# 2.5D algorithms for distributed-memory computing

Edgar Solomonik

UC Berkeley

July, 2012

# Outline

Introduction
- Strong scaling

2.5D dense linear algebra
- 2.5D matrix multiplication
- 2.5D LU factorization
- 2.5D QR factorization

All-pairs shortest-paths

Symmetric tensor contractions

Conclusion

# Solving science problems faster

Parallel computers can solve bigger problems

- **weak scaling**

Parallel computers can also solve a fixed problem faster

- **strong scaling**

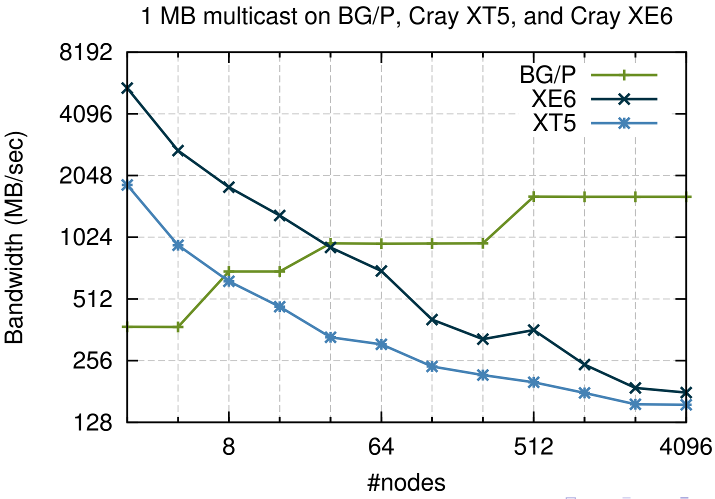
Obstacles to strong scaling

- may increase relative cost of **communication**
- may hurt **load balance**

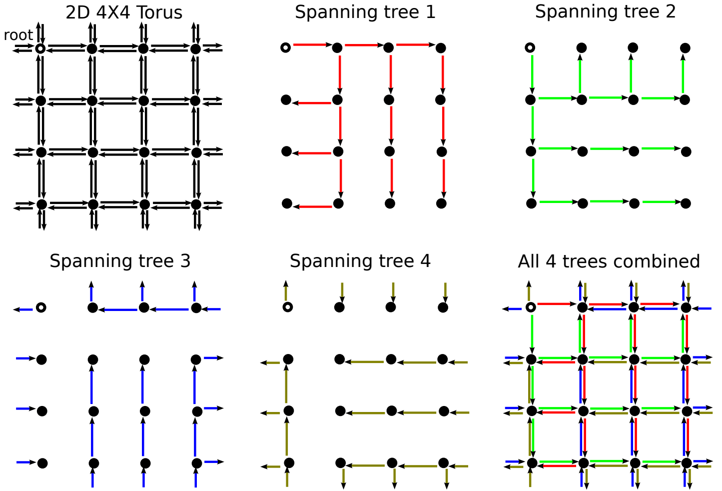
How to reduce communication and maintain load balance?

- reduce (minimize) communication along the **critical path**
- exploit the **network topology**

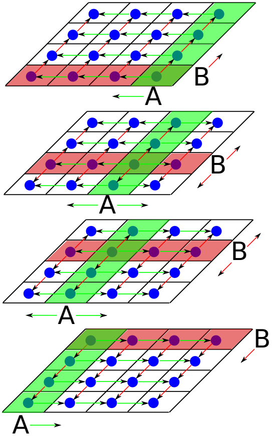
# Topology-aware multicasts (BG/P vs Cray)



# 2D rectangular multicasts trees

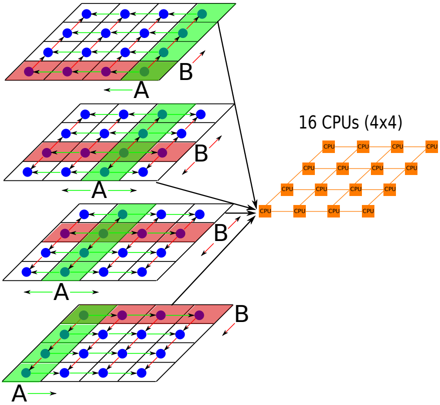


# Blocking matrix multiplication



# 2D matrix multiplication

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




$O(n^3/p)$ flops

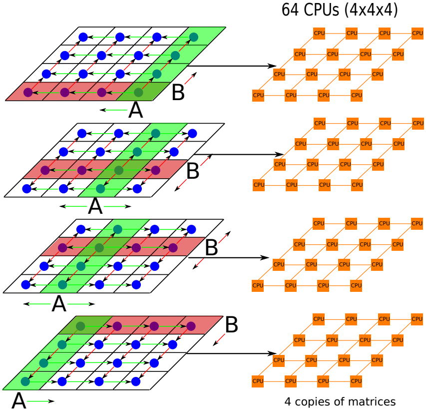
$O(n^2/\sqrt{p})$ words moved

$O(\sqrt{p})$ messages

$O(n^2/p)$ bytes of memory

# 3D matrix multiplication

[Agarwal et al 95],
[Aggarwal, Chandra, and Snir 90],
[Bernsten 89], [McColl and Tiskin 99]




$O(n^3/p)$ flops

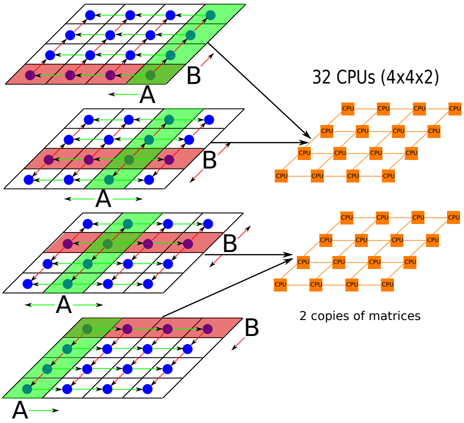
$O(n^2/p^{2/3})$ words moved

$O(1)$ messages

$O(n^2/p^{2/3})$ bytes of memory

# 2.5D matrix multiplication

[McColl and Tiskin 99]




$O(n^3/p)$ flops

$O(n^2/\sqrt{c \cdot p})$ words moved

$O(\sqrt{p/c^3})$ messages

$O(c \cdot n^2/p)$ bytes of memory

# 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

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

*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

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

*ignoring log(p) factors

# 2.5D strong scaling

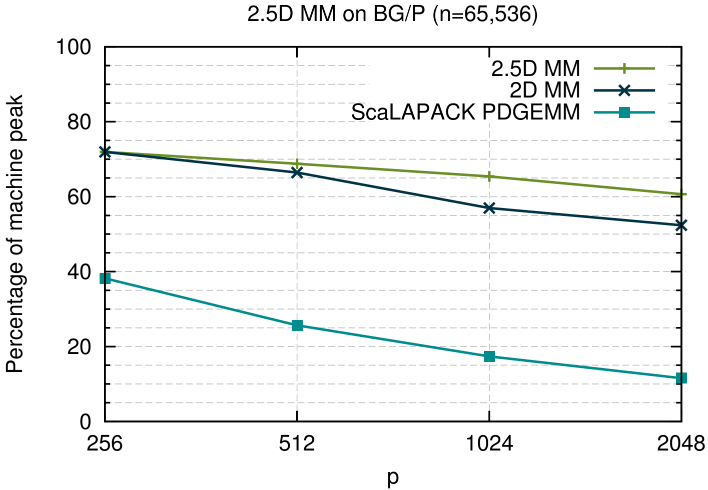
n = dimension, p = #processors, c = #copies of data

- 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

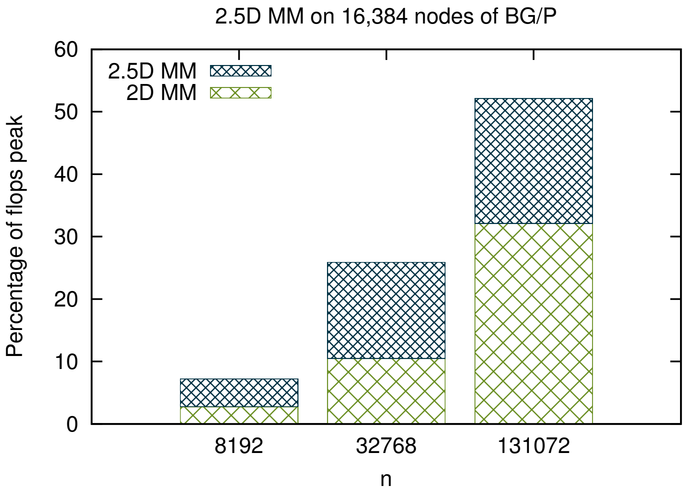
$$\begin{aligned}\text{cost}(\text{2.5D MM}(\mathbf{c} \cdot p, \mathbf{c})) &= O(n^3/(\mathbf{c} \cdot p)) \text{ flops} \\ &\quad + O(n^2/(\mathbf{c} \cdot \sqrt{p})) \text{ words moved} \\ &\quad + O(\sqrt{p}/\mathbf{c}) \text{ messages} \\ &= \text{cost}(\text{2D MM}(p))/\mathbf{c}\end{aligned}$$

**perfect strong scaling**

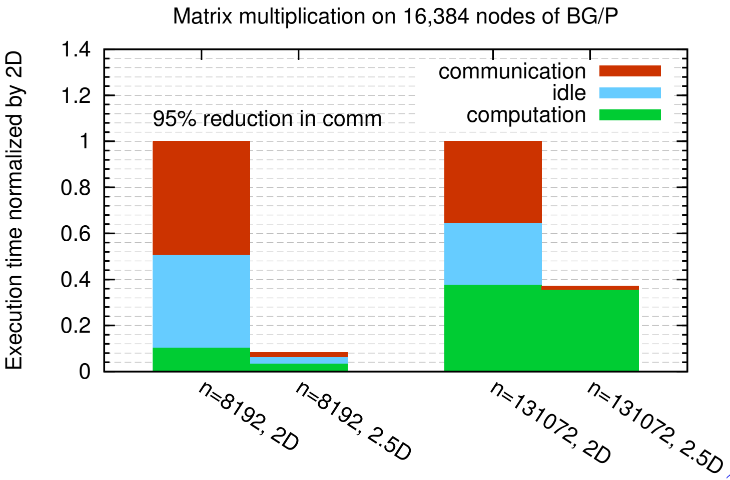
# Strong scaling matrix multiplication



# 2.5D MM on 65,536 cores



# Cost breakdown of MM on 65,536 cores

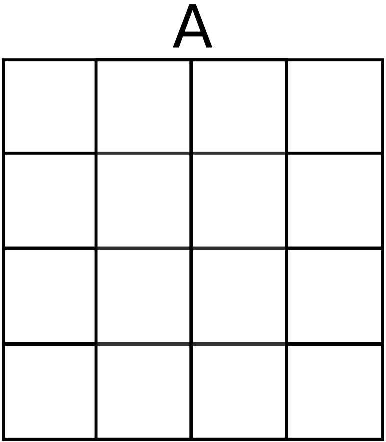


# 2.5D recursive LU

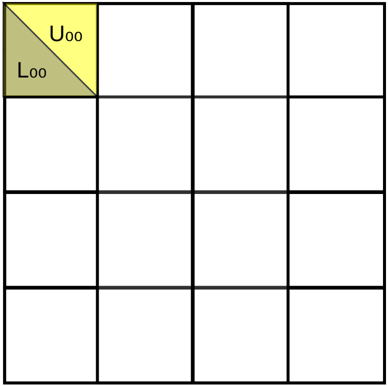
$A = L \cdot U$ where $L$ is lower-triangular and $U$ is upper-triangular

- A 2.5D recursive algorithm with no pivoting [A. Tiskin 2002]
- Tiskin gives algorithm under the BSP model
  - Bulk Synchronous Parallel
  - considers communication and synchronization
- We give an alternative distributed-memory adaptation and implementation
- Also, we lower-bound the latency cost

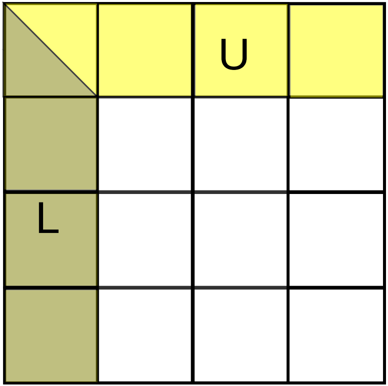
# 2D blocked LU factorization



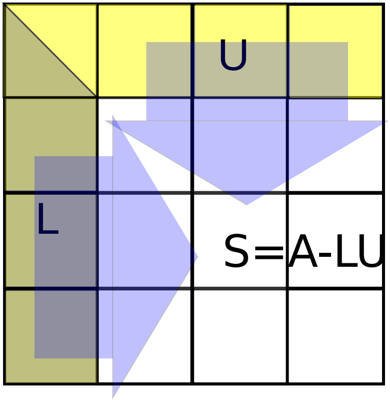
# 2D blocked LU factorization



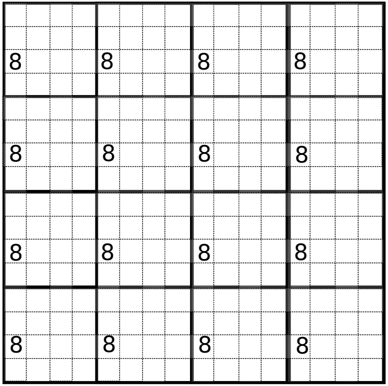
# 2D blocked LU factorization



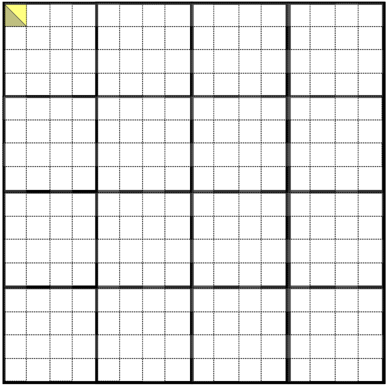
# 2D blocked LU factorization



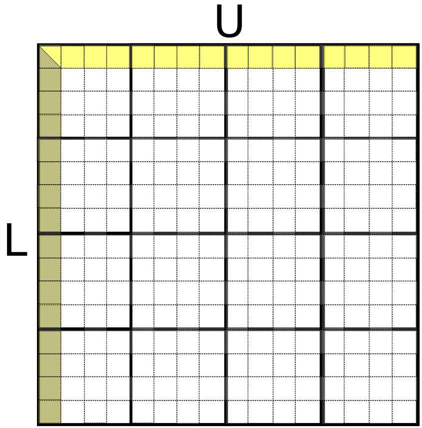
# 2D block-cyclic decomposition



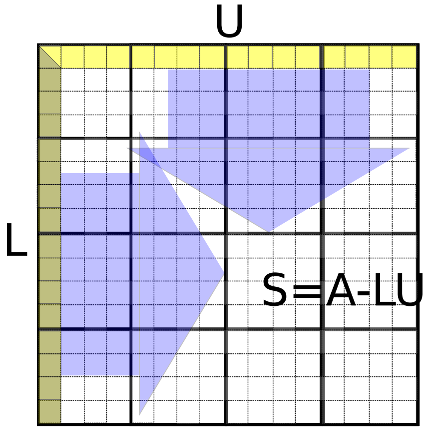
# 2D block-cyclic LU factorization



# 2D block-cyclic LU factorization

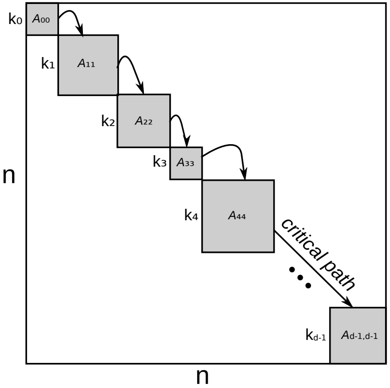


# 2D block-cyclic LU factorization

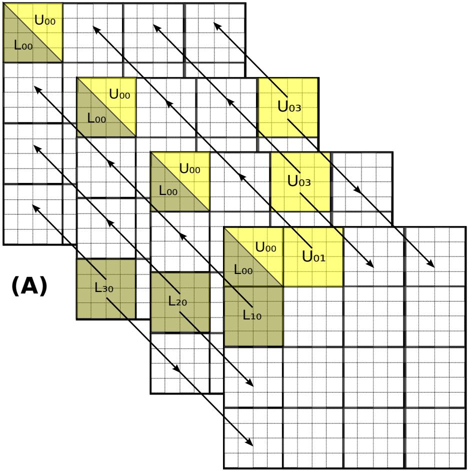


# A new latency lower bound for LU

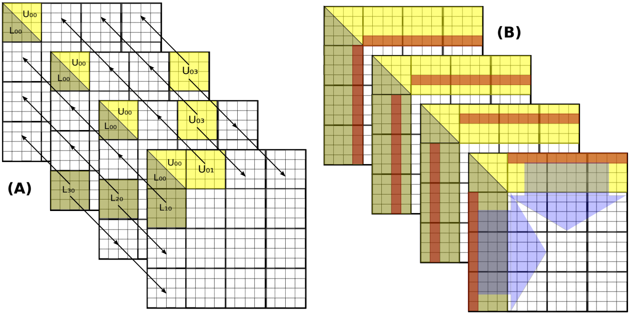
- Relate volume to surface area to diameter
- For block size $n/\mathbf{d}$ LU does
  - $\Omega(n^3/\mathbf{d^2})$ flops
  - $\Omega(n^2/\mathbf{d})$ words
  - $\Omega(\mathbf{d})$ msgs
- Now pick $\mathbf{d}$ (=latency cost)
  - $\mathbf{d} = \mathbf{\Omega}(\sqrt{\mathbf{p}})$ to minimize flops
  - $\mathbf{d} = \mathbf{\Omega}(\sqrt{\mathbf{c} \cdot \mathbf{p}})$ to minimize words
- More generally, latency $\cdot$ bandwidth $= n^2$



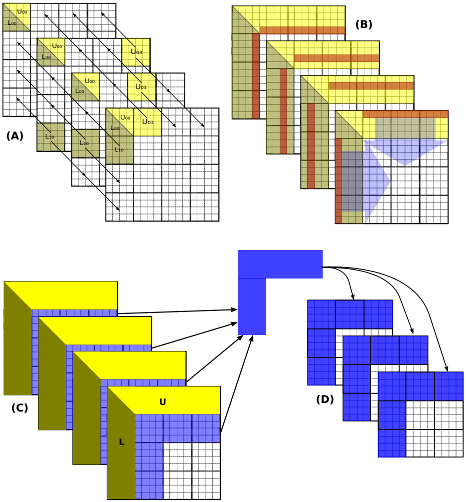
# 2.5D LU factorization



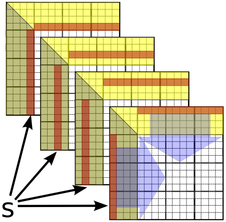
# 2.5D LU factorization



# 2.5D LU factorization



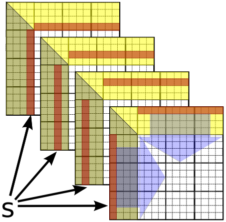
# 2.5D LU factorization



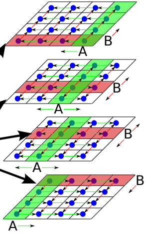
Look at how this
update is distributed.

What does it remind you of?

# 2.5D LU factorization

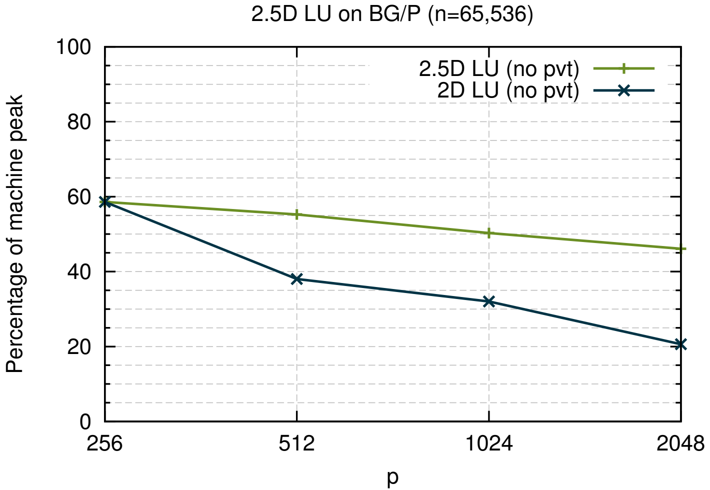


Look at how this
update is distributed.




Same 3D update
in multiplication

# 2.5D LU strong scaling (without pivoting)



# 2.5D LU with pivoting

$A = P \cdot L \cdot U$, where $P$ is a permutation matrix

- 2.5D generic pairwise elimination (neighbor/pairwise pivoting or Givens rotations (QR)) [A. Tiskin 2007]
  - pairwise pivoting does not produce an explicit $L$
  - pairwise pivoting may have stability issues for large matrices
- Our approach uses tournament pivoting, which is more stable than pairwise pivoting and gives $L$ explicitly
  - pass up rows of $A$ instead of $U$ to avoid error accumulation

# Tournament pivoting (CA-pivoting)

$\{\mathbf{P}, \mathbf{L}, \mathbf{U}\} \leftarrow$ **CA-pivot(A,n)**
**if** $n \leq b$ **then**
  **base case**
  $\{P, L, U\} = \text{partial-pivot}(A)$
**else**
  **recursive case**
  $[A_1^T, A_2^T] = A$
  $\{P_1, L_1, U_1\} = \text{CA-pivot}(A_1)$
  $[R_1^T, R_2^T] = P_1^T A_1$
  $\{P_2, L_2, U_2\} = \text{CA-pivot}(A_2)$
  $[S_1^T, S_2^T] = P_2^T A_2$
  $\{P_r, L, U\} = \text{partial-pivot}([R_1^T, S_1^T])$
  Form $P$ from $P_r$, $P_1$ and $P_2$
**end if**

# Tournament pivoting

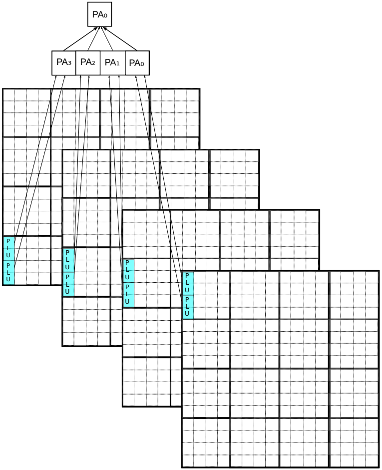
Partial pivoting is not communication-optimal on a blocked matrix

- requires message/synchronization for each column
- $O(n)$ messages needed

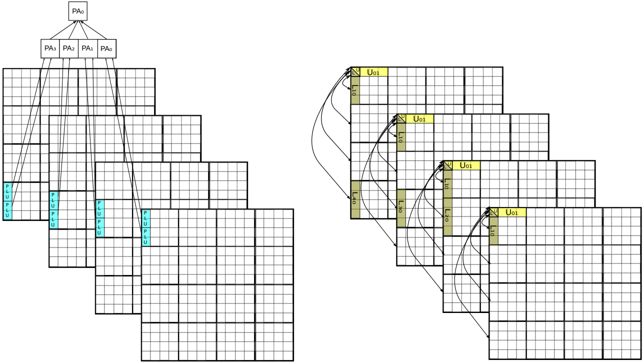
Tournament pivoting is communication-optimal

- performs a tournament to determine best pivot row candidates
- passes up 'best rows' of $A$

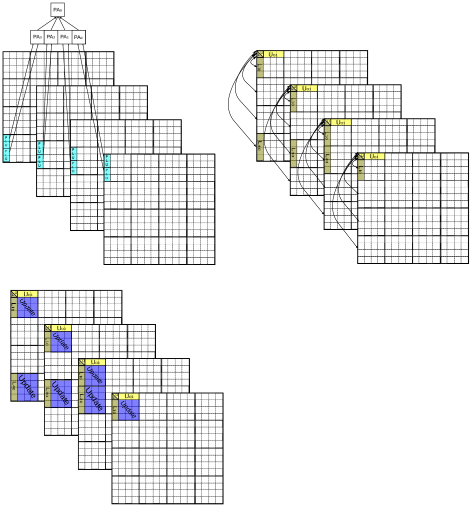
# 2.5D LU factorization with tournament pivoting



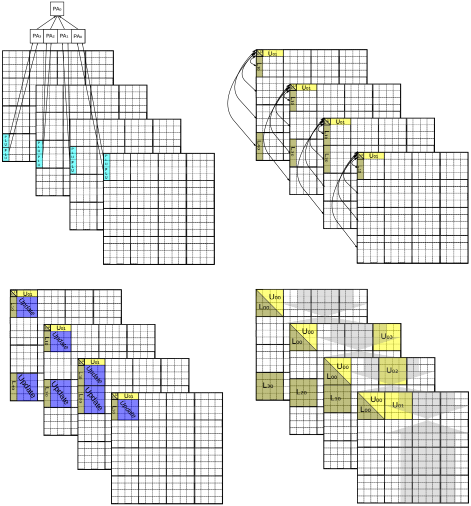
# 2.5D LU factorization with tournament pivoting



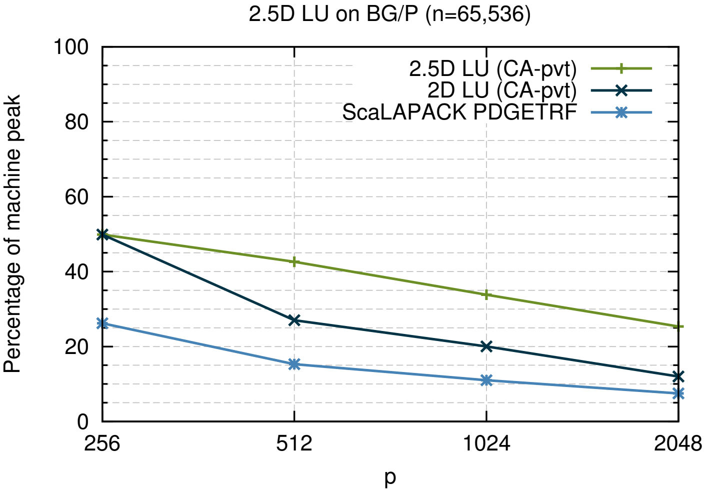
# 2.5D LU factorization with tournament pivoting



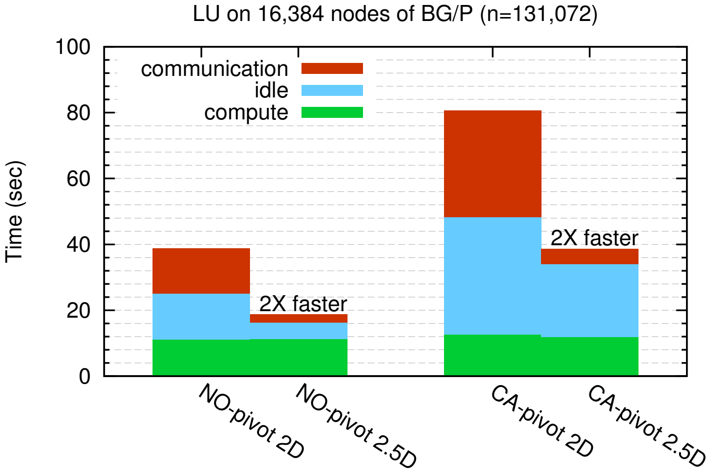
# 2.5D LU factorization with tournament pivoting



# Strong scaling of 2.5D LU with tournament pivoting



# 2.5D LU on 65,536 cores



# 2.5D QR factorization

$A = Q \cdot R$ where $Q$ is orthogonal $R$ is upper-triangular

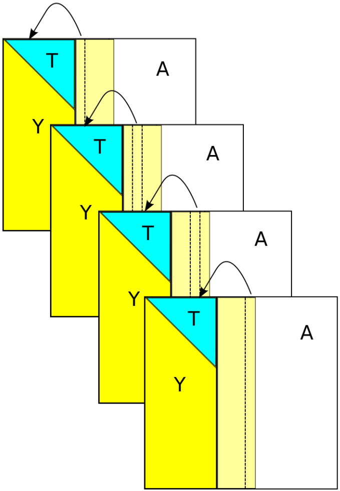
- 2.5D QR using Givens rotations (generic pairwise elimination) is given by [A. Tiskin 2007]
- Tiskin minimizes latency and bandwidth by working on slanted panels
- 2.5D QR cannot be done with right-looking updates as 2.5D LU due to non-commutativity of orthogonalization updates

# 2.5D QR factorization using the YT representation

The $YT$ representation of Householder QR factorization is more work efficient when computing only $R$

- We give an algorithm that performs 2.5D QR using the $YT$ representation
- The algorithm performs left-looking updates on $Y$
- Householder with $YT$ needs fewer computation (roughly 2x) than Givens rotations
- Our approach achieves optimal bandwidth cost, but has $O(n)$ latency

# 2.5D QR using YT representation



# Latency-optimal 2.5D QR

To reduce latency, we can employ the TSQR algorithm

1. Given $n$-by-$b$ panel partition into $2b$-by-$b$ blocks
2. Perform QR on each $2b$-by-$b$ block
3. Stack computed $R$s into $n/2$-by-$b$ panel and recursve
4. $Q$ given in hierarchical representation

Need $YT$ representation from hierarchical $Q$

- Take $Y$ to be the first $b$ columns of $Q$ minus the identity
- Define $T = (Y^T Y - I)^{-1}$
- Sacrifices triangular structure of $T$
- Conjecture: stable if $Q$ diagonal elements selected to be negative

# All-pairs shortest-paths

Given input graph $G = (V, E)$

- Find shortest paths between each pair of nodes $v_i$, $v_j$
- Reduces to semiring matrix multiplication with a dependency along $k$
- Computational structure is similar to LU factorization

Semiring matrix multiplication (SMM)

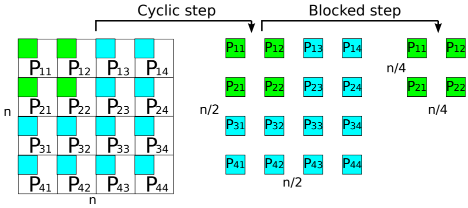
- Replace scalar multiply with scalar add
- Replace scalar add with scalar min
- Depending on processor can require more instructions

# A recursive algorithm for all-pairs shortest-paths

Known alogirhtm for recusively computing APSP:

1. Given adjacency matrix $A$ of graph $G$
2. Recursve on block $A_{11}$
3. Compute SMM $A_{12} \leftarrow A_{11} \cdot A_{12}$
4. Compute SMM $A_{21} \leftarrow A_{21} \cdot A_{11}$
5. Compute SMM $A_{22} \leftarrow A_{21} \cdot A_{12}$
6. Recursve on block $A_{22}$
7. Compute SMM $A_{21} \leftarrow A_{22} \cdot A_{21}$
8. Compute SMM $A_{12} \leftarrow A_{12} \cdot A_{22}$
9. Compute SMM $A_{11} \leftarrow A_{12} \cdot A_{21}$

# Block-cyclic recursive parallelization

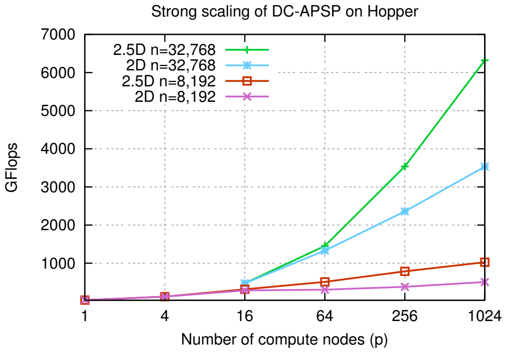


# 2.5D APSP

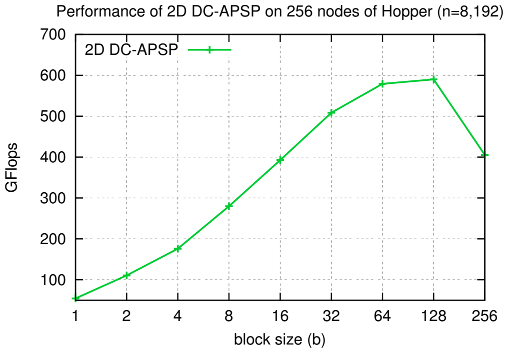
2.5D recursive parallelization is straight-forward

- Perform 'cyclic-steps' using a 2.5D process grid
- Decompose 'blocked-steps' using an octant of the grid
- Switch to 2D algorithm when grid is 2D
- Minimizes latency and bandwidth!

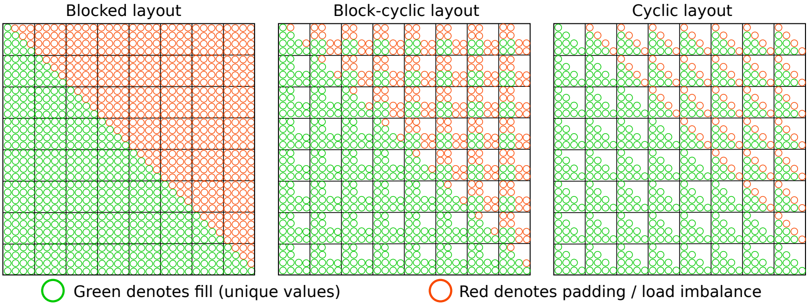
# 2.5D APSP strong scaling performance



# Block-size gives latency-bandwidth tradewoff



# Blocked vs block-cyclic vs cyclic decompositions

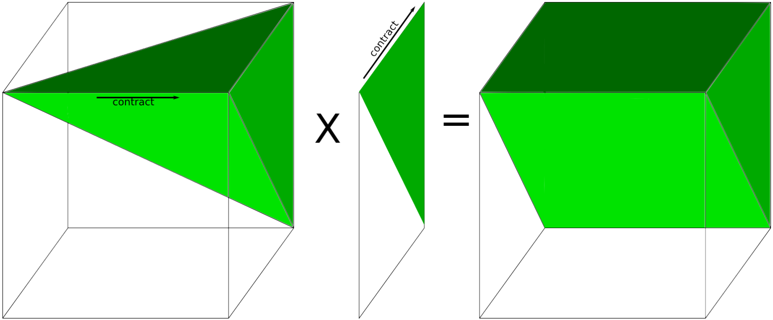


# Cyclops Tensor Framework (CTF)

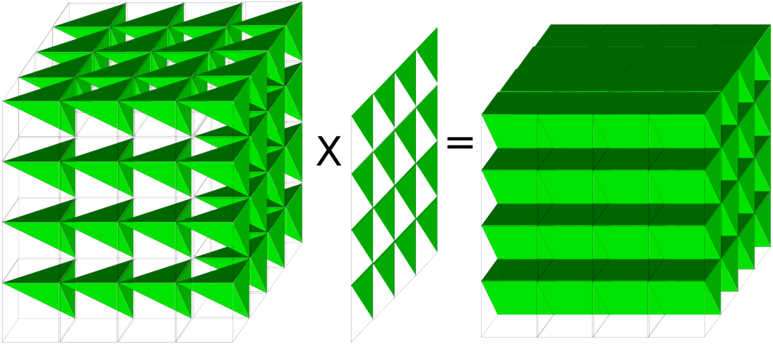
Big idea:

- decompose tensors cyclically among processors
- pick cyclic phase to preserve partial/full symmetric structure

# 3D tensor contraction

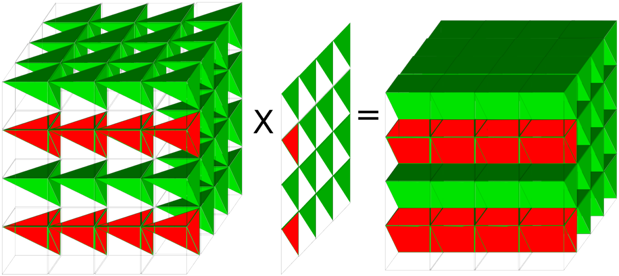


# 3D tensor cyclic decomposition



# 3D tensor mapping

Red portion denotes what processor (2,1) owns




| | | | |
|---|---|---|---|
| $P_{11}$ | $P_{12}$ | $P_{13}$ | $P_{14}$ |
| $P_{21}$ | $P_{22}$ | $P_{23}$ | $P_{24}$ |

# A cyclic layout is still challenging

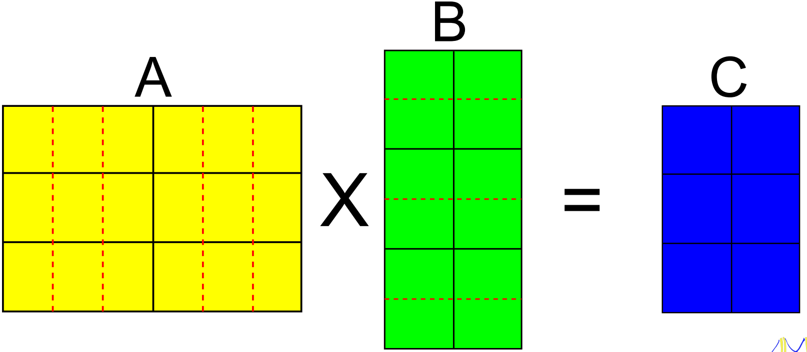
- In order to retain partial symmetry, all symmetric dimensions of a tensor must be mapped with the same cyclic phase
- The contracted dimensions of $A$ and $B$ must be mapped with the same phase
- And yet the virtual mapping, needs to be mapped to a physical topology, which can be any shape

# 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'

# Virtual processor grid construction

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

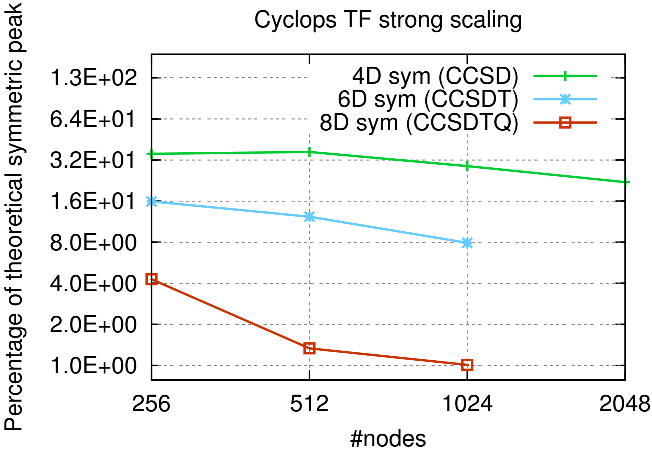


# 2.5D algorithms for tensors

We incorporate data replication for communication minimization into CTF

- Replicate only one tensor/matrix (minimize bandwidth but not latency)
- Autotune over mappings to all possible physical topologies
- Select mapping with least amount of communication
- Achieve minimal communication for tensors of widely different sizes

# Preliminary contraction results on Blue Gene/P



# Preliminary Coupled Cluster results on Blue Gene/Q

A Coupled Cluster with Double exictations (CCD) implementations is up and running

- Already scaled on up to 1024 nodes of BG/Q, up to 400 virtual orbitals
- Preliminary results already indicate performance matching NWChem
- Several major optimizations still in-progress
- Expecting significantly better scalability than any existing software

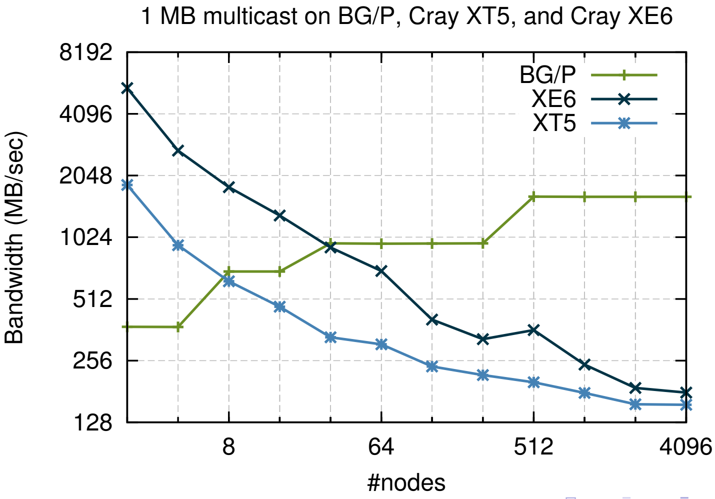
# Conclusion

Our contributions:

- 2.5D mapping of matrix multiplication
  - Optimal according to lower bounds [Irony, Tiskin, Toledo 04] and [Aggarwal, Chandra, and Snir 90]
- A new latency lower bound for LU
- Communication-optimal 2.5D LU, QR, and APSP
  - Both are bandwidth-optimal according to general lower bound [Ballard, Demmel, Holtz, Schwartz 10]
  - LU is latency-optimal according to new lower bound
- Cyclops Tensor Framework
  - Runtime autotuning to minimize communication
  - Topology-aware mapping in any dimension with symmetry considerations

# Backup slides

# 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