2.5D algorithms: from hardware to theory and back

Edgar Solomonik

UC Berkeley

September 23rd, 2011

Outline

Ranger

Ranger

- TACC, Sun, 2008
- Commodity procs / commodity network
- 16 Opterons/node
  - 147.2 GF/node

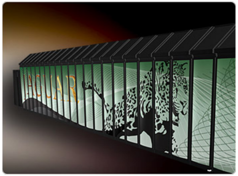
Ranger, Cray XT4

Ranger

- TACC, Sun, 2008
- Commodity procs / commodity network
- 16 Opterons/node
  - 147.2 GF/node

Cray XT4 (Jaguar)

- ORNL, Cray, 2009
- Commodity procs / custom network
- 4 Opterons per node
  - 32.8 GF/node



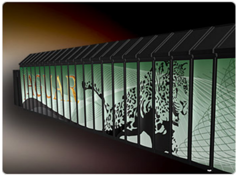
Ranger, Cray XT4, BG/P

Ranger

- TACC, Sun, 2008
- Commodity procs / commodity network
- 16 Opterons/node
  - 147.2 GF/node

Cray XT4 (Jaguar)

- ORNL, Cray, 2009
- Commodity procs / custom network
- 4 Opterons per node
  - 32.8 GF/node



BG/P (Intrepid)

- ANL, IBM, 2007
- Custom procs / custom network
- PowerPC 450 (4 cores/node)
  - 13.4 GF/node

Intra-node memory subsystems

Ranger

- 16 cores
- 32 GB/node (2 GB/core)
- 128 KB / 512 KB / 2 MB
- 21.3 MB/sec node bandwidth

Cray XT4 (Jaguar)

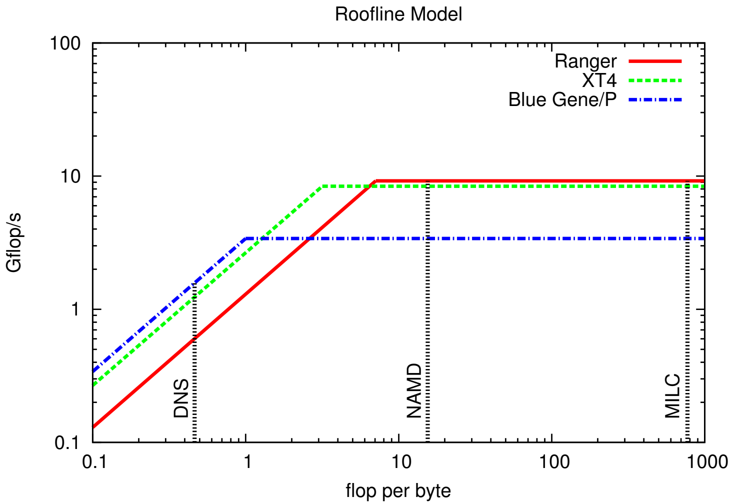
- 4 cores
- 8 GB/node (2 GB/core)
- 128 KB / 512 KB / 2 MB
- 10.6 MB/sec node bandwidth

BG/P (Intrepid)

- 4 cores
- 2 GB/node (0.5 GB/core)
- 64 KB / 2 KB / 8 MB
- 13.4 MB/sec node bandwidth

i

Roofline model



Network architecture

Ranger

- 3,936 nodes
- Infiniband
- full-CLOS (tree/switched)
- 1 GB/sec link bandwidth
- no topology aware scheduler

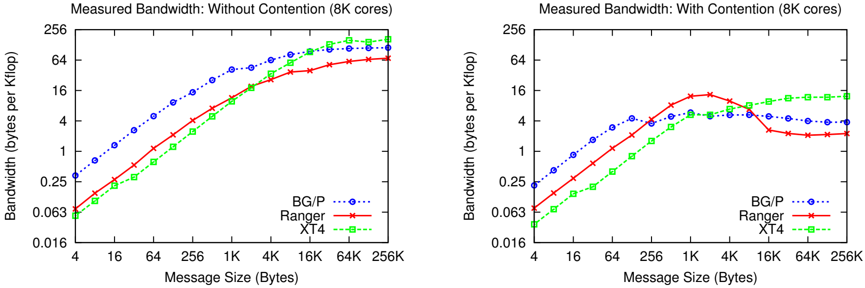
Cray XT4 (Jaguar)

- 7,832 nodes
- Seastar
- 3D torus (not topology-aware)
- 3.8 GB/sec link bandwidth
- no topology aware scheduler

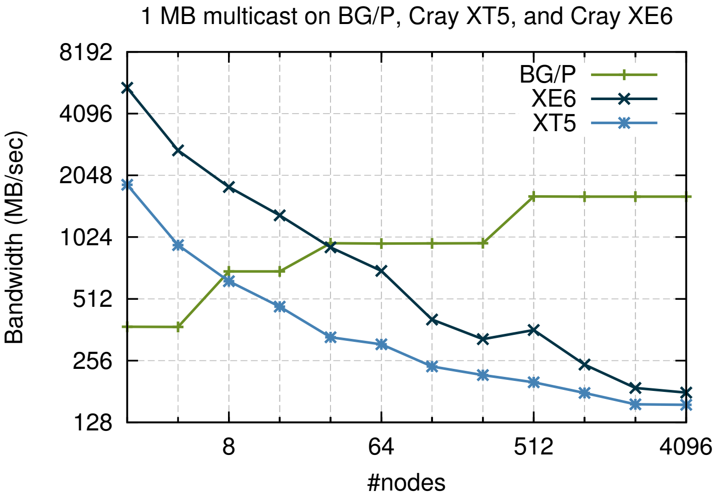
BG/P

- 40,960 nodes
- Custom
- 3D torus
- .425 GB/sec link bandwidth
- topology aware scheduler

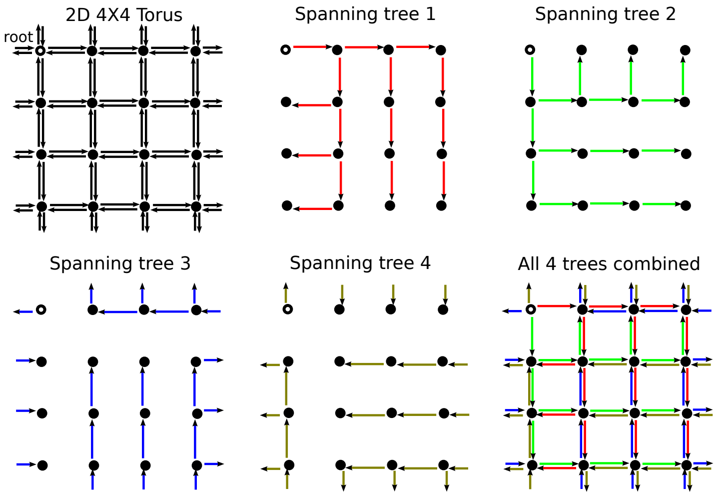
Point-to-point communication bandwidth



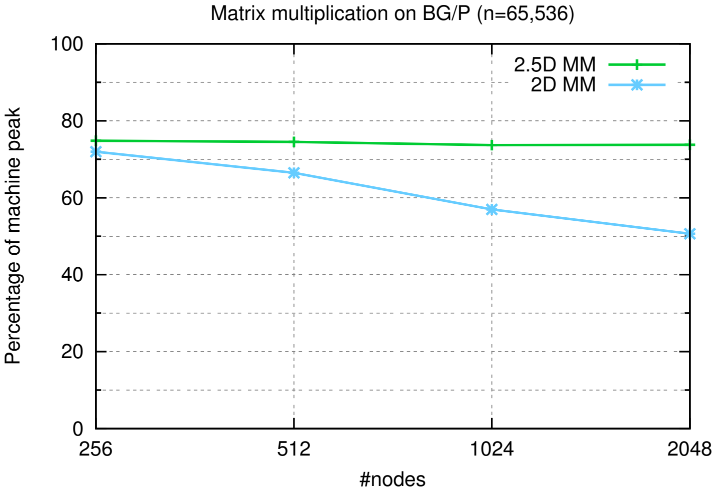
Collective communication (broadcast)



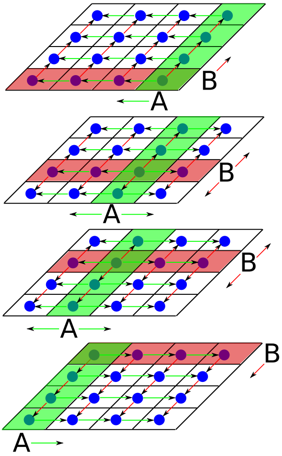
Rectangular broadcasts



Strong scaling matrix multiplication

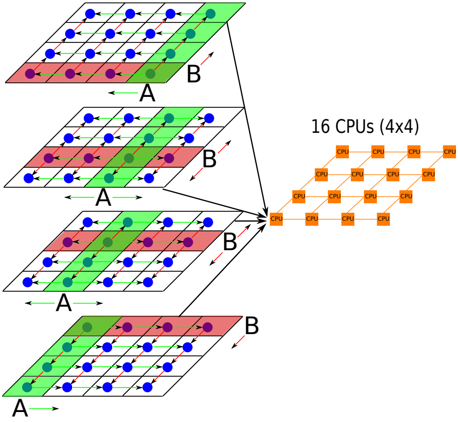


Blocking matrix multiplication



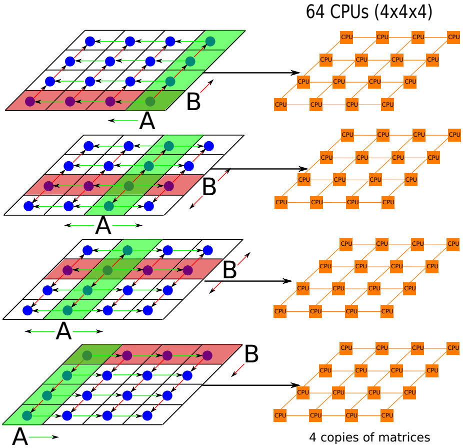
2D matrix multiplication

[Cannon 69], [Van De Geijn and Watts 97]

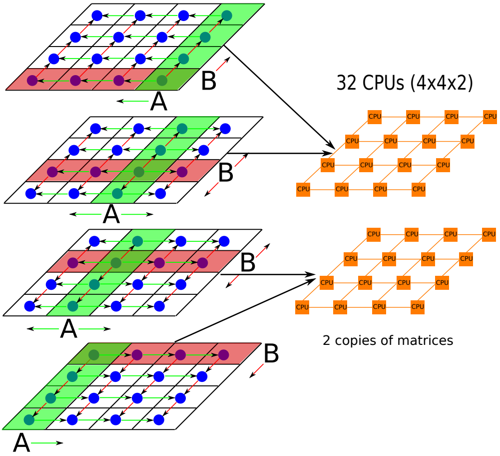


3D matrix multiplication

[Agarwal et al 95], [Aggarwal, Chandra, and Snir 90], [Bernsten 89]



2.5D matrix multiplication



2.5D strong scaling

n = dimension, p = #processors, c = #copies of data

- must satisfy $1 \leq c \leq p^{1/3}$
- special case: $c = 1$ yields 2D algorithm
- special case: $c = p^{1/3}$ yields 3D algorithm

$$\text{cost}(\text{2.5D MM}(p, c)) = O(n^3/p) \text{ flops} + O(n^2/\sqrt{c \cdot p}) \text{ words moved} + O(\sqrt{p/c^3}) \text{ messages}^*$$

*ignoring log(p) factors

2.5D strong scaling

n = dimension, p = #processors, c = #copies of data

- must satisfy $1 \leq c \leq p^{1/3}$
- special case: $c = 1$ yields 2D algorithm
- special case: $c = p^{1/3}$ yields 3D algorithm

$$\begin{aligned} \text{cost}(\text{2D MM}(p)) &= O(n^3/p) \text{ flops} \\ &\quad + O(n^2/\sqrt{p}) \text{ words moved} \\ &\quad + O(\sqrt{p}) \text{ messages}^* \\ &= \text{cost}(\text{2.5D MM}(p, 1)) \end{aligned}$$

*ignoring log(p) factors

2.5D strong scaling

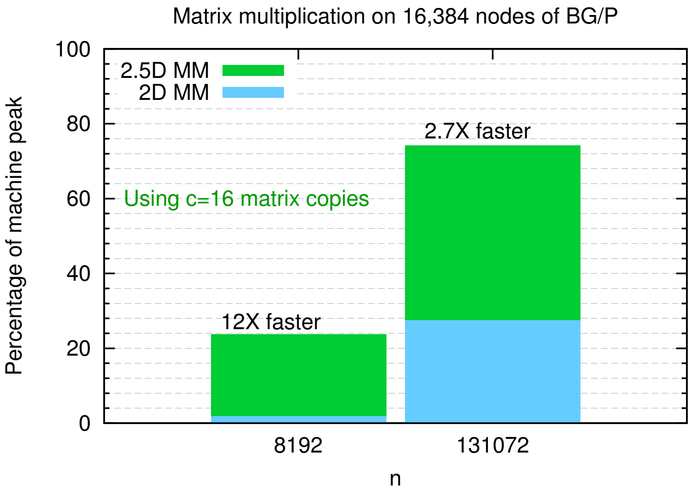
n = dimension, p = #processors, c = #copies of data

- must satisfy $1 \leq c \leq p^{1/3}$
- special case: $c = 1$ yields 2D algorithm
- special case: $c = p^{1/3}$ yields 3D algorithm

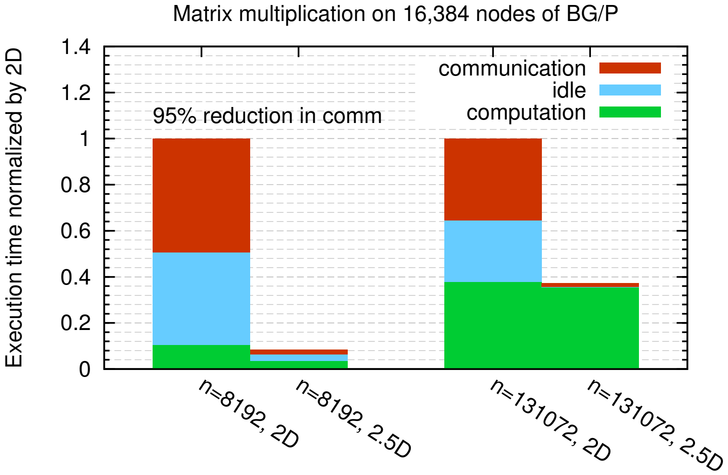
$$\begin{aligned} \text{cost}(2.5\text{D MM}(\mathbf{c} \cdot p, \mathbf{c})) = & \; O(n^3/(\mathbf{c} \cdot p)) \text{ flops} \\ & + O(n^2/(\mathbf{c} \cdot \sqrt{p})) \text{ words moved} \\ & + O(\sqrt{p}/\mathbf{c}) \text{ messages} \\ = & \; \text{cost}(2\text{D MM}(p))/\mathbf{c} \end{aligned}$$

perfect strong scaling

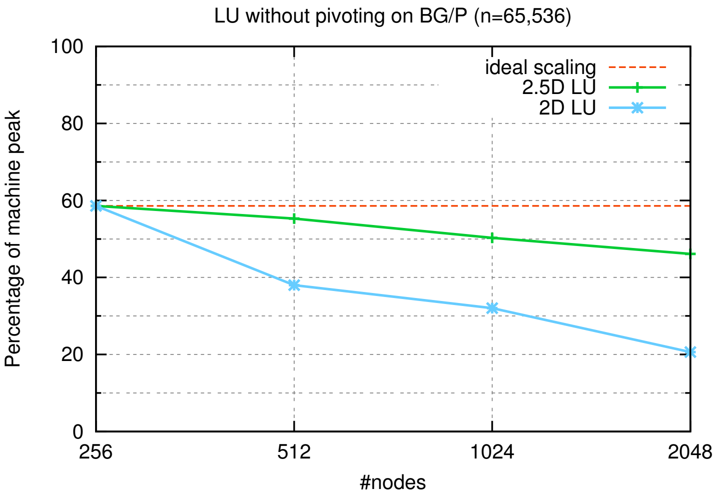
2.5D MM on 65,536 cores



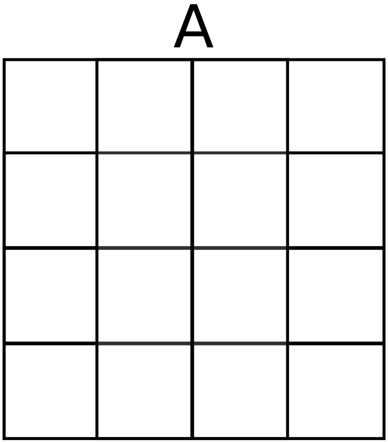
Cost breakdown of MM on 65,536 cores



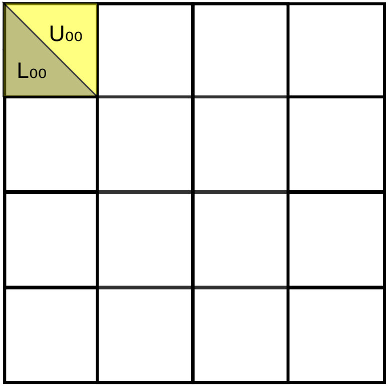
2.5D LU strong scaling (without pivoting)



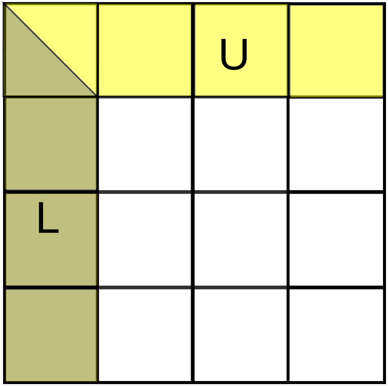
2D blocked LU factorization



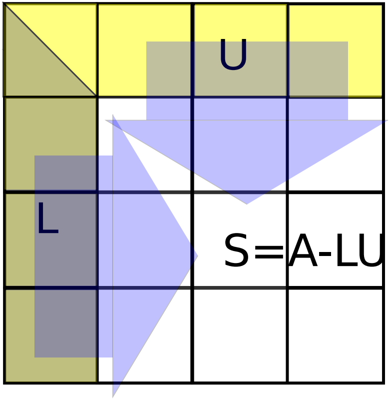
2D blocked LU factorization



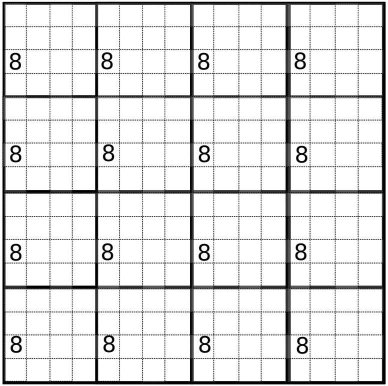
2D blocked LU factorization



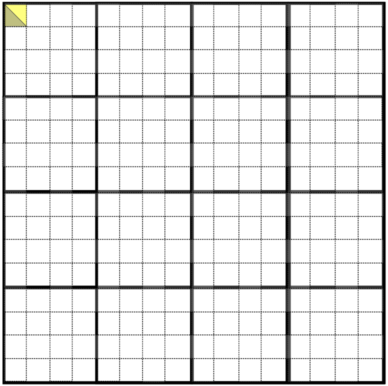
2D blocked LU factorization



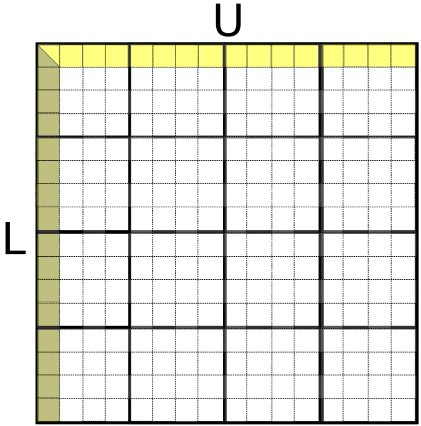
2D block-cyclic decomposition



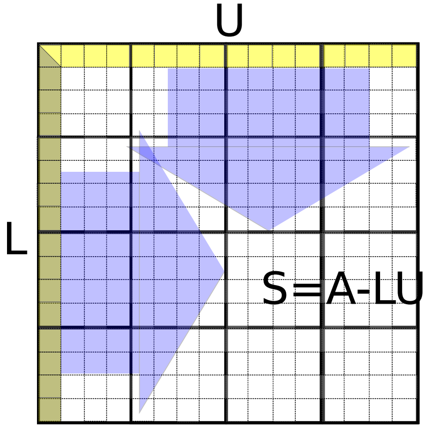
2D block-cyclic LU factorization



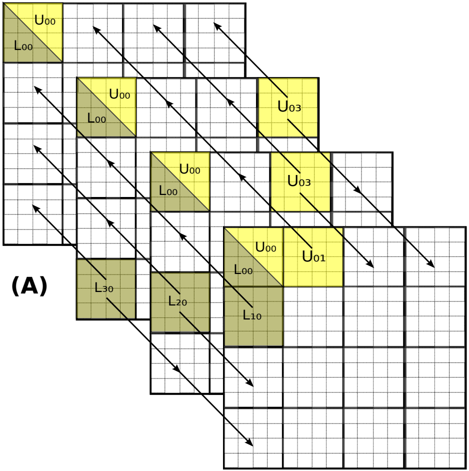
2D block-cyclic LU factorization



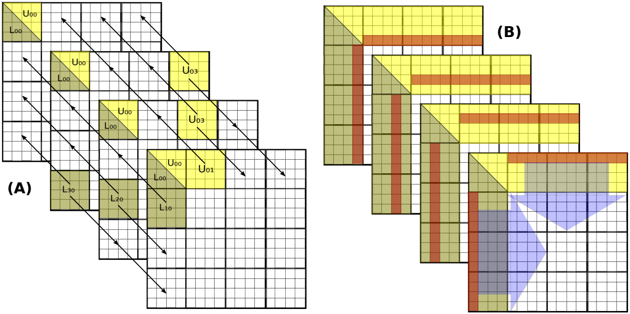
2D block-cyclic LU factorization



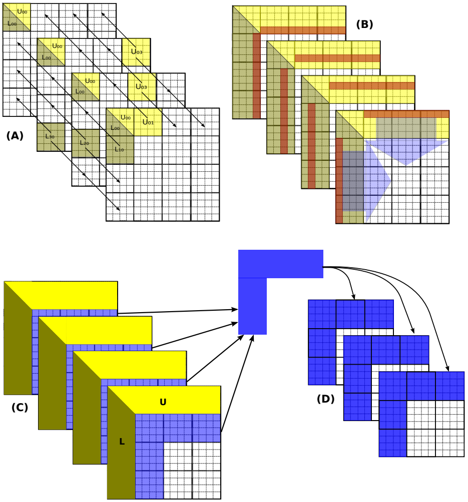
2.5D LU factorization



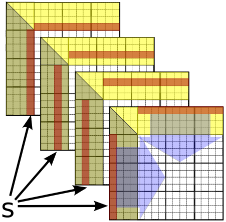
2.5D LU factorization



2.5D LU factorization



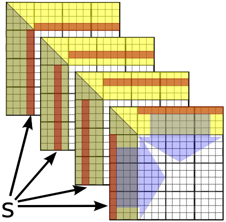
2.5D LU factorization



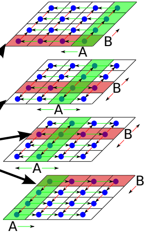
Look at how this update is distributed.

What does it remind you of?

2.5D LU factorization



Look at how this update is distributed.



Same 3D update in multiplication

Communication-avoiding pivoting

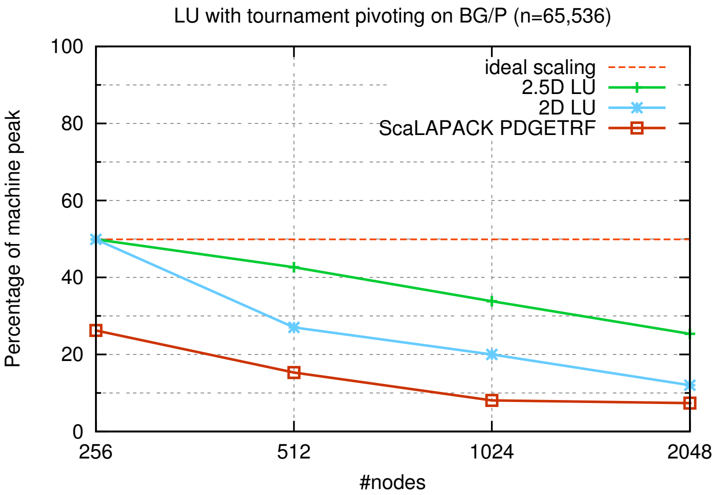
Partial pivoting is not communication-optimal on a blocked matrix

- require message/synchronization for each column
- $O(n)$ messages required

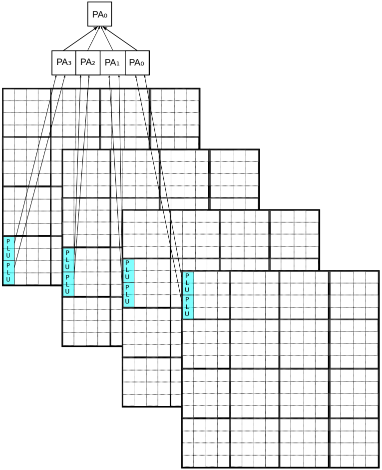
Tournament pivoting or Communication-Avoiding (CA) pivoting

- performs a tournament to determine best pivot row candidates
- blocked CA-pivoting algorithm is communication-optimal

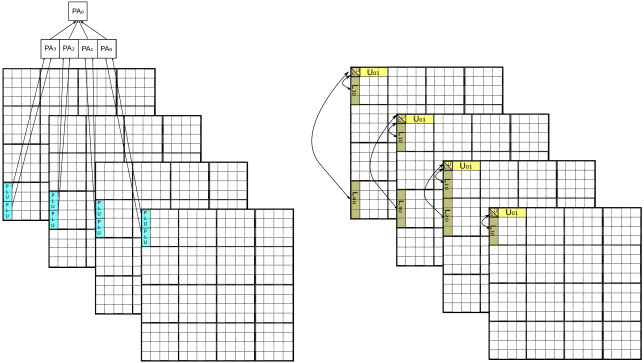
Strong scaling of 2.5D LU with tournament pivoting



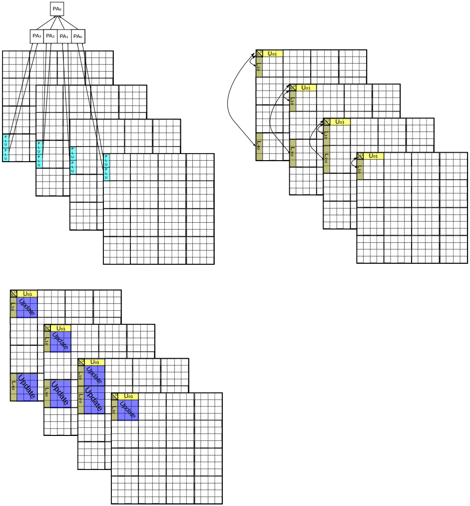
2.5D LU factorization with tournament pivoting



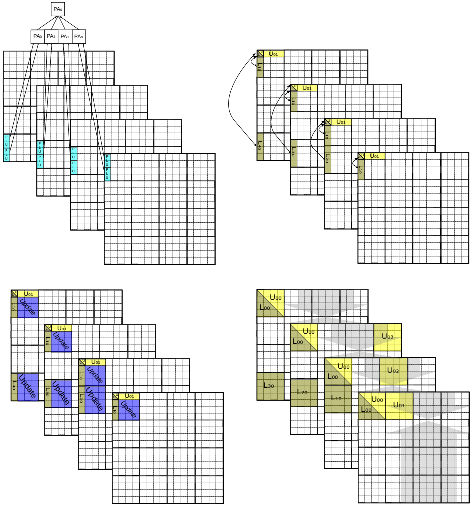
2.5D LU factorization with tournament pivoting



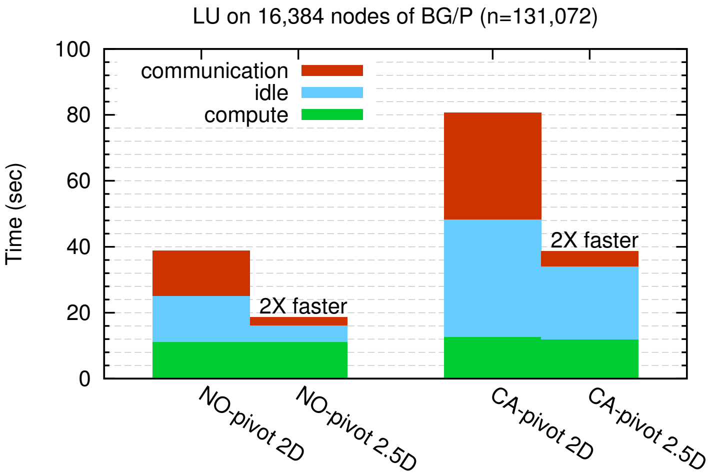
2.5D LU factorization with tournament pivoting



2.5D LU factorization with tournament pivoting



2.5D LU on 65,536 cores



Towards higher dimensions: tensor contractions

- Tensor contractions are a generalization of matrix multiplication (e.g.)

$$C_{cdef} = \sum_a \sum_b A_{cdab} \cdot B_{abef}$$

- Tensor contractions can be reduced to regular MM

$$C_{(cd)(ef)} = \sum_a \sum_b A_{(cd)(ab)} \cdot B_{(ab)(ef)}$$

$$C_{ij} = \sum_k A_{ik} \cdot B_{kj}$$

- Would like to support tensors up to dimensions 8-12

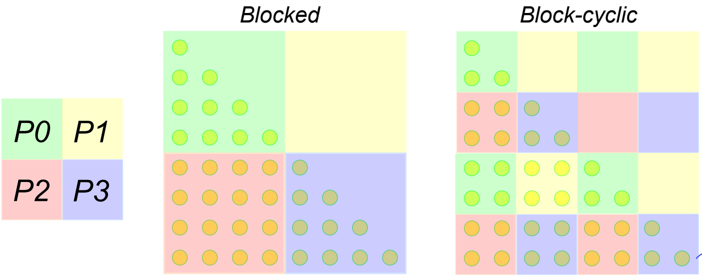
BLAS 4

Can we save communication by dealing with tensors explicitly rather than reducing to MM?

- Cannot improve flops/byte asymptotically over MM
- But *can* exploit higher-dimensional structure in tensors
- Higher-dimensional representation contains 'more information'

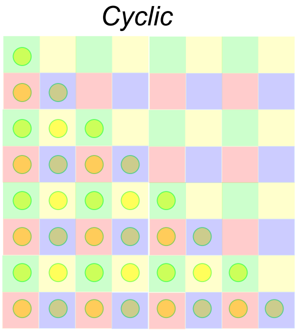
Symmetric tensor contractions

- A fully symmetric tensor of dimenson d requires only $n^d/d!$ storage
- Memory reduction also translates to communcation reduction via 2.5D
- Blocked or block-cyclic virtual processor decmpositions give irregular or imbalanced virtual grids



Solving the symmetry problem

- A **cyclic decomposition** allows balanced and regular blocking of symmetric tensors
- If the cyclic-phase is the same in each symmetric dimension, each sub-tensor retains the symmetry of the whole tensor



A generalized cyclic layout

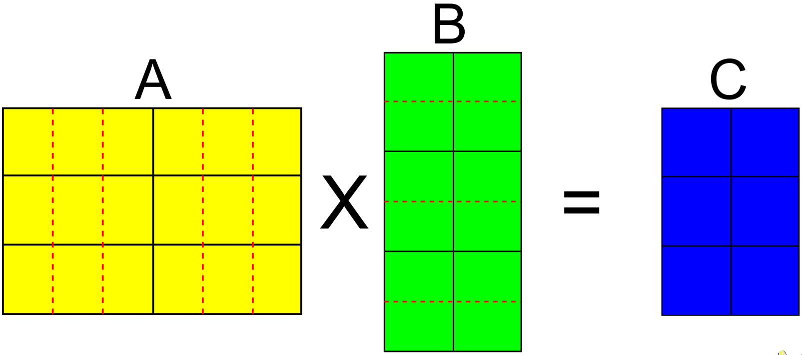
- In order to retain partial symmetry, all symmetric dimensions of a tensor must be mapped with the same cyclic phase
- The contracted dimensions of A and B must be mapped with the same phase
- And yet the virtual mapping, needs to be mapped to a physical topology, which can be any shape

Virtual processor grid dimensions

- Our virtual cyclic topology is somewhat restrictive and the physical topology is very restricted
- Virtual processor grid dimensions serve as a new level of indirection
  - If a tensor dimension must have a certain cyclic phase, adjust physical mapping by creating a virtual processor dimension
  - Allows physical processor grid to be 'stretchable'

Constructing a virtual processor grid for MM

Matrix multiply on 2x3 processor grid. Red lines represent virtualized part of processor grid. Elements assigned to blocks by cyclic phase.



Unfolding the processor grid

- Higher-dimensional fully-symmetric tensors can be mapped onto a lower-dimensional processor grid via creation of new virtual dimensions
- Lower-dimensional tensors can be mapped onto a higher-dimensional processor grid via by unfolding (serializing) pairs of processor dimensions
- However, when possible, replication is better than unfolding, since unfolded processor grids can lead to an unbalanced mapping

A basic parallel algorithm for symmetric tensor contractions

1. Arrange processor grid in any k-ary n-cube shape
2. Define and input (via unfold & virt) both A and B into cyclic layouts
3. Remap (via unfold & virt) both A and B cyclically along the dimensions being contracted
4. Remap (via unfold & virt) the remaining dimensions of A and B cyclically
5. For each tensor dimension contracted over, recursively mulitply the tensors along the mapping
   - Each contraction dimension is represented with a nested call to a local multiply or a parallel algorithm (e.g. Cannon)

Tensor library structure

The library supports arbitrary-dimensional parallel tensor contractions with any symmetries on n-cuboid processor torus partitions

1. Load and map tensor data by (global rank, value) pairs
2. Once a contraction is defined, remap participating tensors
3. Distribute or reshuffle tensor data/pairs
4. Construct contraction algorithm with recursive function/args pointers
5. Contract the sub-tensors with a user-defined sequential contract function
6. Output (global rank, value) pairs on request

Current tensor library status

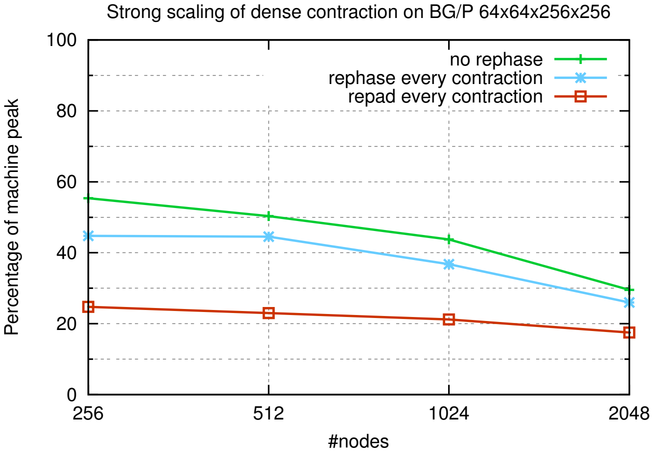
- Dense and symmetric remapping/repadding/contractions implemented
- Currently tuning an efficient symmetric transpose kernel
- Can perform automatic mapping with physical and virtual dimensions, but cannot unfold processor dimensions yet
- Complete library interface implemented, including basic auxillary functions (e.g. map/reduce, sum, etc.)

Next implementation steps

- Currently integrating library with a SCF method code that uses dense contractions
- Automatic unfolding of processor dimensions
- Implement mapping by replication to enable 2.5D algorithms
- Integrate with a sequential symmetric contraction library

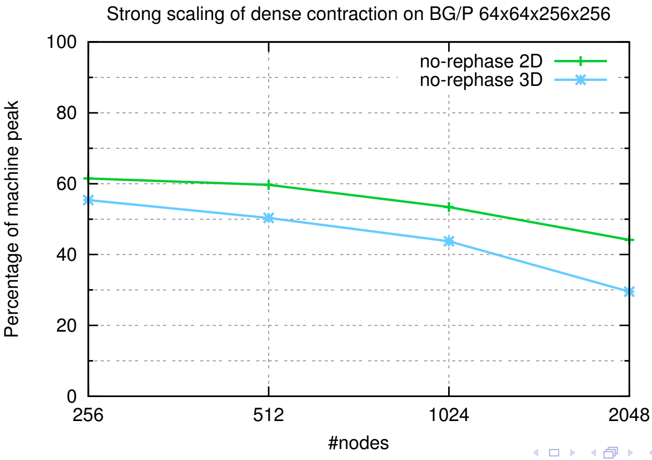
Very preliminary contraction library results

Contracts tensors of size 64x64x256x256 in 1 second on 2K nodes



Potential benefit of unfolding

Unfolding smallest two BG/P torus dimensions improves performance.



A new generation of machines: BlueWaters

BlueWaters

- NCSA, IBM, (scheduled 2011)
- Power 7 processors
- Hierarchical network
- 10 PF installation
  Cancelled

A new generation of machines: Cray XE6

Cray XE6 (Hopper)

- NERSC, Cray, 2011
- 2 twelve-core AMD MagnyCours/node
- 6,384 nodes
- 2.9/5.8 GB/sec per link
- **3D Torus** (Gemini)

A new generation of machines: Cray XE6, K computer

Cray XE6 (Hopper)

- NERSC, Cray, 2011
- 2 twelve-core AMD MagnyCours/node
- 6,384 nodes
- 2.9/5.8 GB/sec per link
- **3D Torus** (Gemini)

K computer

- RIKEN, Japan, Fujitsu, 2011
- 68,544 2 GHz 8-core SPARC64 nodes
- 5 GB/sec per link
- **6D Torus** (Tofu)

Cray XE6, K computer, BG/Q

Cray XE6 (Hopper)

- NERSC, Cray, 2011
- 2 twelve-core AMD MagnyCours/node
- 6,384 nodes
- 2.9/5.8 GB/sec per link
- **3D Torus** (Gemini)

K computer

- RIKEN, Japan, Fujitsu, 2011
- 68,544 8-core SPARC64 nodes
- 5 GB/sec per link
- **6D Torus** (Tofu)

BG/Q

- IBM, 2012
- 16 cores (205 GF/s)
- 98,304 nodes (20 PF
- 2 GB/sec per link
- **5D Torus**

More supercomputers...

- Titan (ORNL) 2012?
  - Gemini 3D torus interconect
  - GPUs
  - 20 PF
- Tianhe 1 (China) 2010
  - GPU accelerated
  - fat-tree network
  - 2.5 PF
- Stampede (TACC) 2011?
  - Intel MIC (Many Integrated Core)
  - Infiniband cluster
  - 10 PF

Supercomputing in higher-dimensionality

- Higher-dimensional interconnects
  - 3D Torus networks wide-spread
  - 5D/6D next-gen machines (BG/Q, K-computer)
  - higher bisection bandwidth and scalability
- Higher-dimensional algorithms
  - avoid communication with higher-dimensional blocking
    - 2.5D algorithms
  - avoid communication by exploiting higher-dimensional structure
    - tensor symmetry
    - conjecture: tensor sparsity

Current HPC programming models

- PGAS models are 1D
  - flat memory
  - weak notion of spacial locality
- Hierarchical models
  - cache-oblivious algorithms, recursive algorithms
  - express hierarchical spacial locality
  - natural for tree networks, not torus networks

A higher-dimensional programming model

- Decompose problem dimensionally
  - communicate along dimensions
  - maintain dimensionality of original problem
- Efficient dimensional communication primitives
  - reductions/broadcasts (rectangular algorithms)
  - replication (2.5D algorithms)
  - reconfiguration/transposition (convert tensor mappings)
  - virtualization (map to architecture)
- Advantages
  - use higher-dimensional blocking to reduce communication
  - use higher-dimensional structure to conserve memory/communication
  - maps communication directly to torus networks to reduce contention

Backup slides

Conclusion

Our contributions:

- 2.5D mapping of matrix multiplication
  - Optimal according to lower bounds in [Irony, Tiskin, Toledo 04] and [Aggarwal, Chandra, and Snir 90]
- A new latency lower bound for LU
- Communication-optimal 2.5D LU
  - Bandwidth-optimal according to general lower bound [Ballard, Demmel, Holtz, Schwartz 10]
  - Latency-optimal according to new lower bound

Open questions:

- 2.5D Householder QR

Reflections:

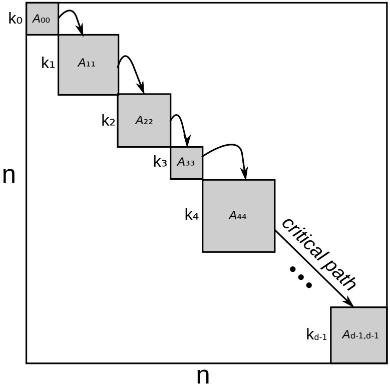
- Replication allows better strong scaling
- Topology-aware mapping cuts communication costs

A new latency lower bound for LU

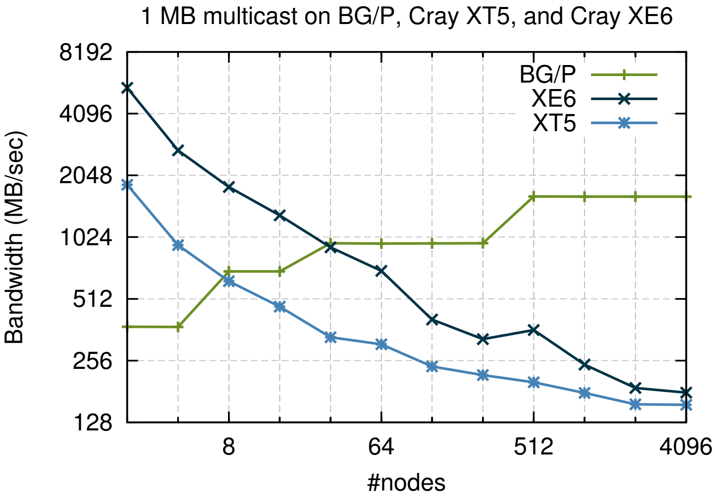
LU with $O(\sqrt{P/c^3})$ messages?

- For block size $n/\mathbf{d}$ LU does
  - $\Omega(n^3/\mathbf{d}^2)$ flops
  - $\Omega(n^2/\mathbf{d})$ words
  - $\Omega(\mathbf{d})$ msgs
- Now pick $\mathbf{d}$ (=latency cost)
  - $\mathbf{d} = \mathbf{\Omega}(\sqrt{\mathbf{P}})$ to minimize flops
  - $\mathbf{d} = \mathbf{\Omega}(\sqrt{\mathbf{c} \cdot \mathbf{P}})$ to minimize words

No dice. Lets minimize bandwidth.



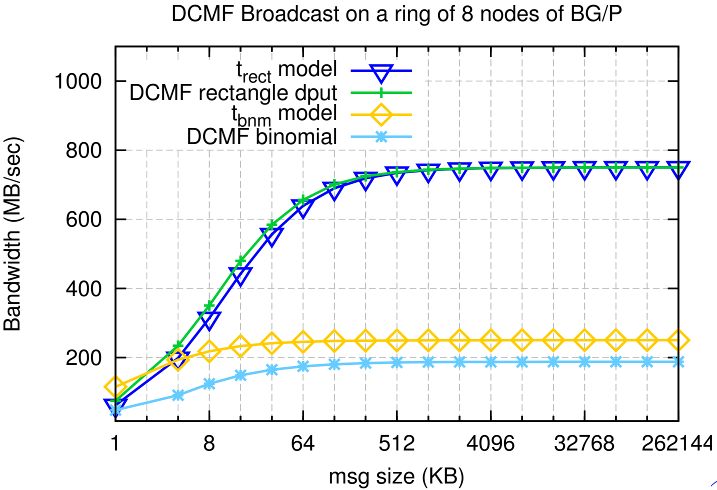
Performance of multicast (BG/P vs Cray)



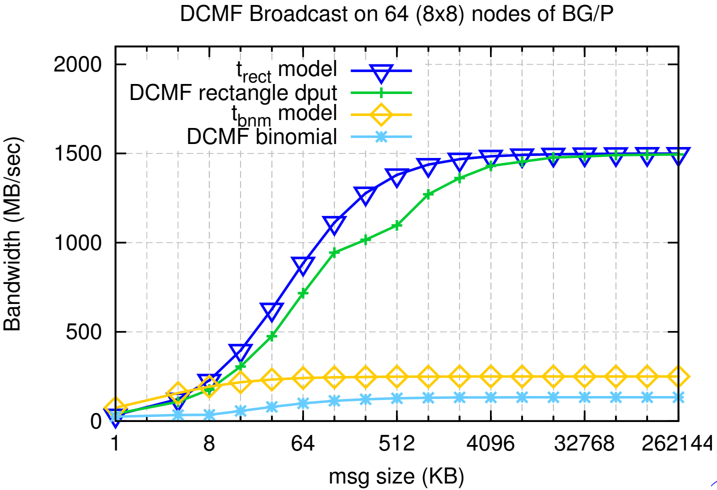
Why the performance discrepancy in multicasts?

- Cray machines use **binomial multicasts**
  - Form spanning tree from a list of nodes
  - Route copies of message down each branch
  - Network contention degrades utilization on a 3D torus
- BG/P uses **rectangular multicasts**
  - Require network topology to be a k-ary n-cube
  - Form $2n$ edge-disjoint spanning trees
    - Route in different dimensional order
    - Use both directions of bidirectional network

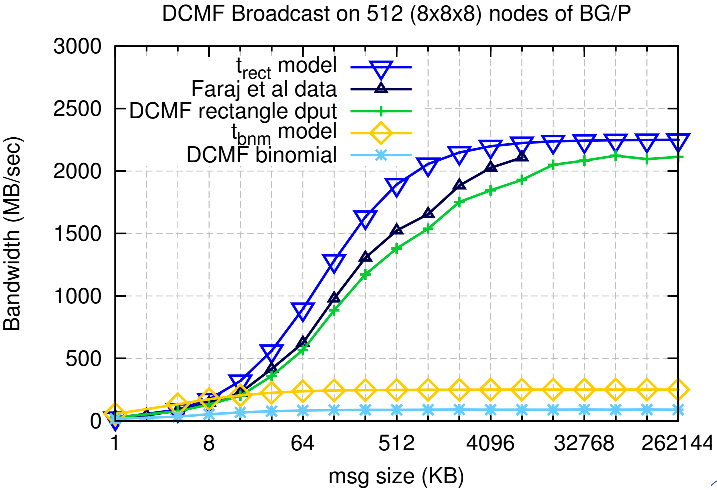
Model verification: one dimension



Model verification: two dimensions



Model verification: three dimensions

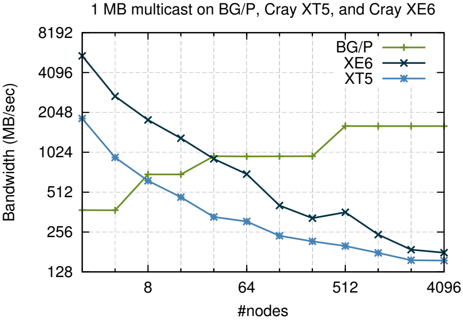


Another look at that first plot

Just how much better are rectangular algorithms on $P = 4096$ nodes?

- Binomial collectives on XE6
  - **1/30th of link bandwidth**
- Rectangular collectives on BG/P
  - **4.3X the link bandwidth**
- **Over 120X improvement in efficiency!**

How can we apply this?



Decoupling memory usage and topology-awareness

- 2.5D algorithms couple memory usage and virtual topology
  - c copies of a matrix implies c processor layers
- Instead, we can nest 2D and/or 2.5D algorithms
- Higher-dimensional algorithms allow smarter topology aware mapping

4D SUMMA-Cannon

How do we map to a 3D partition *without using more memory*

- SUMMA (bcast-based) on 2D layers
- Cannon (send-based) along third dimension
- Cannon calls SUMMA as sub-routine
  - Minimize inefficient (non-rectangular) communication
  - Allow better overlap
- Treats MM as a 4D tensor contraction