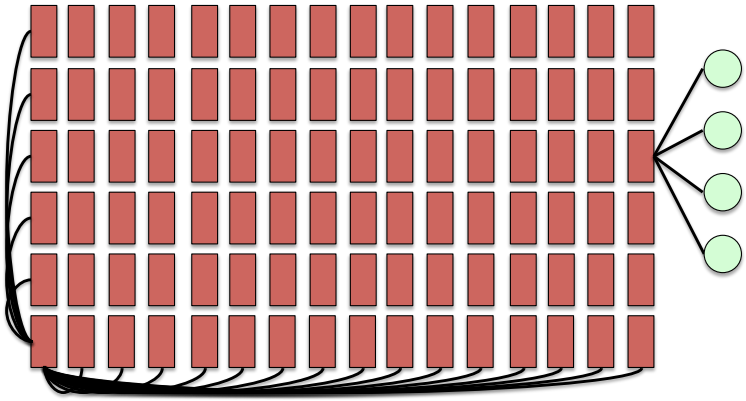


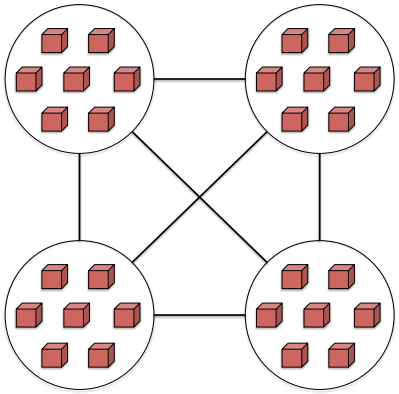


- 1 link to routers
- 2 router links to access optical network (not every router has access)
- 1 optical link
- 2 router links to router attached to target node
- 1 link to target node

Extending to Dragonfly

- Number of links: like fat-tree, midpoint between minimum and maximum
- On XC30:



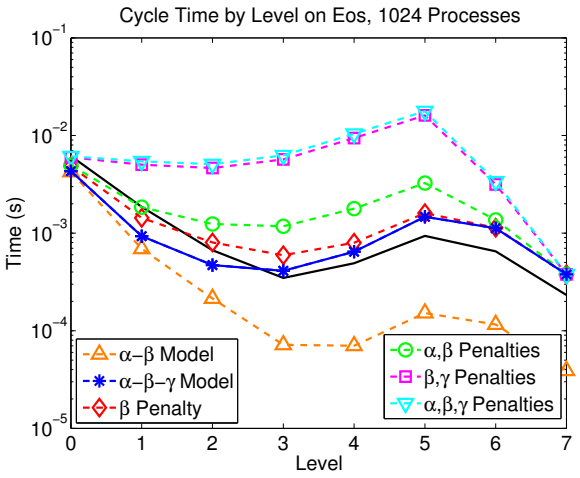


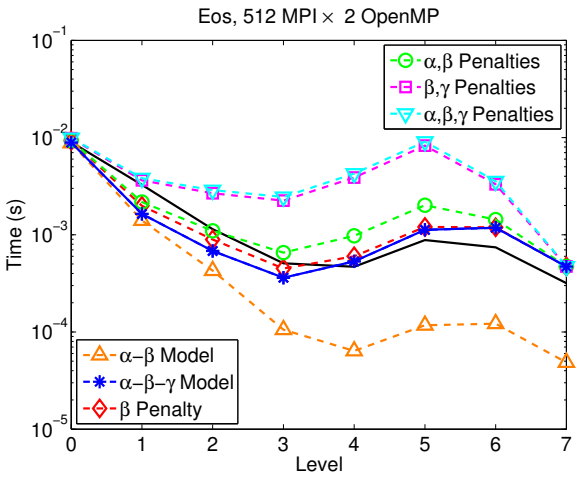
- Optical links have 4x bandwidth, so treat as 4 links
- Minimum: all nodes in a group are taken before moving to the next one
- Maximum: each node is in a new group, until all groups are in use

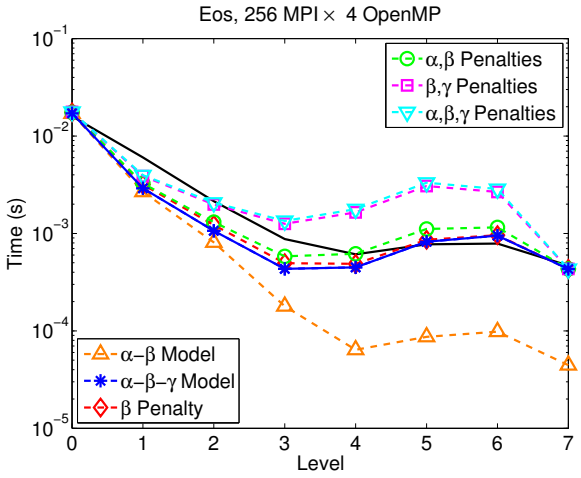
Modeling Results

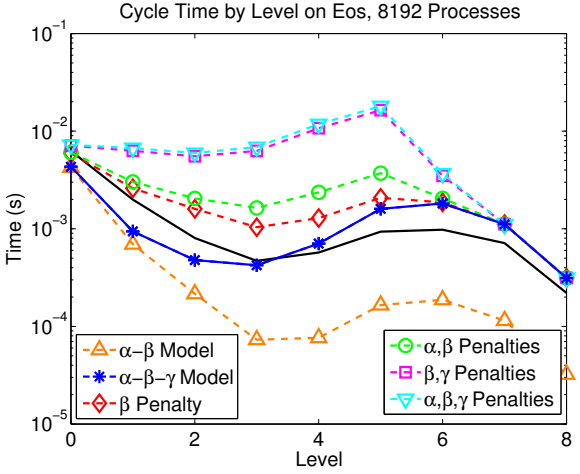
- Tested model on Eos, an XC30 at ORNL
  - 744 nodes with two 8-core processors per node
  - Ran experiments on 1024 and 8192 cores
- Test problem: 3D Laplace with 50 x 50 x 25 points per core
- Compared actual performance (black line in plots) with different model scenarios (colored lines); best fit highlighted

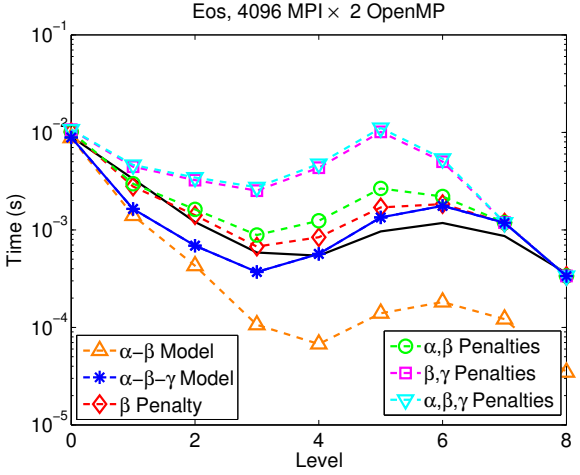
Modeling Results

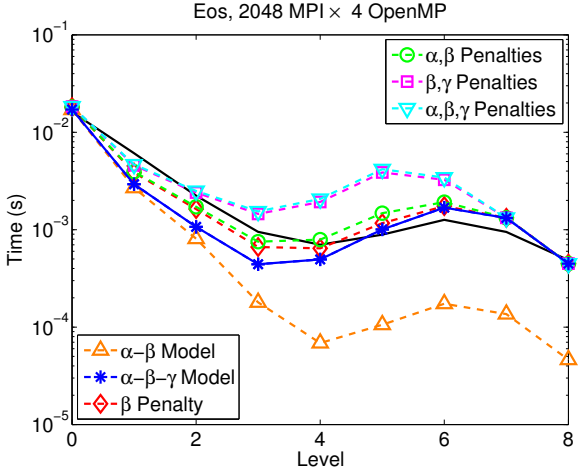




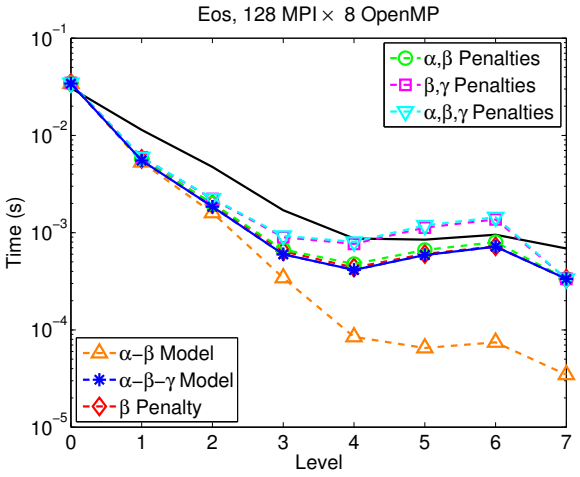


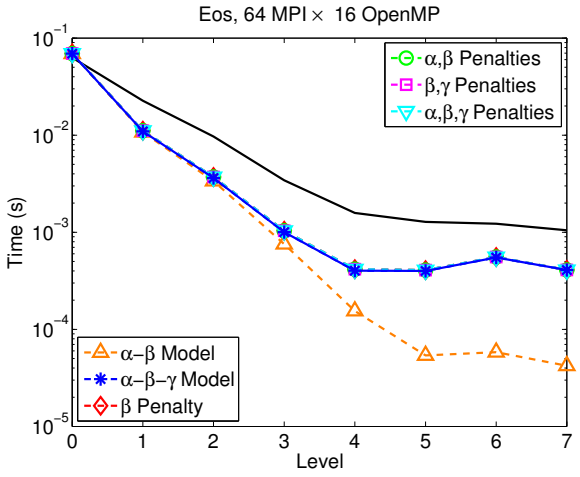


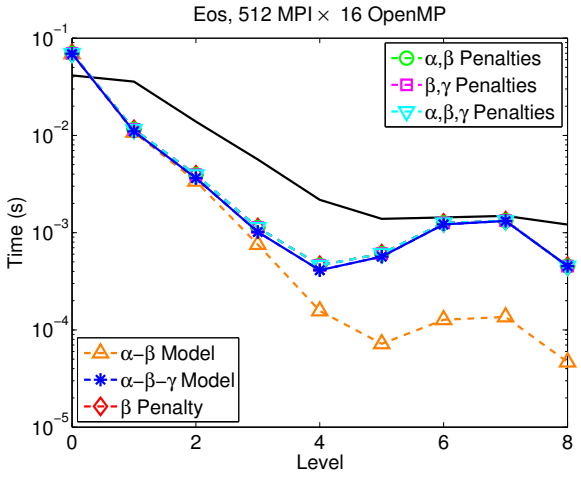


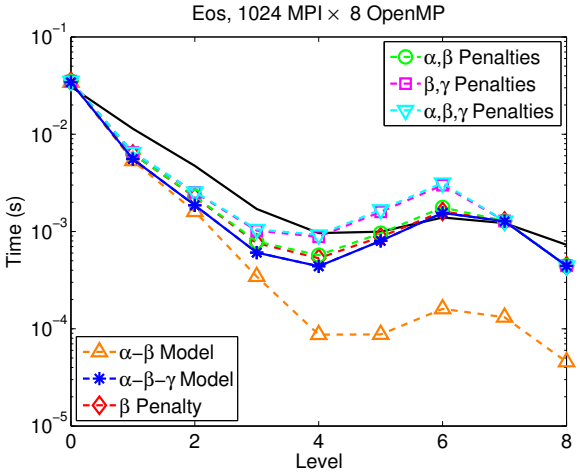


Modeling Results









Cycle Time Prediction Accuracies

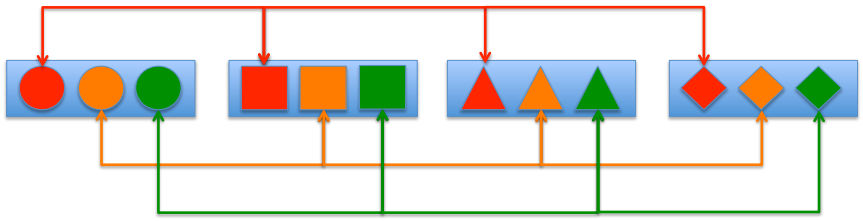
Mix	1024 Cores			8192 Cores		
	Modeled	Measured	Accuracy	Modeled	Measured	Accuracy
16 × 1	9.75 ms	11.3 ms	86.0%	11.7 ms	13.0 ms	90.4%
8 × 2	14.9 ms	16.3 ms	91.3%	16.8 ms	18.1 ms	92.8%
4 × 4	24.2 ms	27.6 ms	87.4%	26.5 ms	29.7 ms	89.2%
2 × 8	44.2 ms	51.8 ms	85.3%	46.7 ms	53.9 ms	86.6%
1 × 16	86.4 ms	104 ms	83.4%	88.4 ms	104 ms	85.9%

Modeling Results

- The good:
  - Best fit was α-β-γ with no additional penalties
  - No other machine we've tested was this good
  - Very little slowdown in cycle time going from 1K to 8K cores
- The concerning:
  - Substantial gap between α-β and best fit
  - γ (0.42 µs) was measured to be on par with α (0.24 µs), similar to a prior fat-tree machine which suffered from a lot of contention
  - Hybrid MPI/OpenMP performance also disappointing

Data Redistribution in AMG

- Idea that has gained traction:
  - Concentrate data on coarse grids so that fewer messages are sent
  - Has been done with or without redundant replication
- Illustration of redistribution strategy:



- Split problem domain into C chunks (blue boxes)
- Processes within a chunk have same part of domain
- Redundant version shown with 12 processes and 4 chunks
- Nonredundant version would keep just one color group

- Performance model can be adjusted to model this:
  - At level where redistribution is performed, charge for needed collective operations
  - On this and coarser levels, communication is with at most C-1 partners. Adjust computation based on amount of data concentrated
  - Adjust time per flop based on “problem size classification” (parts of data that fit in cache)

Guiding Data Redistribution

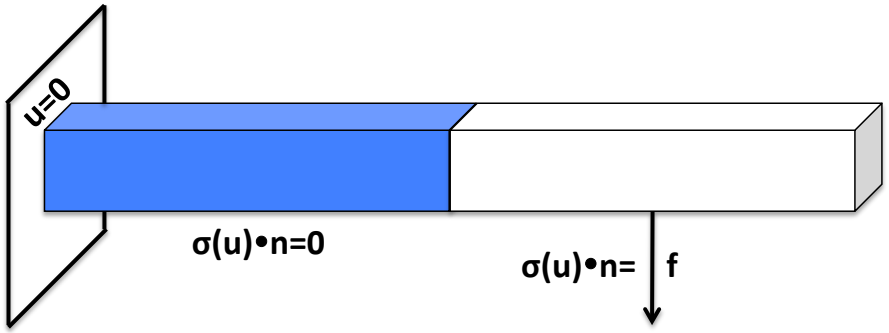
- At each coarse grid in setup phase, use model to estimate:
  1. Time spent at that level in solve cycle when redistributing (T_{switch})
  2. And when not redistributing ($T_{noswitch}$)
  3. If $T_{switch} < T_{noswitch}$, then redistribute

- Requires extra information:
  - Interpolation operator unavailable: substitute MatVec with solve operator
  - Time per flop unknown: measure with MatVecs using local portion of parallel data

- Other concerns:
  - Time per flop changes after redistribution: do not change it in model, but prevent redistribution if problem size classification increases
  - Hybrid MPI/OpenMP use? Requires essentially no change! Implicit in measurement of time per flop
  - Number of chunks to carve problem into? For quick setup, search powers of 2 <= max # sends
  - Possible overeager switching: keep track of running estimated cycle time, do not switch if overall modeled improvement is < 5%

Redistribution Experiments

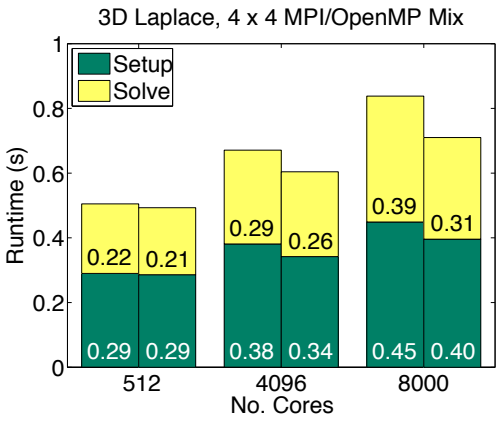
- Pair of test problems:
  1. 3D Laplace with 30 x 30 x 30 points/core
  2. Linear Elasticity with ~6,300 points/core:



- Used nonredundant redistribution owing to issues with large numbers of MPI communicators at scale
- Ran on Eos, like before

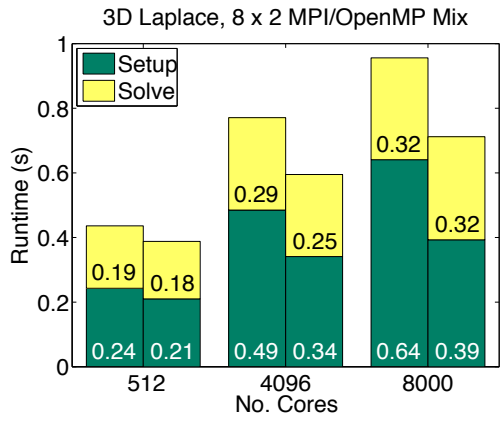
Redistribution Results

Laplace:



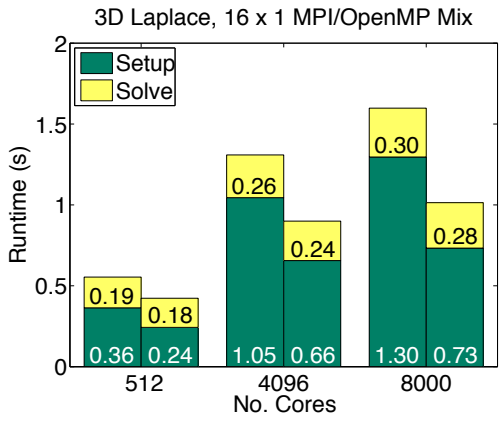
Speedups:

Cores	Setup	Solve	Overall
512	1.00	1.05	1.02
4096	1.12	1.12	1.12
8000	1.12	1.26	1.18



Speedups:

Cores	Setup	Solve	Overall
512	1.14	1.06	1.10
4096	1.44	1.16	1.32
8000	1.64	1.00	1.35



Speedups:

Cores	Setup	Solve	Overall
512	1.50	1.06	1.31
4096	1.59	1.08	1.46
8000	1.78	1.07	1.58

Linear Elasticity (all MPI):

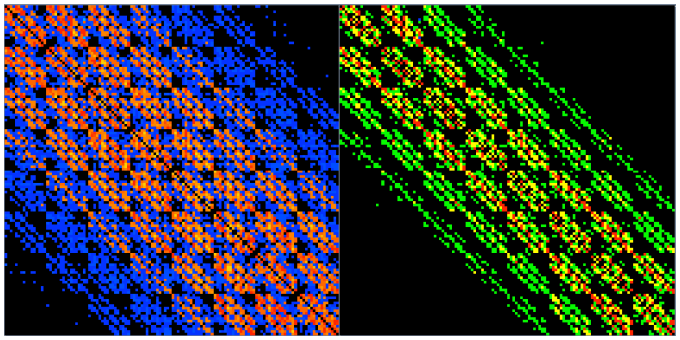
	1024 Cores			8192 Cores		
	Setup	Solve	Total	Setup	Solve	Total
No Redistribution	0.78 s	1.06 s	1.84 s	6.72 s	2.64 s	9.36 s
With Redistribution	0.75 s	0.82 s	1.58 s	2.99 s	1.55 s	4.54 s
Speedup	1.04	1.29	1.16	2.25	1.93	2.06

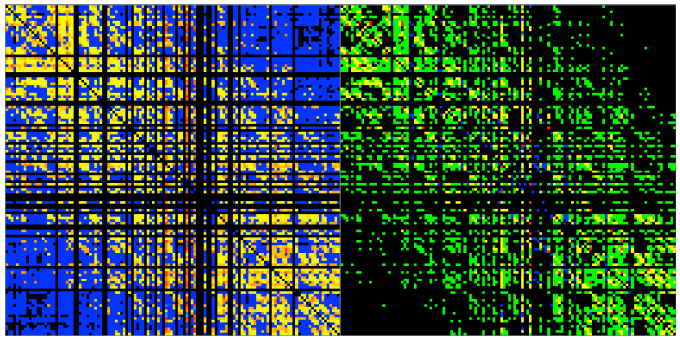
Observations

- Laplace problem: overall performance and redistribution benefit depended on phase
  - Solve phase: all-MPI was best, small gain from redistribution
  - Setup phase: introducing OpenMP helped performance; more MPI-rich configurations showed substantial gains from redistribution
- Linear elasticity problem: big redistribution benefit in both phases on 8K cores

Observations

- Common thread: bigger benefits from OpenMP or redistribution came in higher-communication settings
  - Laplace problem: setup vs. solve communication patterns for test problem from model validation on 128 cores on two most communication-intensive levels. Setup communicates much more:





  - Linear elasticity problem: much larger stencils (up to 500 nnz/row) than Laplace (up to 100 nnz/row)
- So even with interconnect's favorable rating by the model, its performance can dramatically degrade when communication is high

Conclusions and Future Work

- XC30 interconnect rated well in terms of contention penalties
- There were still, however, big benefits from reducing the number of messages sent
- Future work on XC30:
  - Model AMG setup phase
  - See if there is communication threshold at which substantial performance degradation occurs and map it if found

Conclusions and Future Work

- Results hint at dragonfly networks having problems with communication heavy applications:
  - Unifying feature of router groups with all-to-all links
  - Performance degradation and improvement from communication reduction was most pronounced when running on 8K cores (majority of the machine)
  - Large distance term in performance model
- Risk of slowdowns when communicating between different router groups that will need to be addressed to make topology effective at scale

Questions?

Acknowledgements

Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract DE-AC05-00OR22725. An award of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program.