Joint Laboratory
for Petascale Computation

INRIA NCSA

Topics for Collaboration in Numerical Libraries

William Gropp

www.cs.illinois.edu/~wgropp

Numerical Library and Algorithm Issues at Scale

- Barriers to Scalability
  - Communication
    - Total volume
    - Synchronizing communication (e.g., dot product)
    - Computation/communication balance (e.g., extra computation for communication)
      - Nonblocking vs. pipelines
  - Load balance
    - Static work decomposition
      - Coarse grain (partitioning), fine grain (loop decomposition)
    - Dynamic work decomposition
      - Low overhead, guided by communication activity

Numerical Library and Algorithm Issues at Scale

- Architectural Evolution
  - GPUs; new generation of vector/stream algorithms
  - CPU + GPU (heterogeneous)
- Barriers to Experimentation
  - Test cases and frameworks
- Barriers to Understanding
  - Performance models
- Barriers to adoption
  - Risk to adopters – cost/benefit analysis
  - Data structure changes

Purpose

- Bring complementary skills together to solve problems in numerical analysis for extreme scale platforms
- Current areas of interest and activity
  - Dense linear algebra
  - Sparse linear algebra and preconditioners
  - 3D FFTs
- Other areas of interest include
  - Alternatives to algorithms that use alltoall
  - Memory locality efficient methods for CPUs and GPUs
  - Heterogeneous-friendly algorithms
  - Latency tolerant or synchronization avoiding algorithms

Application Drivers

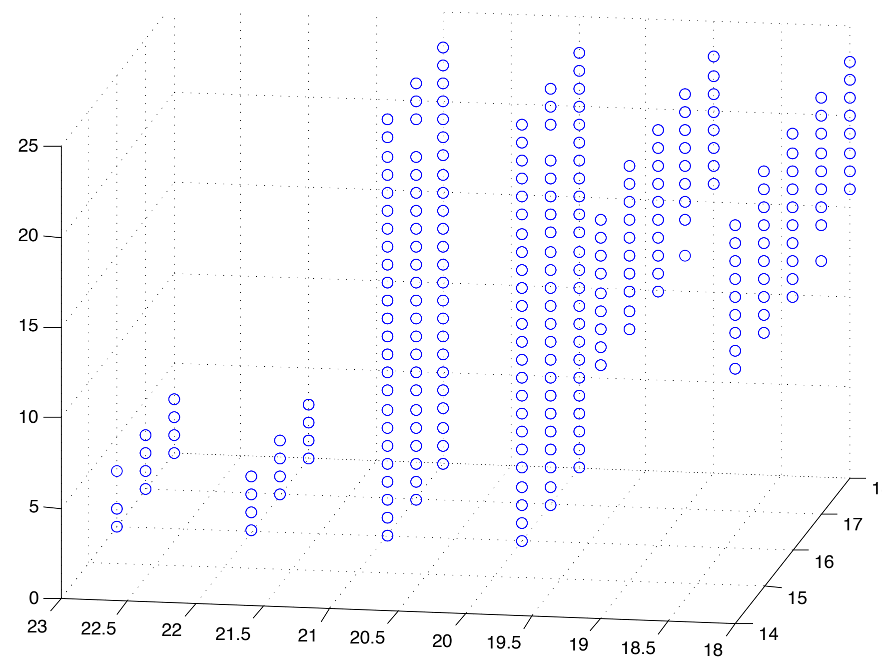
- Blue Waters applications (“PRAC”) provide good drivers
  - FFT for DNS, preconditioning for MILC
  - Need new algorithms for new execution model
    - GPUs have different memory model that further emphasizes medium-grain memory regularity
  - Top to bottom heterogeneity and irregularity of resource availability also requires new thinking

Some Current PRAC Requests

- Two of the PRAC teams are looking for faster Poisson solvers
  - One currently using FFT from UCSD P3DFFT
  - One using CG/ILU and looking at MG (CG/MG?)
- What teams need and what they want may not be the same
  - Alternative problem formulations?
  - CG without blocking synchronization?
  - Effective use of nearby solutions?

Algorithms and Topology

- Complex hierarchy:
  - Multiple chips per node; different access to local memory and to interconnect; multiple cores per chip
  - Mesh has different bandwidths in different directions
  - Allocation of nodes may not be regular (you are unlikely to get a compact brick of nodes)
  - Some nodes have GPUs
- Most algorithms designed for simple hierarchies and ignore network issues



Recent work on general topology mapping e.g.,

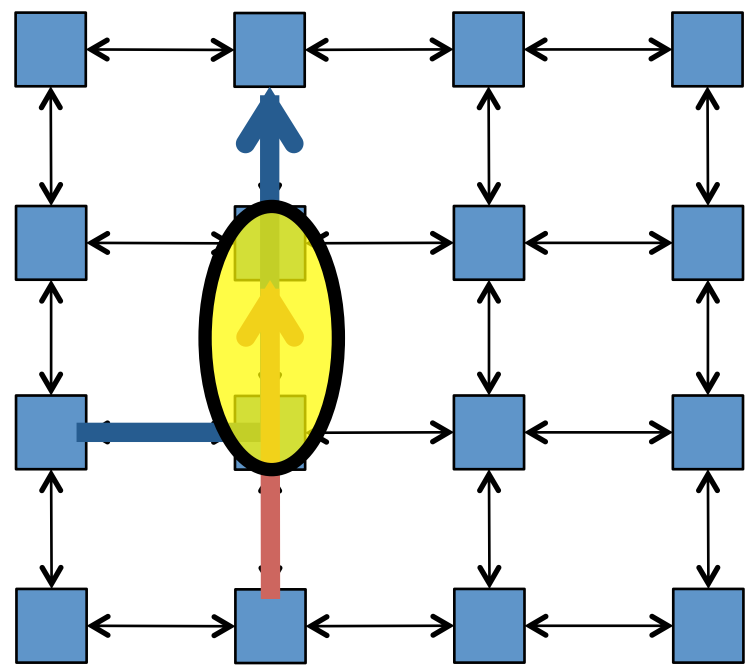
Generic Topology Mapping Strategies for Large-scale Parallel Architectures, Hoefler and Snir

Dynamic Workloads Require New, More Integrated Approaches

- Performance irregularities mean that classic approaches to decomposition are increasingly ineffective
  - Irregularities come from OS, runtime, process/thread placement, memory, heterogeneous nodes, power/clock frequency management
- Static partitioning tools can lead to persistent load imbalances
  - Mesh partitioners have incorrect cost models, no feedback mechanism
  - "Regrid when things get bad" won't work if the cost model is incorrect; also costly
- Basic building blocks must be more dynamic without introducing too much overhead

Communication Cost Includes More than Latency and Bandwidth

- Communication does not happen in isolation
- Effective bandwidth on shared link is ½ point-to-point bandwidth
- Real patterns can involve many more (integer factors)
- Loosely synchronous algorithms ensure communication cost is worst case



Halo Exchange on BG/Q and Cray XE6

- 2048 doubles to each neighbor
- Rate is MB/sec (for all tables)

BG/Q	**8 Neighbors**	
	Irecv/Send	Irecv/Isend
World	662	1167
Even/Odd	711	1452
1 sender		2873

Cray XE6	**8 Neighbors**	
	Irecv/Send	Irecv/Isend
World	352	348
Even/Odd	338	324
1 sender		5507

Discovering Performance Opportunities

- Lets look at a single process sending to its neighbors.
- Based on our performance model, we *expect* the rate to be roughly twice that for the halo (since this test is only sending, not sending and receiving)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	488	490	389	389
BG/P	1139	1136	892	892
BG/Q			2873	
XT3	1005	1007	1053	1045
XT4	1634	1620	1773	1770
XE6			5507	

Discovering Performance Opportunities

- Ratios of a single sender to all processes sending (in rate)
- *Expect* a factor of roughly 2 (since processes must also receive)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	2.24		2.01	
BG/P	3.8		2.2	
BG/Q			1.98	
XT3	7.5	8.1	9.08	9.41
XT4	10.7	10.7	13.0	13.7
XE6			15.6	15.9

- BG gives roughly double the halo rate. XTn and XE6 are much higher.
  - It should be possible to improve the halo exchange on the XT by scheduling the communication
  - Or improving the MPI implementation

Scaling Problems

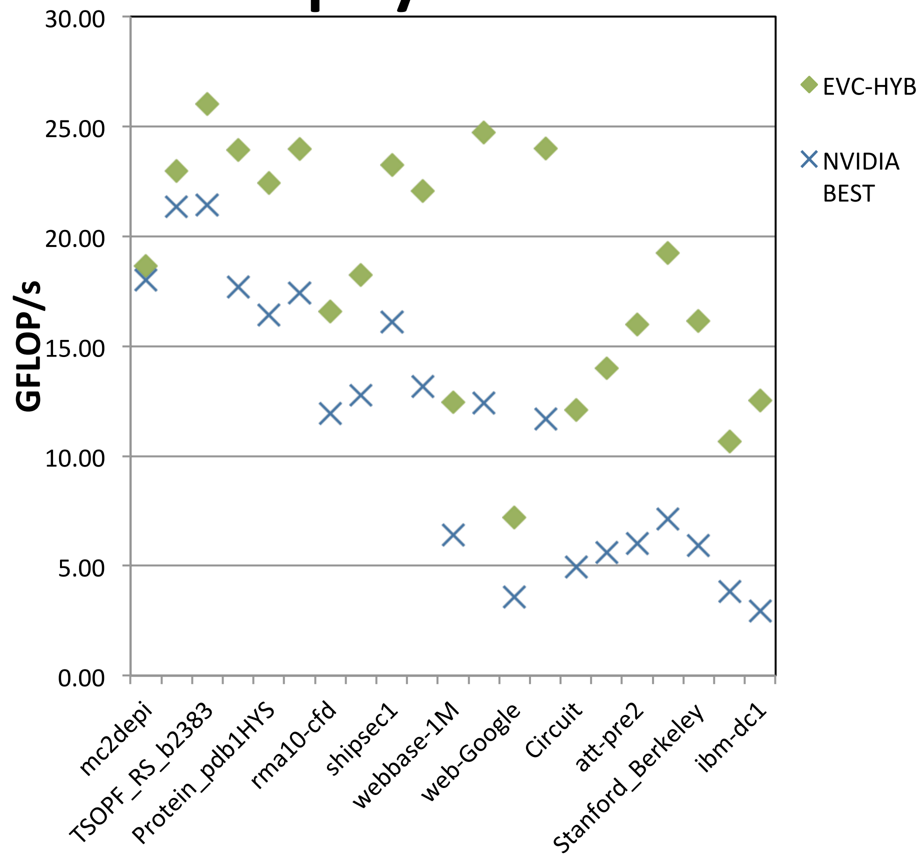
- Simple, data-parallel algorithms easy to reason about but inefficient
  – True for decades, but ignored (memory)
  – Log p terms can dominate at $p = 10^6$
- One solution: fully asynchronous methods
  – Very attractive (parallel efficiency high), yet solution efficiency is low and there are good reasons for that
  – Blocking (synchronizing) communication can be due to fully collective (e.g., Allreduce) or neighbor communications (halo exchange)
  – Can we save methods that involve global, synchronizing operations?

CG Reconsidered

- By reordering operations, nonblocking dot products (MPI_Iallreduce in MPI-3) can be overlapped with other operations
- Trades extra local work for overlapped communication
  - On a pure floating point basis, our nonblocking version requires 2 more DAXPY operations
  - A closer analysis shows that some operations can be merged (in terms of memory references)
    - Count *memory motion*, not *floating point*
- Other approaches possible; see "Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm," P. Ghysels and W. Vanroose.
- ***More work does not imply more time***

Sparse Matrix-vector Multiply on GPUs

- Continuation of work from last year by Dahai Guo
- Basic idea is a hybrid format, with adaptively distributed work (based on matrix structure)
- "Best of all worlds" approach
  - Faster than NVIDIA sparse matrix library
  - Robust performance
- Looking for applications!



Scheduling Large Systems

- Assigning jobs to nodes in a large system is a challenging problem
  - A version of a set assignment problem
  - Hard problem – but can use all unused nodes on parallel system to look for a better solution (power cost relatively low because of idle power)
  - Total number of queued jobs is not the correct measure of the problem size – often, many jobs are identical to the queuing system. # of different equivalence classes are a better measure
- User resource requests inaccurate, particularly time.
  - Would assigning jobs based on expected time produce a significantly different solution?
  - How would it interact with scheduling policy and guarantees?

Scheduling and Policy

- Policy constraints complicate the problem
  - Do they over-constrain the problem?
  - Under what assumptions can the achievable utilization be determined? How do changes in policy affect achievable utilization? Do elastic constraints rather than hard constraints significantly improve utilization?
- In all of this
  - What can we *prove?*
  - View as an optimization problem; use results from operations research, others

Summary

- Opportunities to impact running applications at scale
  - Looking for Poisson solvers, topology mappers, communication schedulers
  - Looking for applications needing SpMV on GPU, alternative CG (nonblocking)
- New challenge: a more effective, *mathematical* basis for effective job scheduling
- Need a "top 10" list of challenging numerical problems *at scale* – what's yours?
- Always looking for true big data problems that require 10us access to 1PB or more of data

Joint Laboratory
for Petascale Computation

INRIA NCSA

SC13 Denver, CO | 2013

sighpc

ACM Special Interest Group on High Performance Computing