Scalable numerical algorithms for electronic structure calculations

Edgar Solomonik

UC Berkeley

July, 2012

Outline

Algorithmic roots: communication avoidance

Targeting leadership class platforms (e.g. BG/Q)

- Large amount of distributed memory parallelism
- Hierarchical parallelism
- Communication architecture lagging behind compute architecture

Architectures motivates communication-avoiding algorithms which consider

- bandwidth cost (amount of data communicated)
- latency cost (number of messages or synchronizations)
- critical path (communication load balance)
- topology (communication contention)

Application roots: electronic structure calculations

Electronic structure calculations seek approximate solution to the Schrodinger equation

$$i\hbar\frac{\partial}{\partial t}|\Psi\rangle = \hat{H}|\Psi\rangle$$

which satisfy

$$\hat{H}|\Psi\rangle = E|\Psi\rangle$$

often we want the ground-state (lowest-energy) wave-function Ψ_0, such that

$$\langle\Psi_0|\hat{H}|\Psi_0\rangle = E_0.$$

Density Function Theory (DFT)

DFT uses the fact that the ground-state wave-function Ψ_0 is a unique functional of the particle density $n(\vec{r})$

$$\Psi_0 = \Psi[n_0]$$

Since $\hat{H} = \hat{T} + \hat{V} + \hat{U}$, where $\hat{T}$, $\hat{V}$, and $\hat{U}$, are the kinetic, potential, and interaction contributions respectively,

$$E[n_0] = \langle \Psi[n_0] | \hat{T}[n_0] + \hat{V}[n_0] + \hat{U}[n_0] | \Psi[n_0] \rangle$$

DFT assumes $\hat{U} = 0$, and solves the Kohn-Sham equations

$$\left[-\frac{\hbar^2}{2m} \nabla^2 + V_s(\vec{r}) \right] \phi_i(\vec{r}) = \epsilon_i \phi_i(\vec{r})$$

where V_s has a exchange-correlation potential correction,

$$V_s(\vec{r}) = V(\vec{r}) + \int \frac{e^2 n_s(\vec{r'})}{|\vec{r} - \vec{r'}|} d^3 r' + V_{XC}[n_s(\vec{r})]$$

Density Function Theory (DFT), contd.

The exchange-correlation potential V_{XC} is approximated by DFT, by a functional which is often system-dependent. This allows the following iterative scheme

1. Given an (initial guess) $n(\vec{r})$ calculate V_s via Hartree-Fock and functional
2. Solve (diagonalize) the Kohn-Sham equation to obtain each ϕ_i
3. Compute a new guess at $n(\vec{r})$ based on ϕ_i

Due to the rough approximation of correlation and exchange DFT is good for weakly-correlated systems (which appear in solid-state physics), but suboptimal for strongly-correlated systems.

Linear algebra in DFT

DFT requires a few core numerical linear algebra kernels

- Matrix multiplication (of rectangular matrices)
- Linear equations solver
- Symmetric eigensolver (diagonalization)

We proceed to study schemes for optimization of these algorithms.

Matrix multiplication (MM)

We consider matrix multiplication

$$C[i,j] = \sum_{k=0}^{n-1} A[i,k]B[k,j]$$

which in algorithmic form is

---

for $i = 0$ to $n-1$ **do**
  for $j = 0$ to $n-1$ **do**
    for $k = 0$ to $n-1$ **do**
      $C[i,j] += A[i,k] \cdot B[k,j]$

SUMMA/Cannon blocked algorithm for MM

On a l-by-l processor grid, with $b = n/l$

---

for $k = 0$ to $n-1$ **do**
  for $p = 0$ to $l-1$ **in parallel do**
    for $q = 0$ to $l-1$ **in parallel do**
      broadcast $A[:, k]$
      broadcast $B[k, :]$
      for $i = 0$ to $b-1$ **do**
        for $j = 0$ to $b-1$ **do**
          $C[i+pb, j+qb] += A[i+pb, k] \cdot B[k, j+qb]$

SUMMA/Cannon cyclic algorithm for MM

Replace

$$C[i+pb, j+qb] += A[i+pb, k] \cdot B[k, j+qb]$$

with

$$C[il+p, jl+q] += A[il+p, k] \cdot B[k, jl+q]$$

---

for $k = 0$ to $n-1$ **do**
  for $p = 0$ to $l-1$ **in parallel do**
    for $q = 0$ to $l-1$ **in parallel do**
      broadcast $A[:, k]$
      broadcast $B[k, :]$
      for $i = 0$ to $b-1$ **do**
        for $j = 0$ to $b-1$ **do**
          $C[il+p, jl+q] += A[il+p, k] \cdot B[k, jl+q]$

SUMMA/Cannon costs

With blocking, on p processors these algorithms move A and B $l = \sqrt{p}$ times. Each process own n^2/p of the matrices, so the bandwidth cost is

$$W = O(n^2/\sqrt{p})$$

and the number of synchronizations necessary is

$$S = O(\sqrt{p}).$$

For rectangular matrices with dimensions n, m, k, we select an algorithm on process grid l_1-by-l_2 that communicates A and B, or A and C, or B and C, for a cost of

$$W = O(\min_{l_1, l_2}(nml_1 + mkl_2, nml_1 + nkl_2, mkl_1 + nkl_2)/p)$$

3D matrix multiplication

On a l-by-l-by-l processor grid, with $b = n/l$

---

broadcast A
broadcast B
for $r = 0$ to $l-1$ **in parallel do**
  for $p = 0$ to $l-1$ **in parallel do**
    for $q = 0$ to $l-1$ **in parallel do**
      for $k = 0$ to $b-1$ **do**
        for $i = 0$ to $b-1$ **do**
          for $j = 0$ to $b-1$ **do**
            $C[i+pb, j+qb] += A[i+pb, k+rb] \cdot B[k+rb, j+qb]$
reduce C

3D matrix multiplication with block notation

On a l-by-l-by-l processor grid, with $b = n/l$

---

broadcast A
broadcast B
for $r = 0$ to $l - 1$ **in parallel do**
  for $p = 0$ to $l - 1$ **in parallel do**
    for $q = 0$ to $l - 1$ **in parallel do**
      $C[p, q] += A[p, r] \cdot B[r, q]$
reduce C

3D MM costs

On p processors these algorithms move A and B and C once. Each process own $n^2/p^{2/3}$ of the matrices, so the bandwidth cost is

$$W = O(n^2/p^{2/3})$$

and the number of synchronizations necessary is

$$S = O(1).$$

However, the algorithm requires storage (memory usage) of

$$M = \Omega(n^2/p^{2/3})$$

which is $p^{1/3}$ more than minimal.

2.5D matrix multiplication

On a l-by-l-by-c processor grid, with $b = n/l$

---

for $s = 0$ to $l/c - 1$ **do**
  broadcast A
  broadcast B
  for $r = 0$ to $l - 1$ **in parallel do**
    for $p = 0$ to $l - 1$ **in parallel do**
      for $q = 0$ to $l - 1$ **in parallel do**
        $C[p, q] += A[p, sr] \cdot B[sr, q]$
reduce C

2.5D MM costs

On p processors these algorithms move A and B, $\sqrt{p/c^3}$ times and C once. Each process own $\frac{n^2}{\sqrt{cp}}$ of the matrices, so the bandwidth cost is

$$W = O\left(\frac{n^2}{\sqrt{cp}}\right)$$

and the number of synchronizations necessary is

$$S = O\left(\sqrt{p/c^3}\right).$$

while the memory is now

$$M = \Omega(cn^2/p)$$

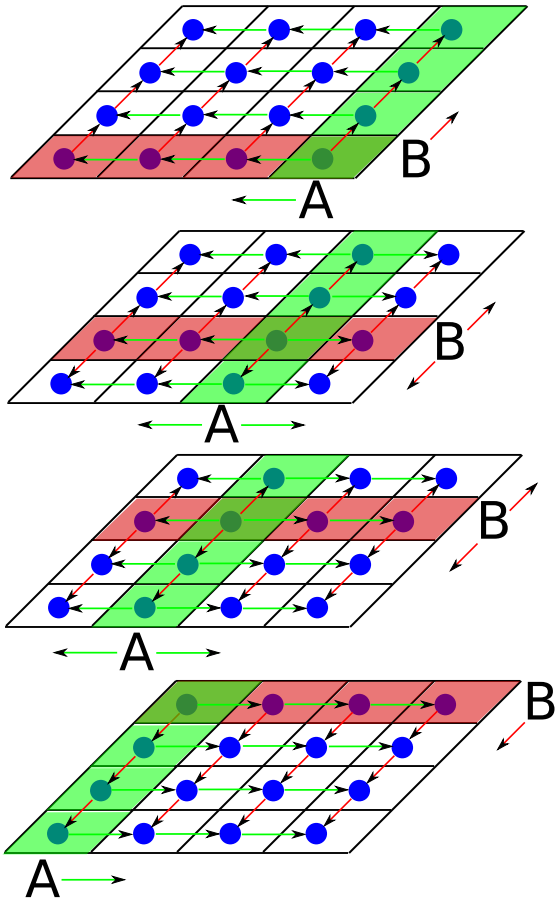
which is tunable given the architectural constraint, allowing better asymptotic strong scaling.

3D rectangular matrix multiplication

On a l_1-by-l_2-by-l_3 processor grid, with $b = n/l_1 = m/l_2 = k/l_3$

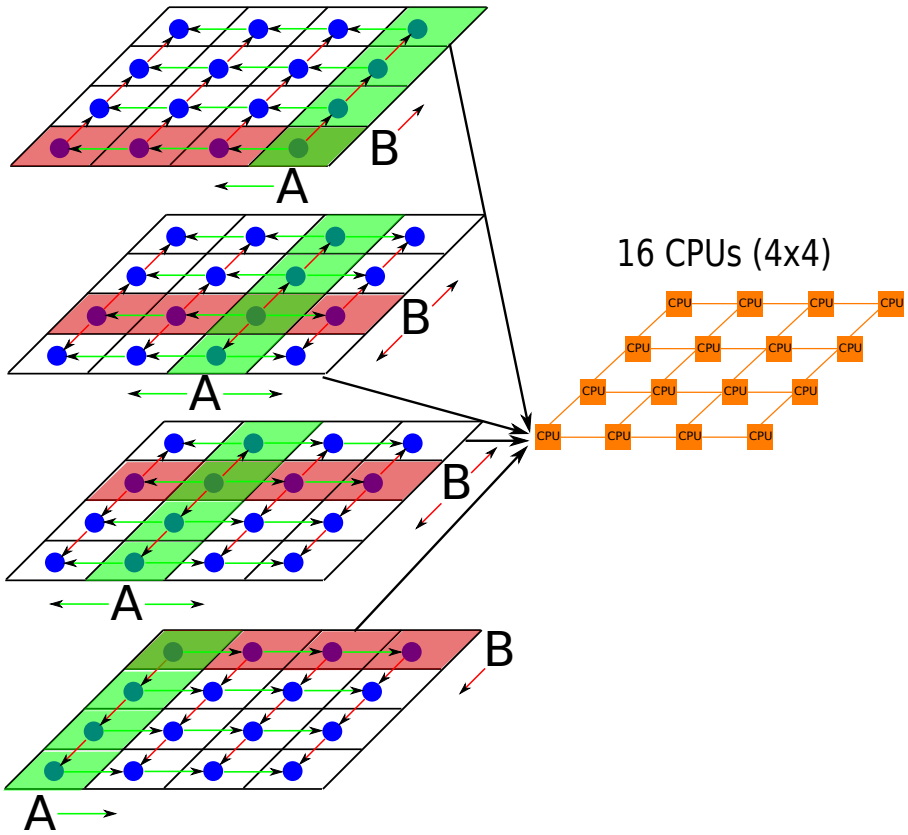
broadcast A
broadcast B
for $r = 0$ to $l_3 - 1$ **in parallel do**
  for $p = 0$ to $l_1 - 1$ **in parallel do**
    for $q = 0$ to $l_2 - 1$ **in parallel do**
      $C[p, q] += A[p, r] \cdot B[r, q]$
reduce C

Blocking matrix multiplication



2D matrix multiplication

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



$O(n^3/p)$ flops

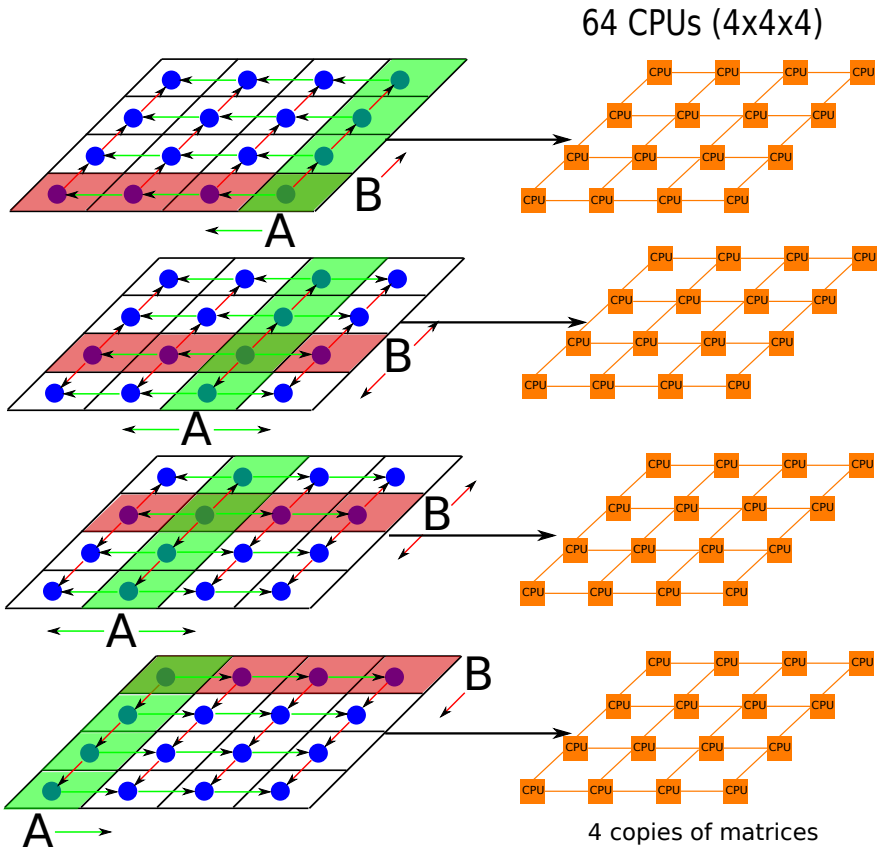
$O(n^2/\sqrt{p})$ words moved

$O(\sqrt{p})$ 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

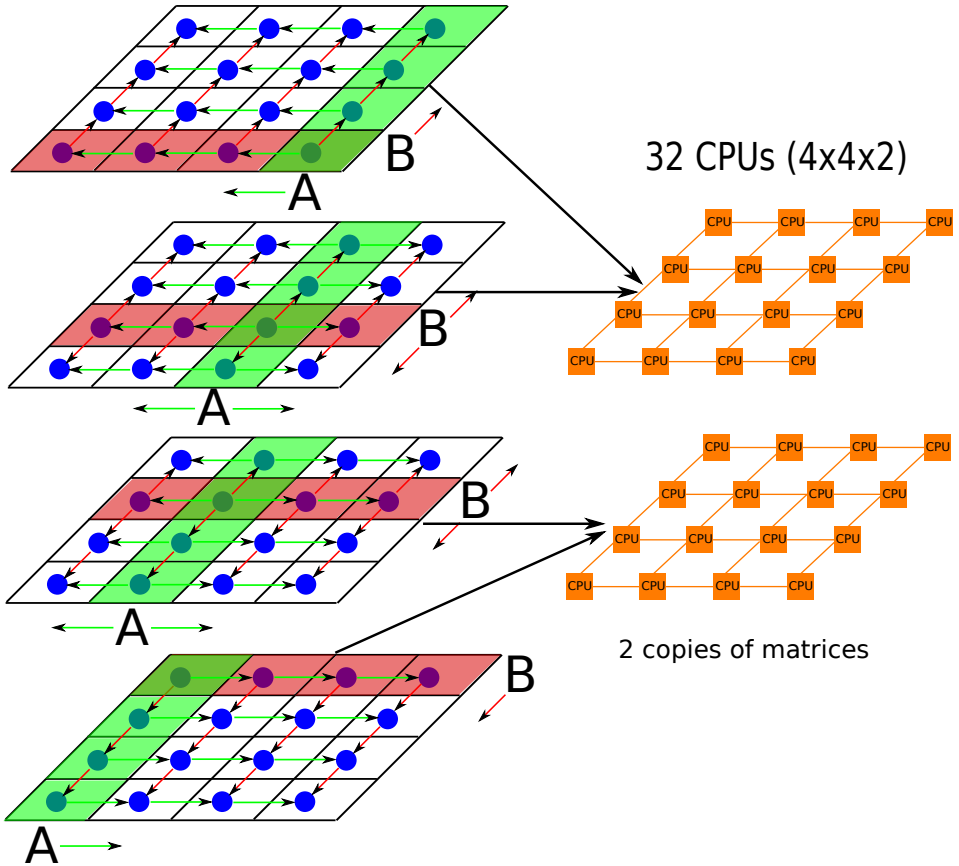
$O(n^2/p^{2/3})$ words moved

$O(1)$ messages

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

2.5D matrix multiplication

[McColl and Tiskin 99]



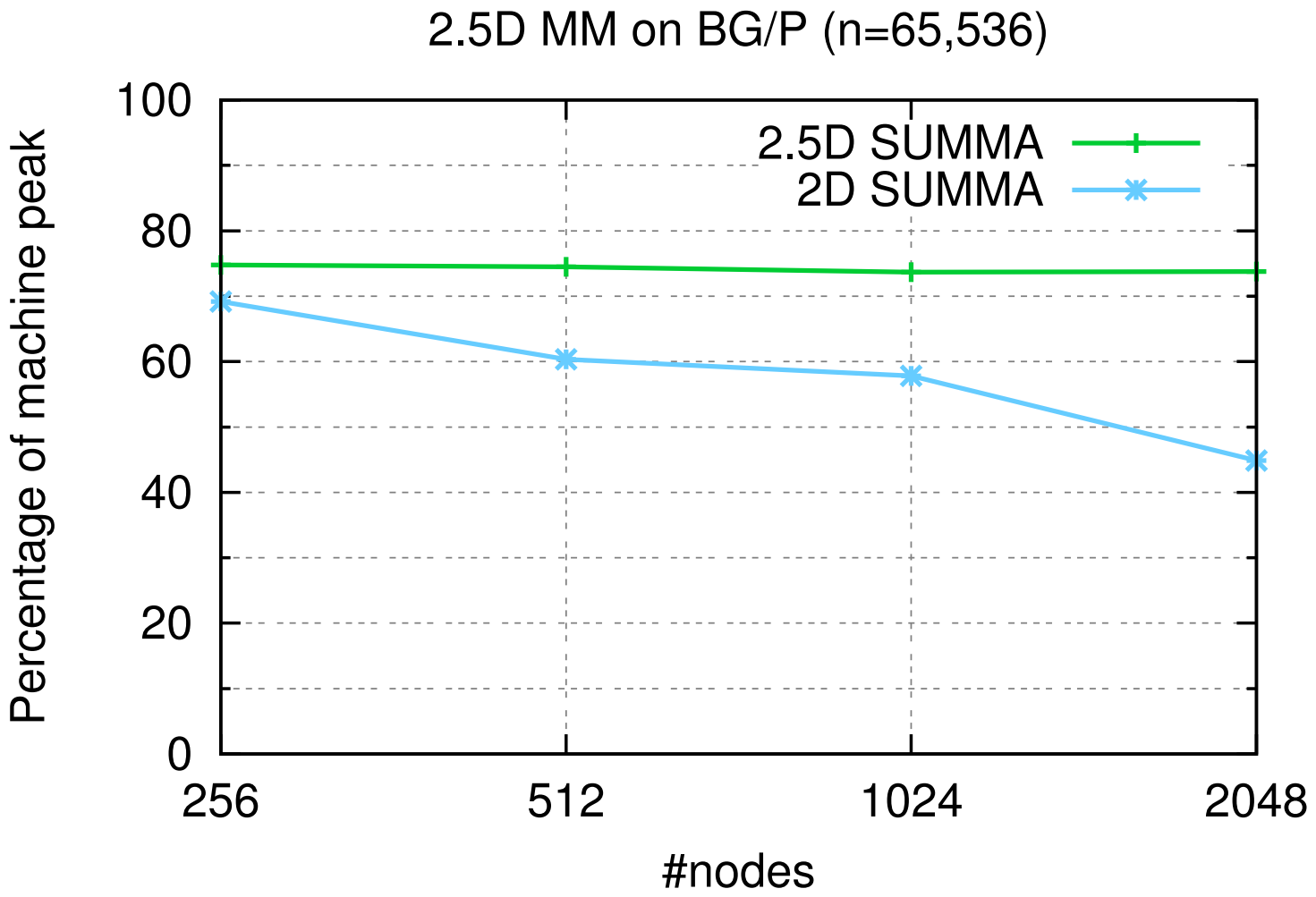
$O(n^3/p)$ flops

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

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

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

2.5D MM on 65,536 cores



Solutions to linear systems of equations

We want to solve some matrix equation

$$A \cdot X = B$$

where A and B are known. Can solve by factorizing $A = LU$ (L lower triangular and U upper triangular) via Gaussian elimination, then computing TRSMs

$$X = U^{-1}L^{-1}B$$

via triangular solves. If A is symmetric positive definite, we can use Cholesky factorization. Cholesky and TRSM are no harder than LU.

Non-pivoted LU factorization

---

for $k = 0$ to $n-1$ **do**
  $U[k, k:n-1] = A[k, k:n-1]$
  for $i = k+1$ to $n-1$ **do**
    $L[i,k] = A[i,k]/U[k,k]$
    for $j = k+1$ to $n-1$ **do**
      $A[i,j] -= L[i,k] \cdot U[k,j]$

This algorithm has a dependency that requires

$$k \leq i, k \leq j.$$

Non-pivoted 2D LU factorization

On a l-by-l process grid

Algorithm 1 $[L, U] = \text{2D-LU}(A)$

for $k = 0$ to $n-1$ **do**
  Factorize $A[k,k] = L[k,k] \cdot U[k,k]$
  Broadcast $L[k,k]$ and $U[k,k]$
  for $p = 0$ to $l-1$ **in parallel do**
    solve $L[p,k] = A[p,k]U[k,k]^{-1}$
  for $q = 0$ to $l-1$ **in parallel do**
    solve $U[k,q] = L[k,k]^{-1}A[1,k]$
  Broadcast $L[p,k]$ and $U[k,q]$
  for $p = 0$ to $l-1$ **in parallel do**
    for $q = 0$ to $l-1$ **in parallel do**
      $A[p,q] -= L[p,k] \cdot U[k,q]$

3D recursive non-pivoted LU and Cholesky

A 3D 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 have a new lower-bound for the latency cost

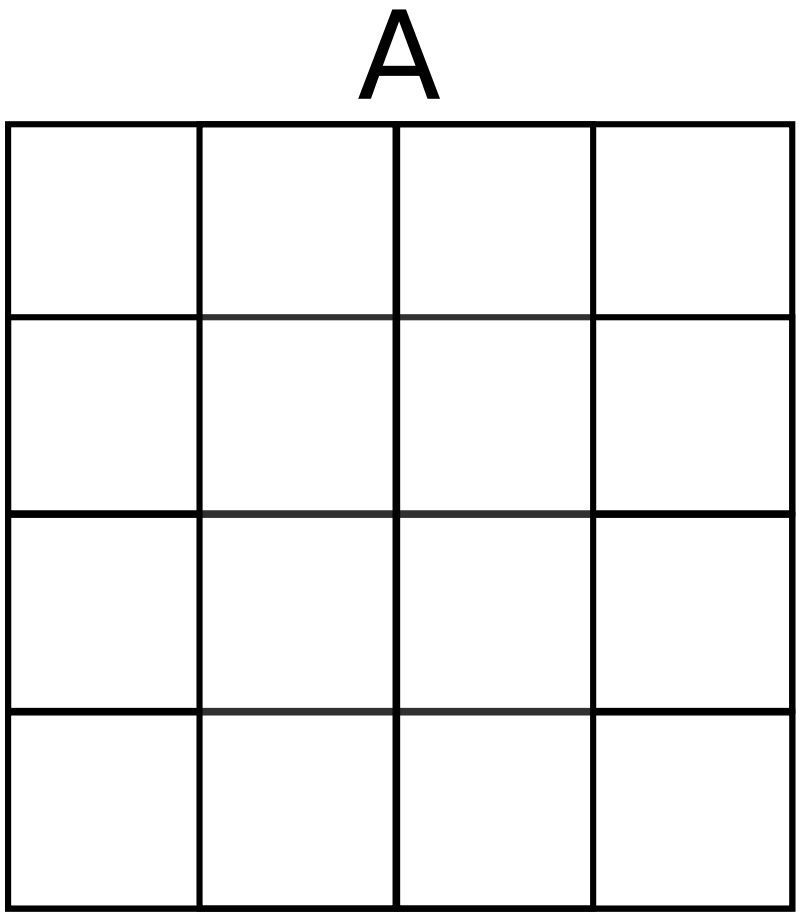
3D non-pivoted LU and Cholesky

On a l-by-l-by-l process grid

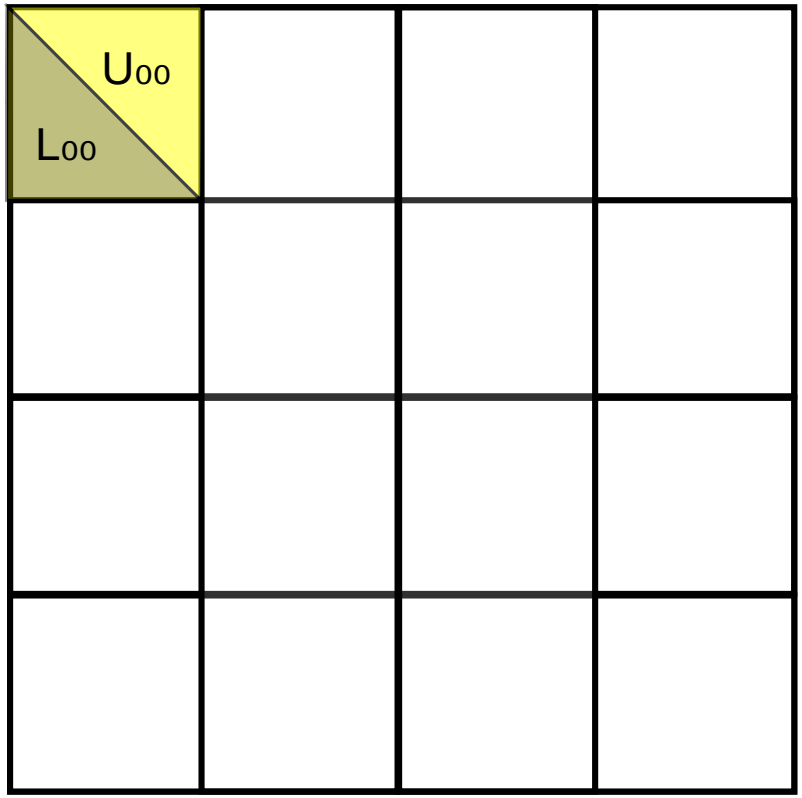
---

for $r = 0$ to $l-1$ **do**
  $[L[r,r], U[r,r]] = \text{2D-LU}(A[r,r])$
  Broadcast $L[k,k]$ and $U[k,k]$
  $[L[r+1:l-1, r]] = \text{2D-TRSM}(A[r+1:l-1, r], U[r,r])$;
  $[U[r, r+1:l-1]] = \text{2D-TRSM}(A[r, r+1:l-1], L[r,r])$;
  for $s = 0$ to $l-1$ **in parallel do**
    Broadcast $L[p, rs]$ and $U[rs, q]$
    for $p = 0$ to $l-1$ **in parallel do**
      for $q = 0$ to $l-1$ **in parallel do**
        $A[p,q] -= L[p,rs] \cdot U[rs,q]$

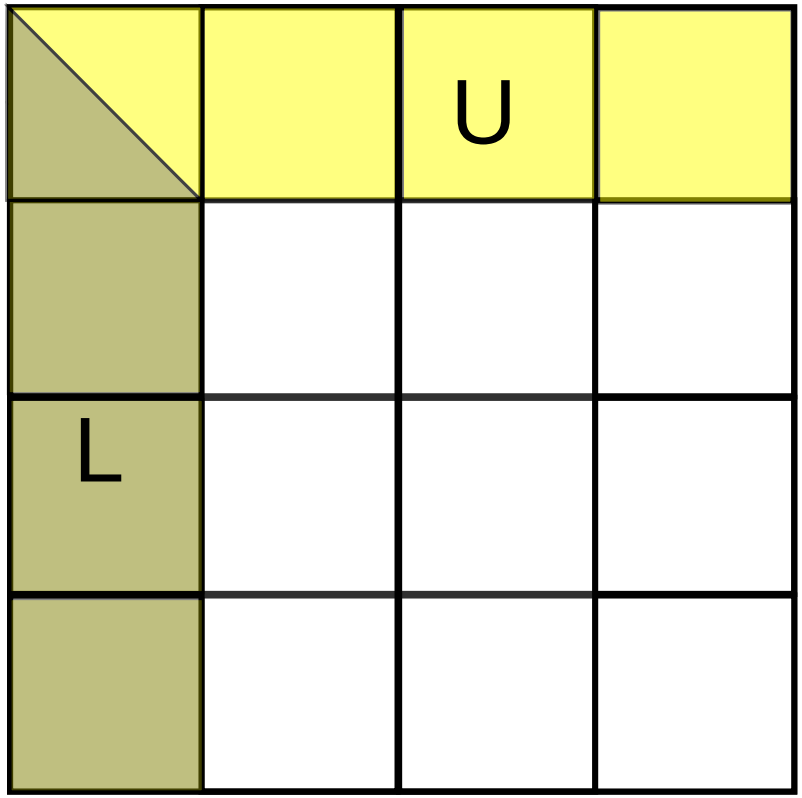
2D blocked LU factorization



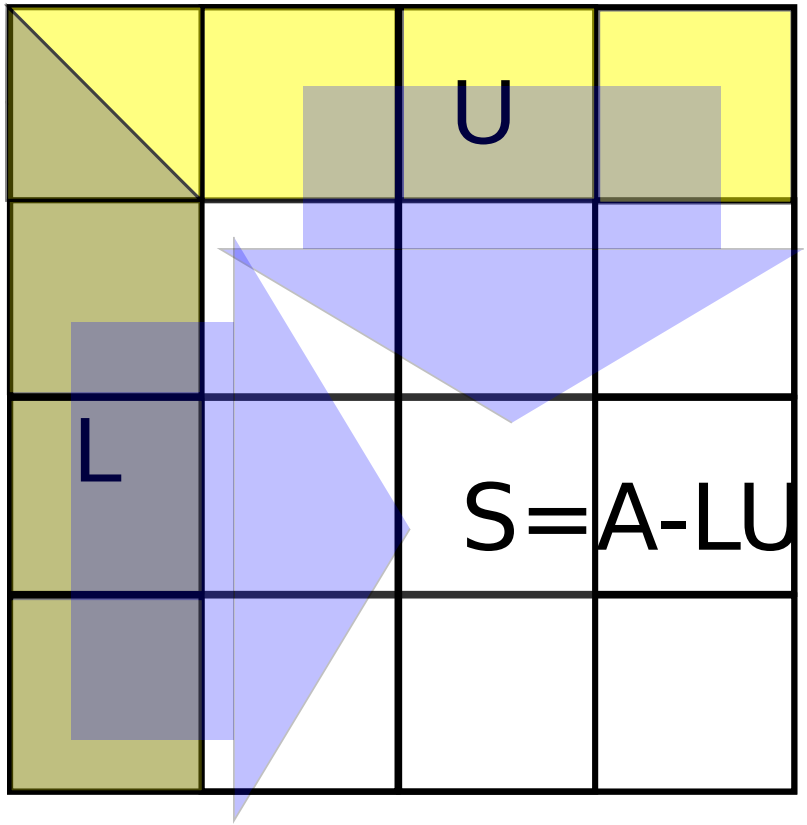
2D blocked LU factorization



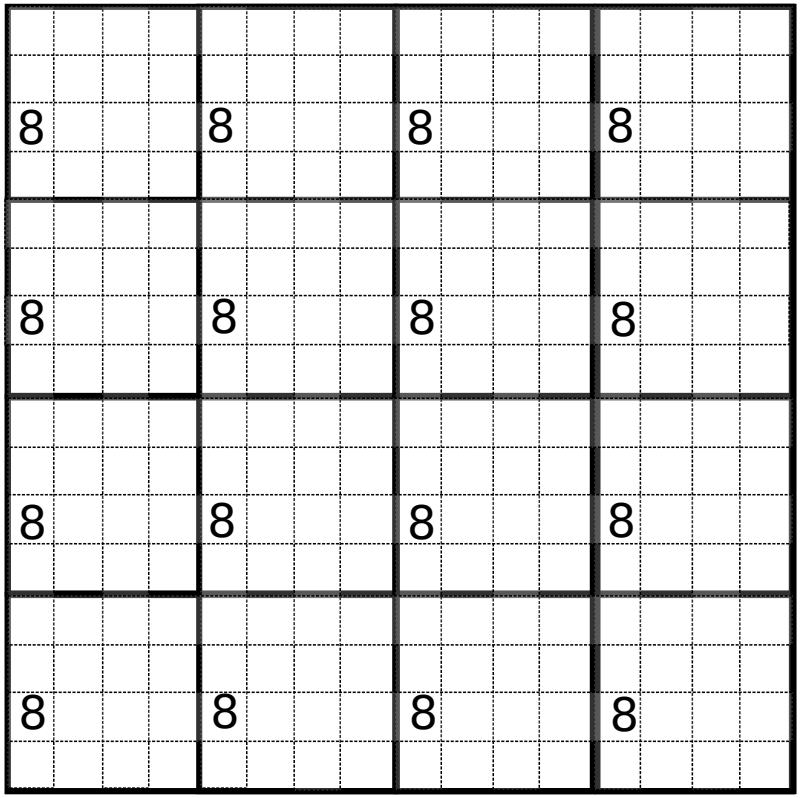
2D blocked LU factorization



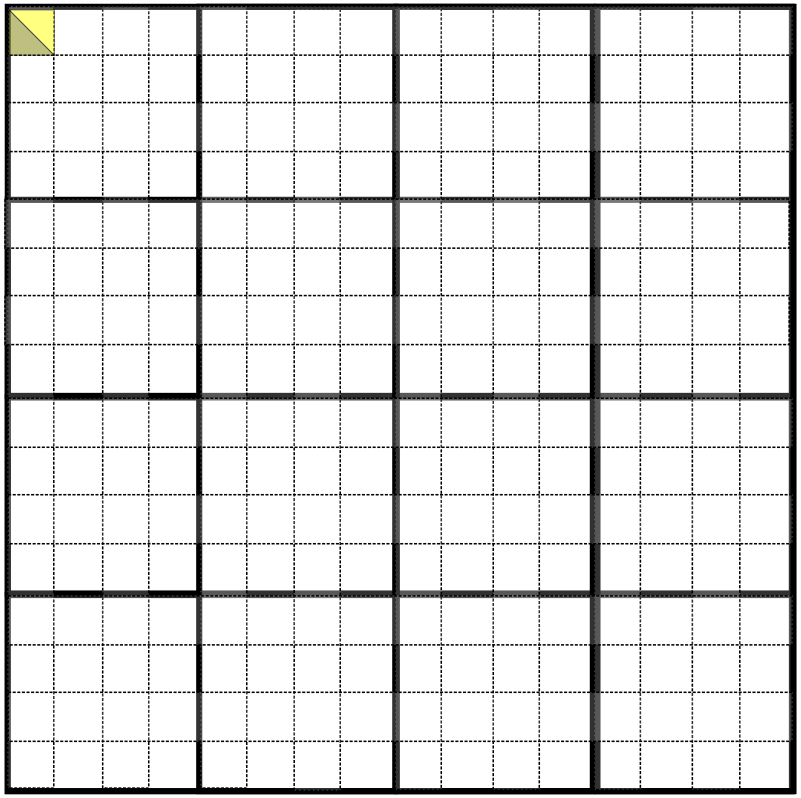
2D blocked LU factorization



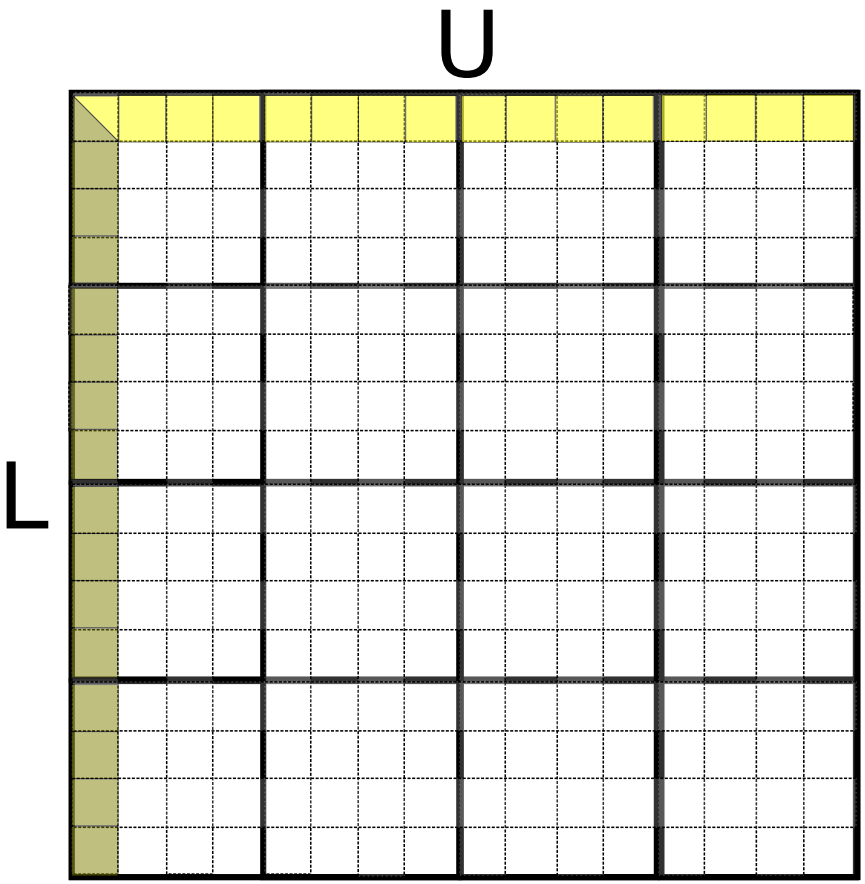
2D block-cyclic decomposition



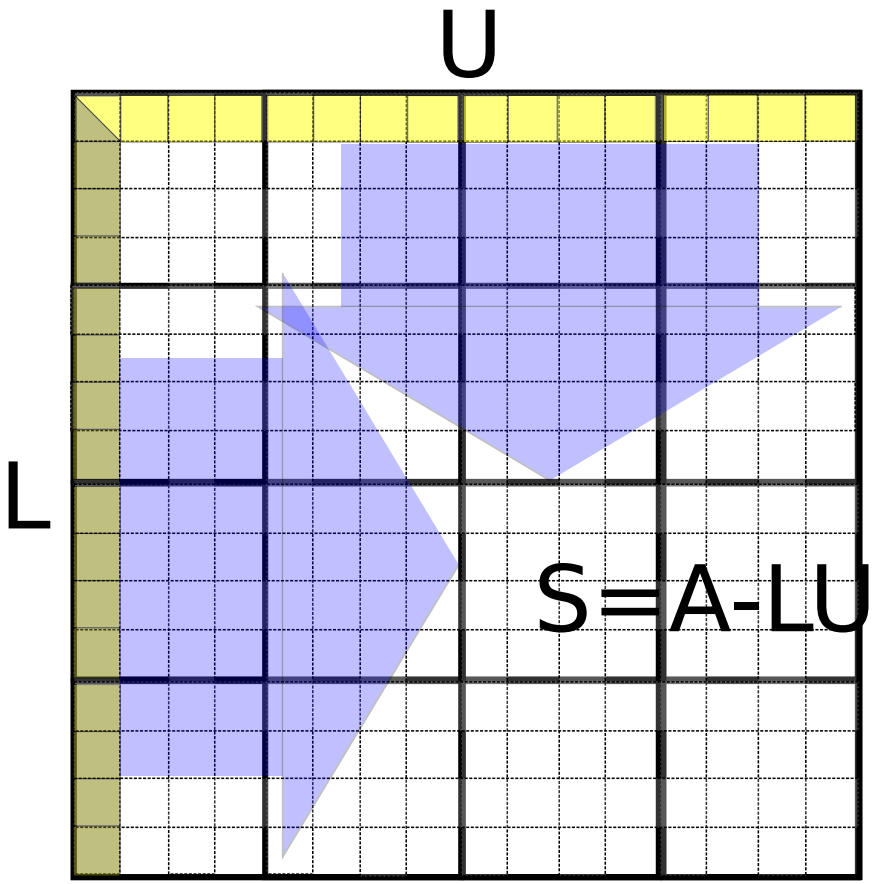
2D block-cyclic LU factorization



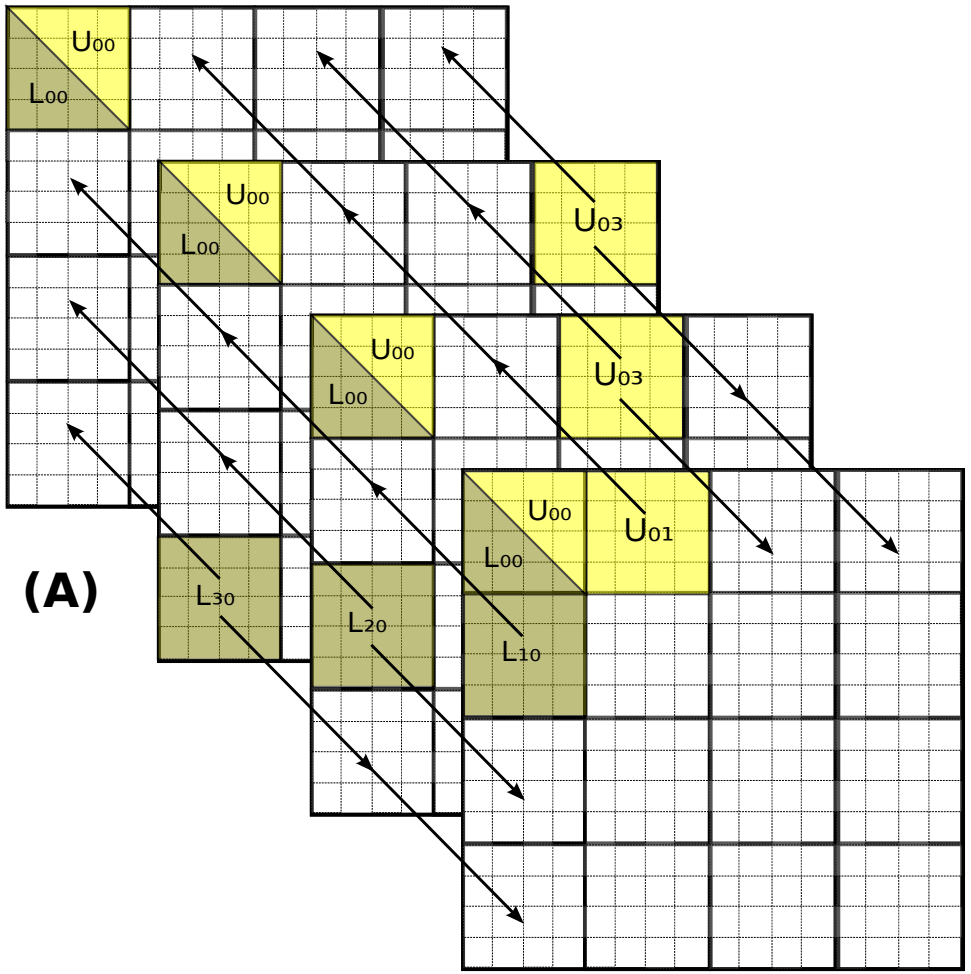
2D block-cyclic LU factorization



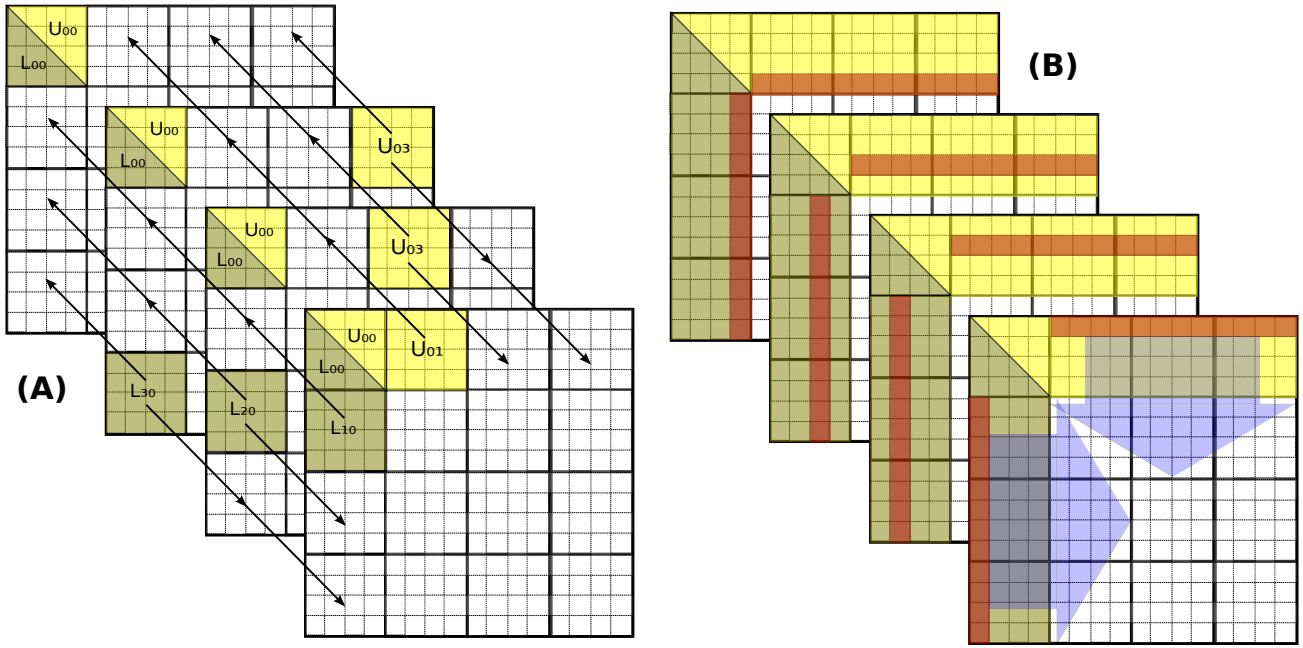
2D block-cyclic LU factorization



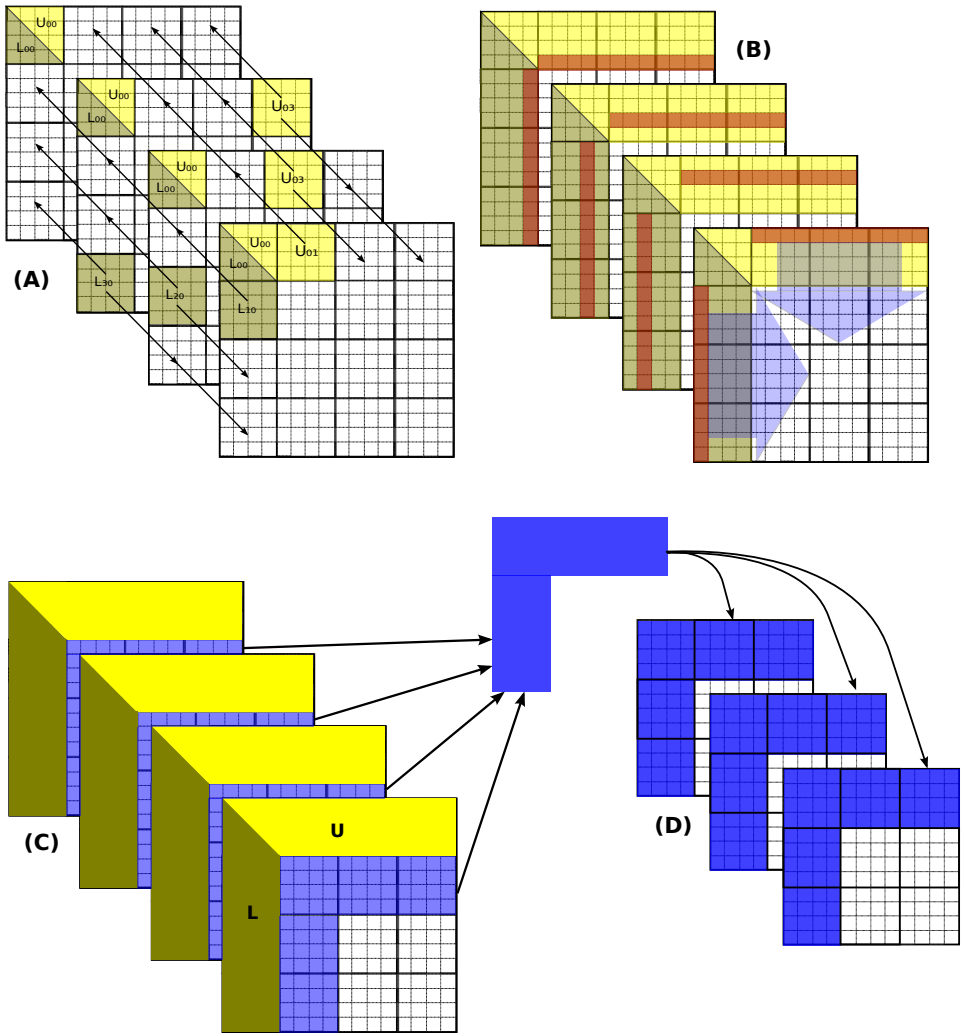
2.5D LU factorization



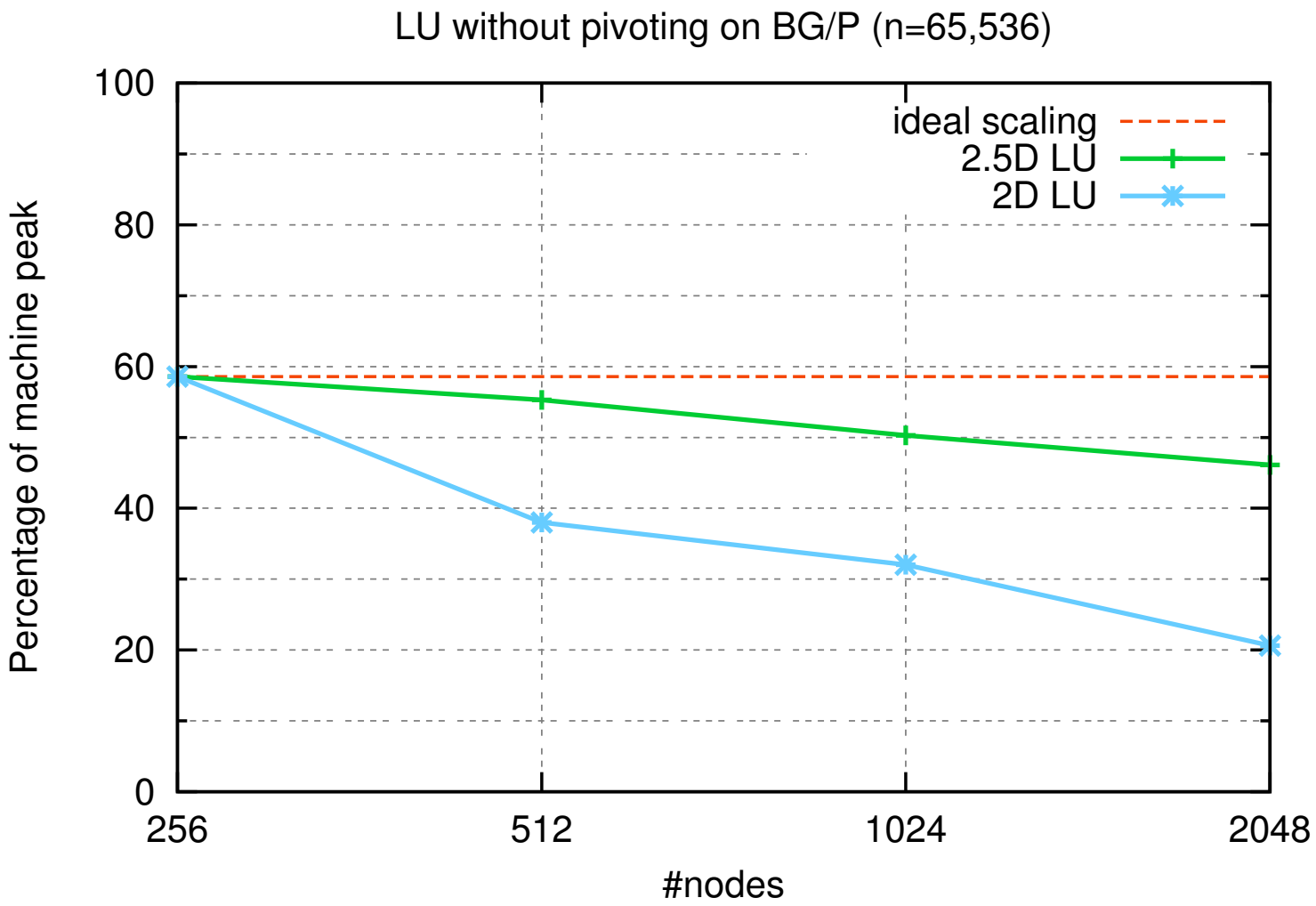
2.5D LU factorization



2.5D LU factorization



2.5D LU 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

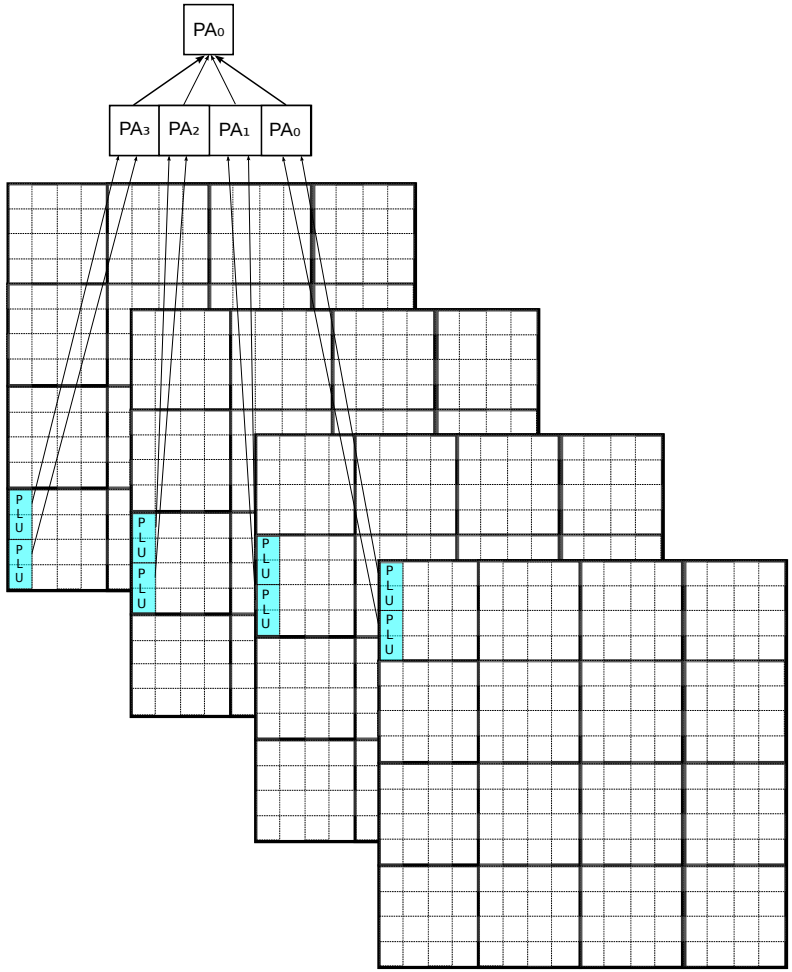
Partial pivoting is not communication-optimal on a blocked matrix

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

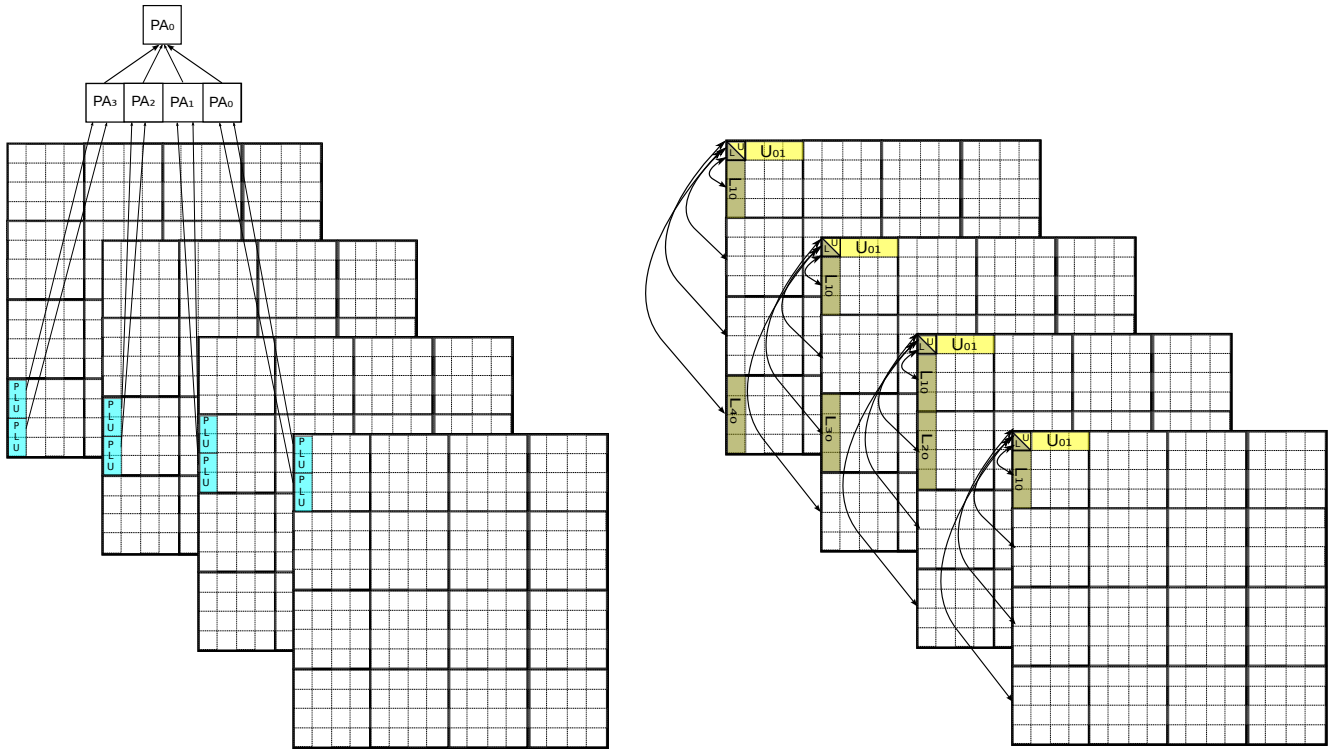
Tournament pivoting is communication-optimal

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

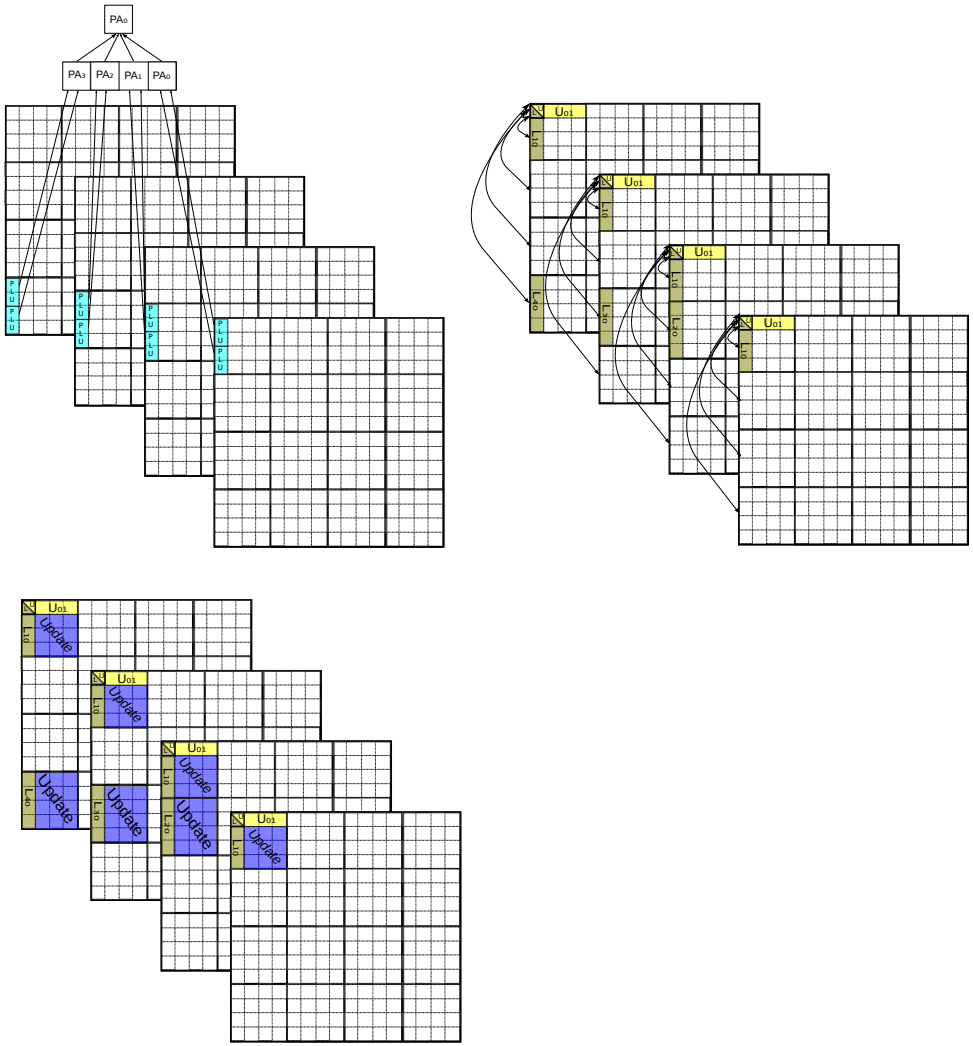
2.5D LU factorization with tournament pivoting



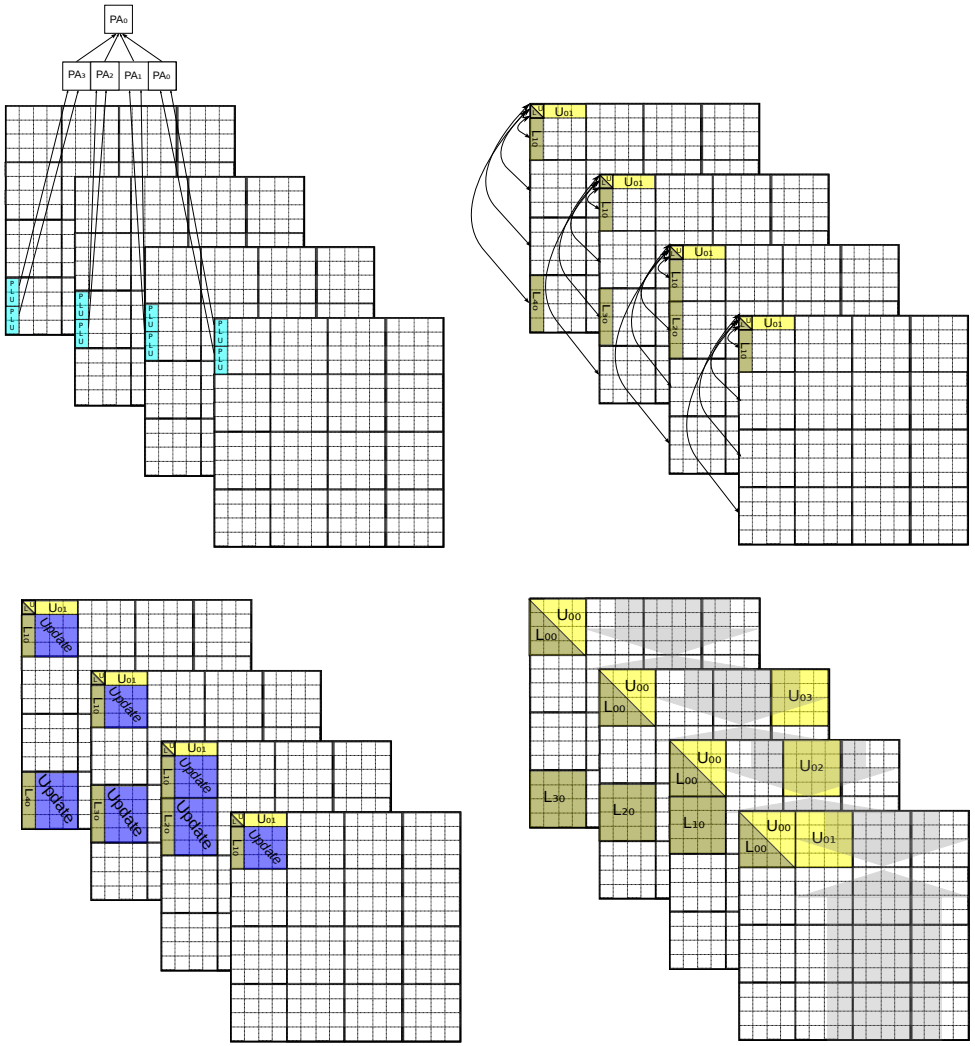
2.5D LU factorization with tournament pivoting



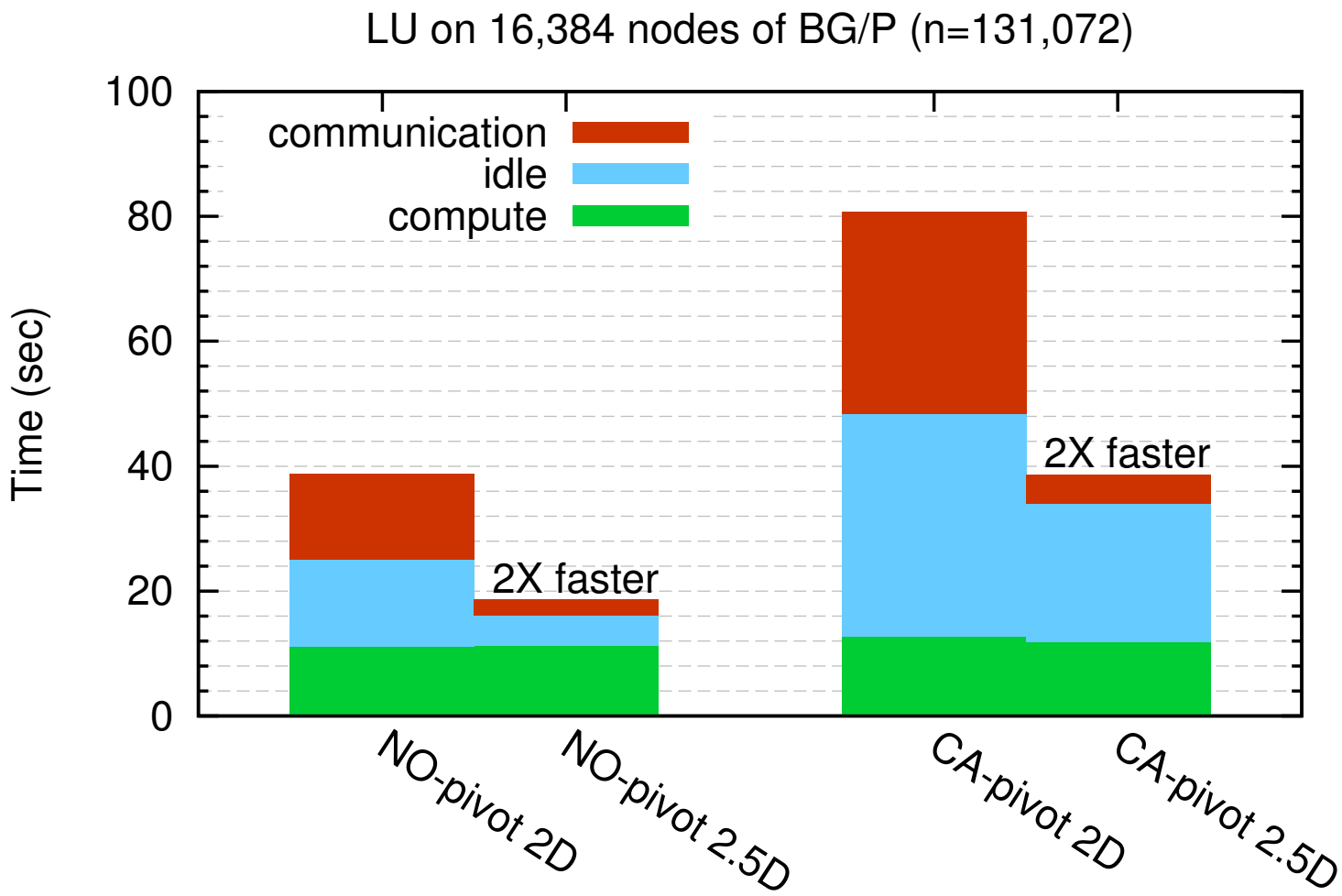
2.5D LU factorization with tournament pivoting



2.5D LU factorization with tournament pivoting



2.5D LU on 65,536 cores



Symmetric eigensolve via QR

To solve the symmetric eigenproblem on matrix A, we need to diagonalize

$$A = UDU^T$$

where U are the singular vectors and D is the singular values. This can be done by a series of two-sided orthogonal transformations

$$A = U_1 U_2 \ldots U_k D U_k^T \ldots U_2^T U_1^T$$

The process may be reduced to three stages: a QR factorization reducing to banded form, a reduction from banded to tridiagonal, and a tridiagonal eigensolve. We consider the QR, which is the most expensive step.

3D QR factorization

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

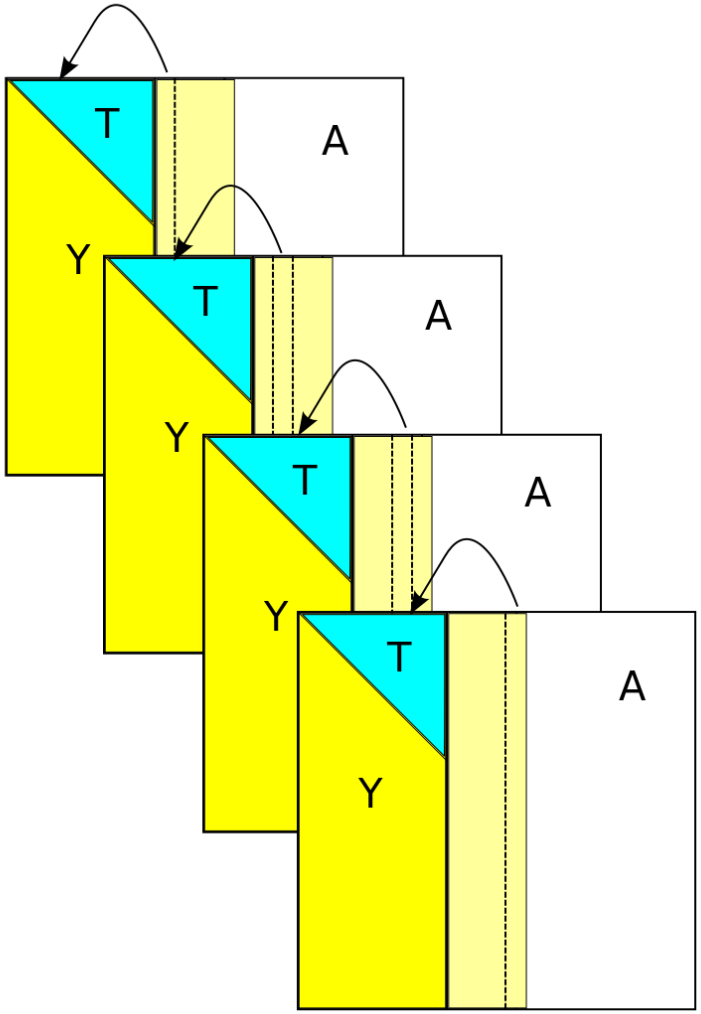
- 3D QR using Givens rotations (generic pairwise elimination) is given by [A. Tiskin 2007]
- Tiskin minimizes latency and bandwidth by working on slanted panels
- 3D QR cannot be done with right-looking updates as 2.5D LU due to non-commutativity of orthogonalization updates

3D 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

3D 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 Rs into $n/2$-by-b panel and recursive
4. Q given in hierarchical representation

Need YT representation from hierarchical Q...

YT reconstruction

Yamamoto et al.

- Take Y to be the first b columns of Q minus the identity
- Define $T = (I - Q_1)^{-1}$
- Sacrifices triangular structure of T and Y.

Our first attempt

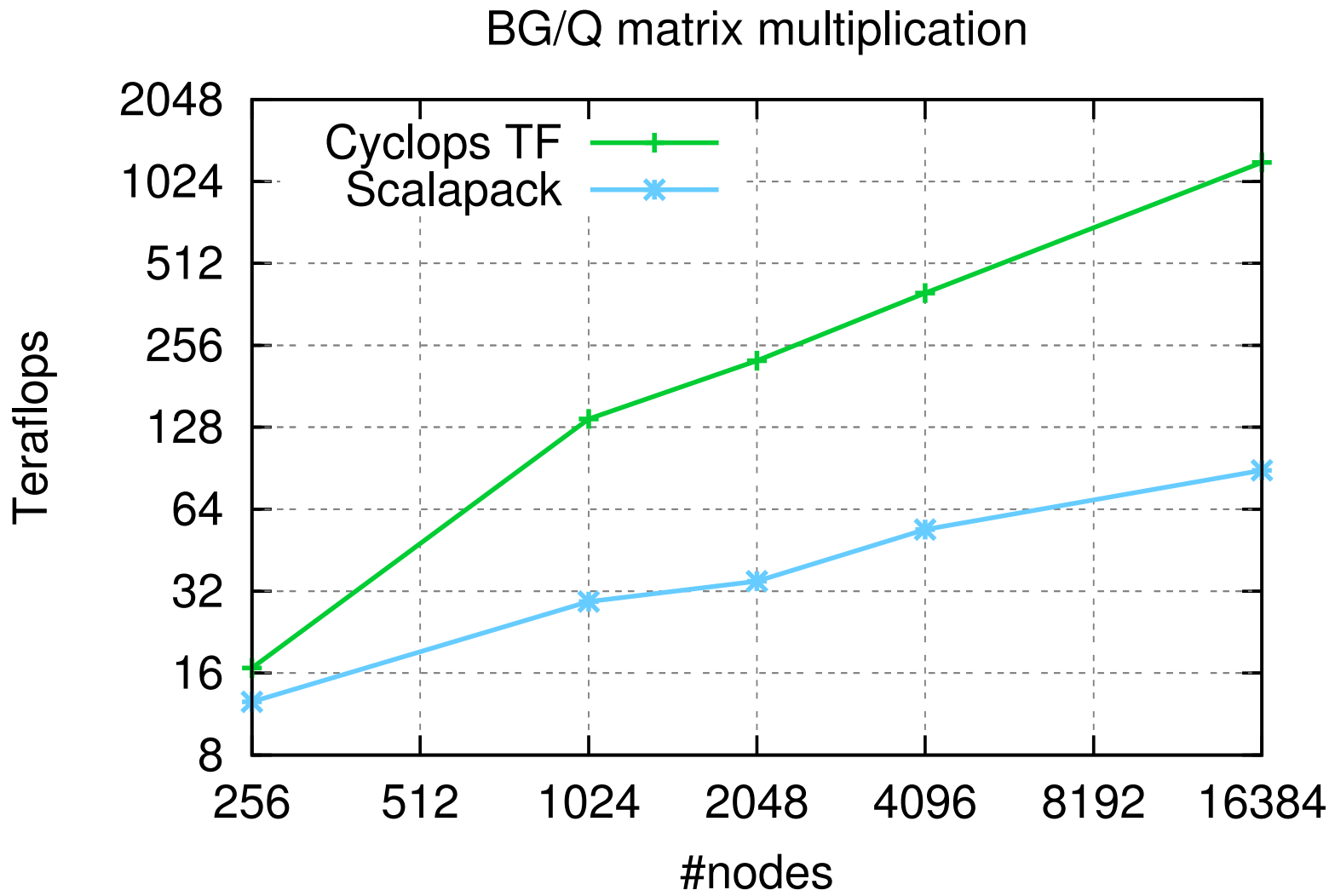
$$LU(R-A) = LU(R-(I-YTY^T)R) = LU(YTY^TR) = (Y)\cdot(TY^TR)$$

was unstable due to being dependent on the condition number of R. However, performing LU on Yamamoto's T seems to be stable,

$$LU(I-Q_1) = LU(I-(I-Y_1TY_1^T)) = LU(Y_1TY_1^T) = (Y_1)\cdot(TY_1^T)$$

and should yield triangular Y and T.

3D algorithms on BG/Q



3D algorithms for DFT

3D matrix multiplication is integrated into QBox.

- QBox is a DFT code developed by Erik Draeger et al.
- Depending on system/functional can spend as much as 80% time in MM
- Running on most of Sequoia and getting significant speed up from 3D
- 1.75X speed-up on 8192 nodes 1792 gold atoms, 31 electrons/atom
- Eventually hope to build and integrate a 3D eigensolver into QBox

Coupled Cluster (CC)

Coupled Cluster is a method for electronic structure calculations of strongly-correlated systems. CC rewrites the wave-function $|\Psi\rangle$ as an excitation operator $\hat{T}$ applied to the Slater determinant $|\Psi_0\rangle$

$$|\Psi\rangle = e^{\hat{T}}|\Psi_0\rangle$$

where $\hat{T}$ is as a sum of $\hat{T}_n$ (the n'th excitation operators)

$$\hat{T}_{\text{CCSD}} = \hat{T}_1 + \hat{T}_2$$
$$\hat{T}_{\text{CCSDT}} = \hat{T}_1 + \hat{T}_2 + \hat{T}_3$$
$$\hat{T}_{\text{CCSDTQ}} = \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \hat{T}_4$$

Coupled Cluster (CC)

The CC amplitudes $\hat{T}_n$ can be solved via the coupled equations

$$\langle \Psi_{ij\ldots}^{ab\ldots} | e^{-\hat{T}} H e^{\hat{T}} | \Psi_0 \rangle$$

where we expand out the excitation operator

$$e^{\hat{T}} = 1 + \hat{T} + \frac{\hat{T}^2}{2!} \ldots$$

By Wick's theorem only fully contracted terms of the expansion will be non-zero, and diagrammatic or algebraic derivations yield many terms such as

$$\sum_{klcd} \langle kl||cd\rangle \, T_k^c \, T_l^a \, T^{db} \, T_{ij}$$

which can be factorized into two-term contractions.

A parallel algorithm for any dimensional tensor

Dense tensor contractions can be reduced to matrix multiplication

- tensors must be transposed (indices must be reordered)
- parallelized via 2D/3D algorithms

Alternatively, we can keep tensors in high-dimensional layout and perform recursive SUMMA

- replicate along any dimension for 3D
- SUMMA along each dimension where indices are mismatched.

4D tensor contraction

On a l-by-l-by-l-by-l processor grid, with $b = n/l$

---

for $p = 0$ to $n-1$ **do**
  for $q = 0$ to $n-1$ **do**
    for $r = 0$ to $l-1$ **in parallel do**
      for $s = 0$ to $l-1$ **in parallel do**
        broadcast $A[p, :, :, :]$
        broadcast $B[:, p, :, :]$
        for $t = 0$ to $l-1$ **in parallel do**
          for $u = 0$ to $l-1$ **in parallel do**
            broadcast $A[:, :, q, :]$
            broadcast $B[:, :, :, q]$
            $C[r, s, t, i] += A[p, s, q, i] \cdot B[r, p, t, q]$

Tensor symmetry

Most physical tensors of interest have symmetric indices

$$W^{ab} = W^{ba}$$

or skew-symmetric indices

$$W^{ab} = -W^{ba}.$$

Multi-index symmetries and partial index symmetries also arise, e.g.

$$W^{ab}_{ijkl}$$

where (a, b) are permutationally symmetric and (i, j, k, l) and permutationally symmetric. Symmetry is a vital computational consideration, since it can save computation and much memory scaling as $d!$ where d is the number of symmetric indices.

Symmetric contractions

Consider the contraction

$$C_{ij}^{ab} = \sum_c A_{ij}^{ac} \cdot B^{cb},$$

if A and C both have i, j symmetry (symmetry preserved), compute

$$C_{i<j}^{ab} = \sum_c A_{i<j}^{ac} \cdot B^{cb}$$

if B is symmetric in (c, b) (broken symmetry), compute

$$C_{ij}^{ab} = \sum_c A_{ij}^{ac} \cdot B^{c<b} + A_{ij}^{ac} \cdot B^{b\leq c}$$

if C is skew-symmetric in (a, b) (broken symmetry), symmetrize

$$C_{ij}^{a<b} = \sum_c A_{ij}^{ac} \cdot B^{cb} - A_{ij}^{bc} \cdot B^{ca}$$

Cyclops Tensor Framework (CTF)

Big idea:

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

Interface:

$$C[\text{"}abij\text{"}] += A[\text{"}acij\text{"}] \cdot B[\text{"}cb\text{"}]$$

with symmetries pre-specified for A, B, and C.

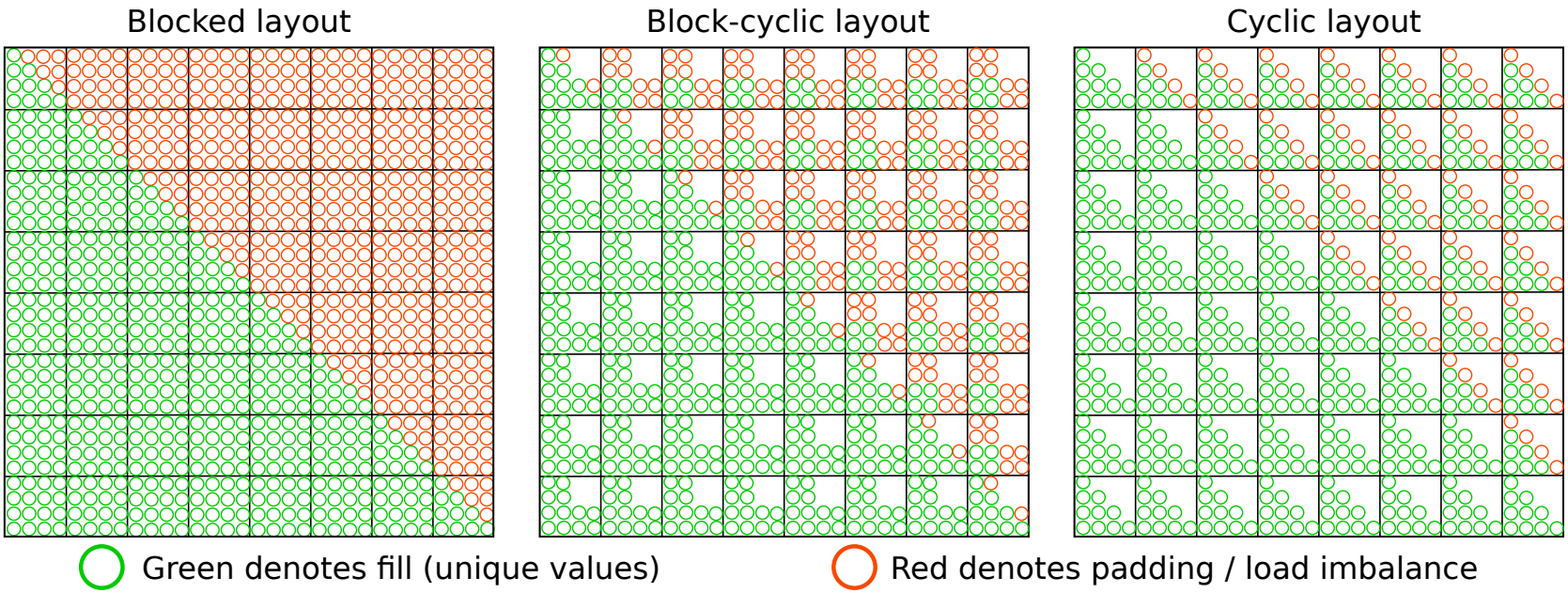
NWChem blocked approach

for $k = 0$ to $n-1$ **do**
  for $p = 0$ to $l-1$ **in parallel do**
    for $q = 0$ to $p-1$ **in parallel do**
      broadcast $A[:, k]$
      broadcast $B[k, :]$
      for $i = 0$ to $b-1$ **do**
        for $j = 0$ to $b-1$ **do**
          $C[i+pb, j+qb] += A[i+pb, k] \cdot B[k, j+qb]$

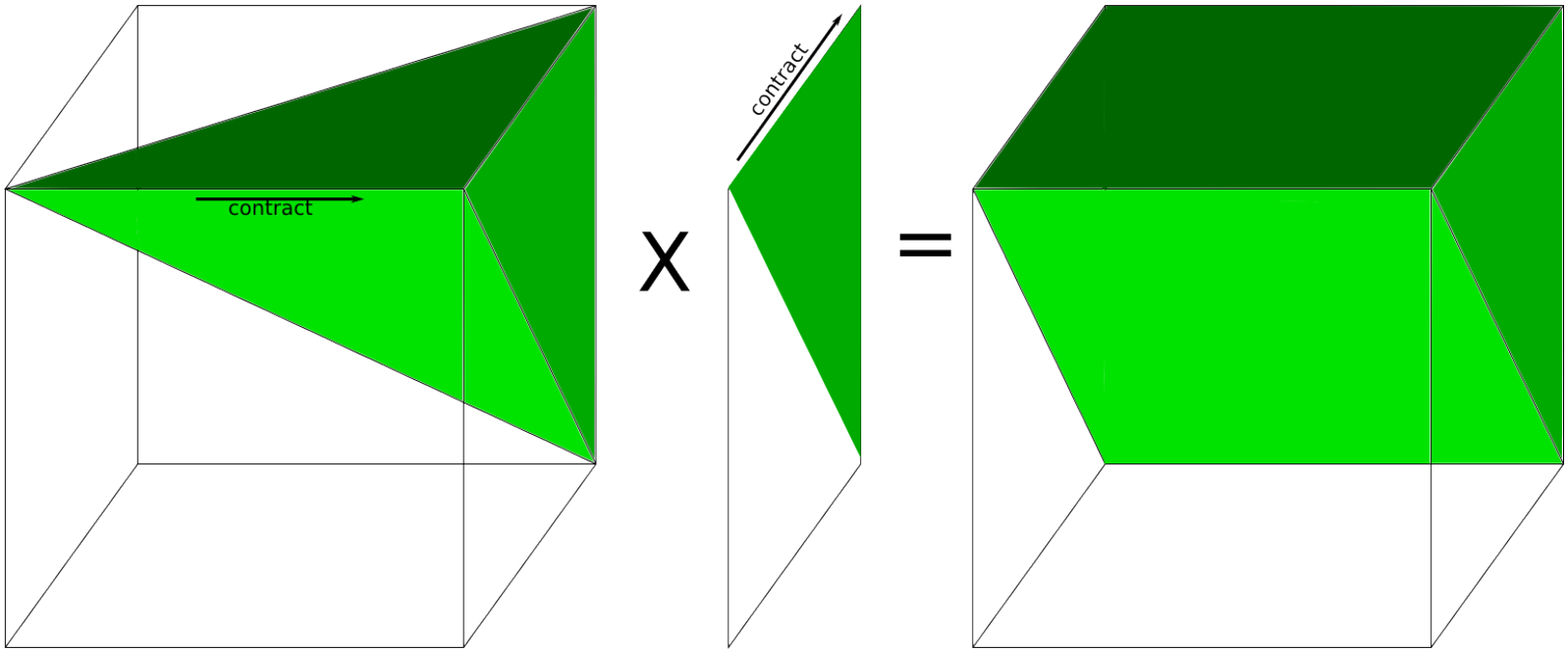
Cyclops TF cyclic approach

for $k = 0$ to $n-1$ **do**
  for $p = 0$ to $l-1$ **in parallel do**
    for $q = 0$ to $l-1$ **in parallel do**
      broadcast $A[:,k]$
      broadcast $B[k,:]$
      for $i = 0$ to $b-1$ **do**
        for $j = 0$ to $i-1$ **do**
          $C[il+p, jl+q] += A[il+p, k] \cdot B[k, jl+q]$

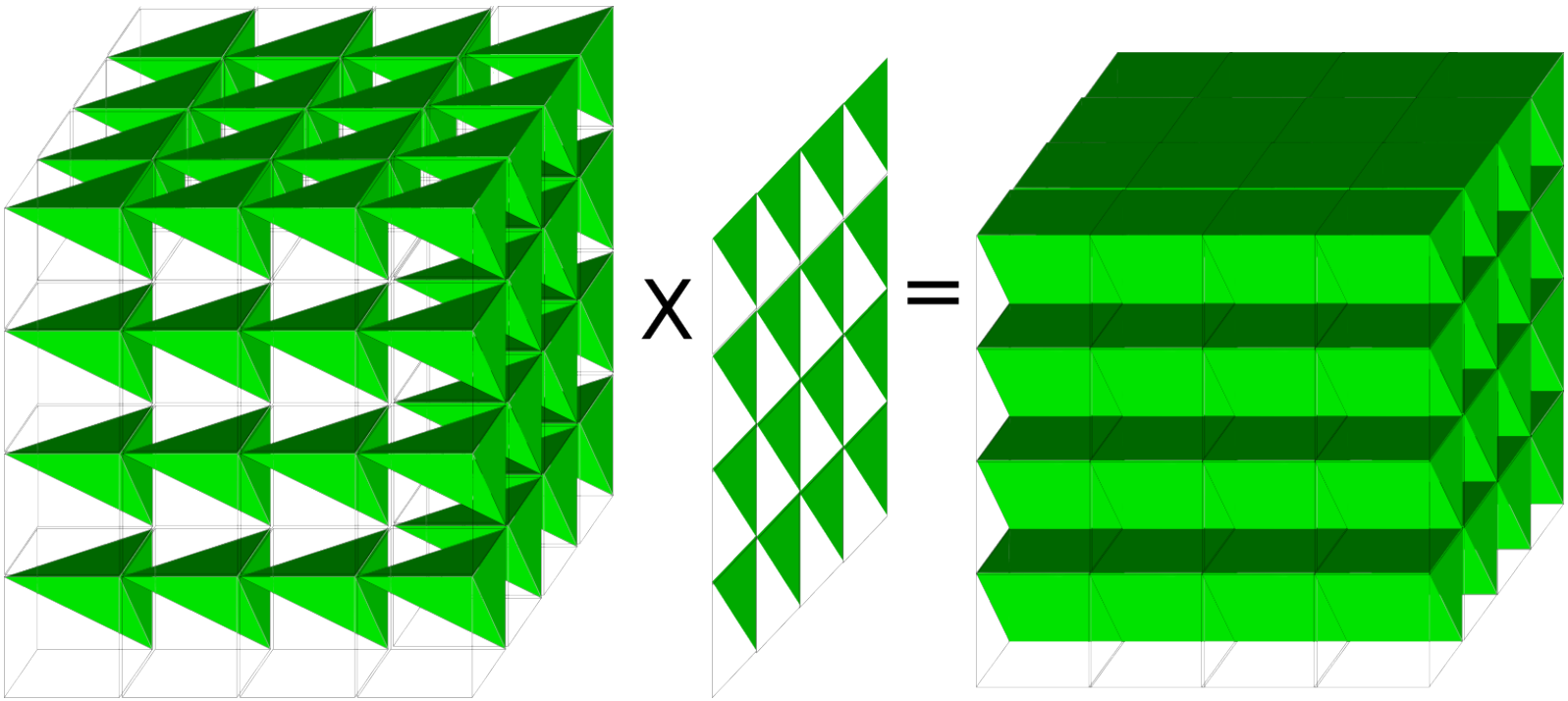
Blocked vs block-cyclic vs cyclic decompositions



3D tensor contraction

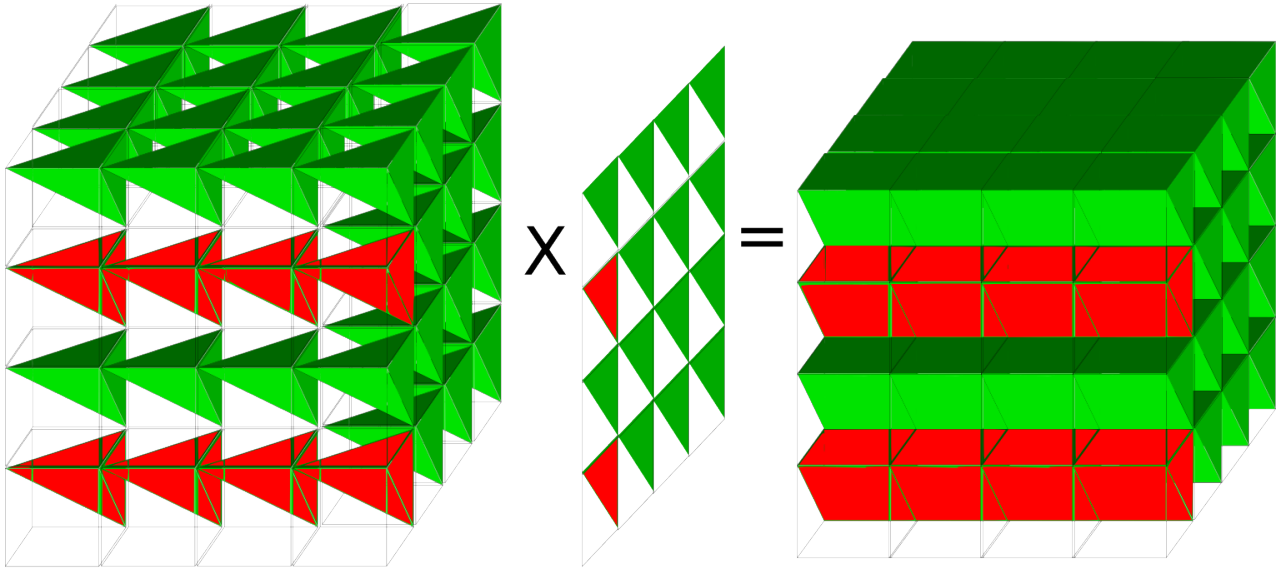


3D tensor cyclic decomposition



3D tensor mapping

Red portion denotes what processor (2,1) owns



P11	P12	P13	P14
P21	P22	P23	P24

A cyclic layout is still challenging

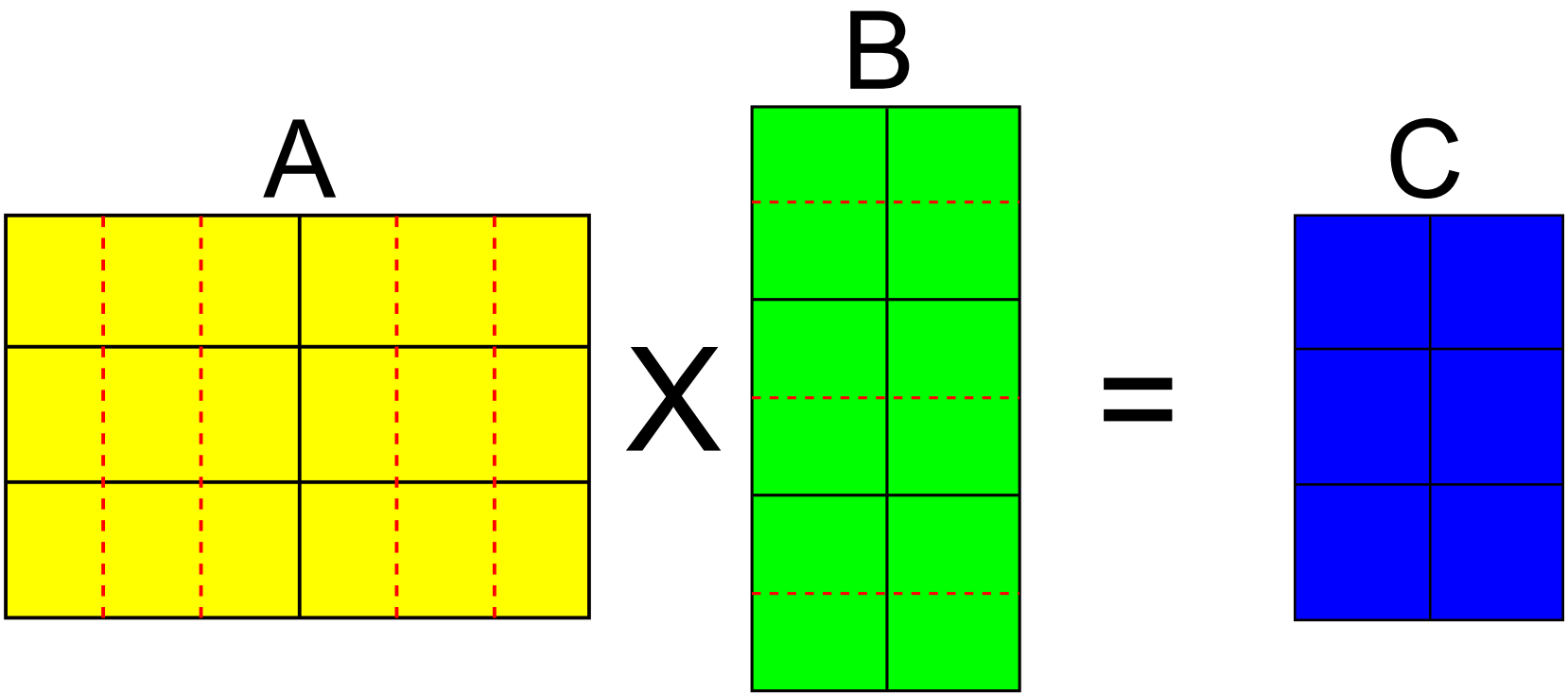
- In order to retain partial symmetry, all symmetric dimensions of a tensor must be mapped with the same cyclic phase
- The contracted dimensions of A and B must be mapped with the same phase
- And yet the virtual mapping, needs to be mapped to a physical topology, which can be any shape

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.



3D 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 Coupled Cluster results on Blue Gene/Q

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

- Already scaled on up to 1024 nodes of BG/Q, up to 480 virtual orbitals
- Preliminary results already favorable performance with respect to NWChem
- Spending 30-40% of time in DGEMM, with good strong and weak scalability

Future Work

3D eigensolver

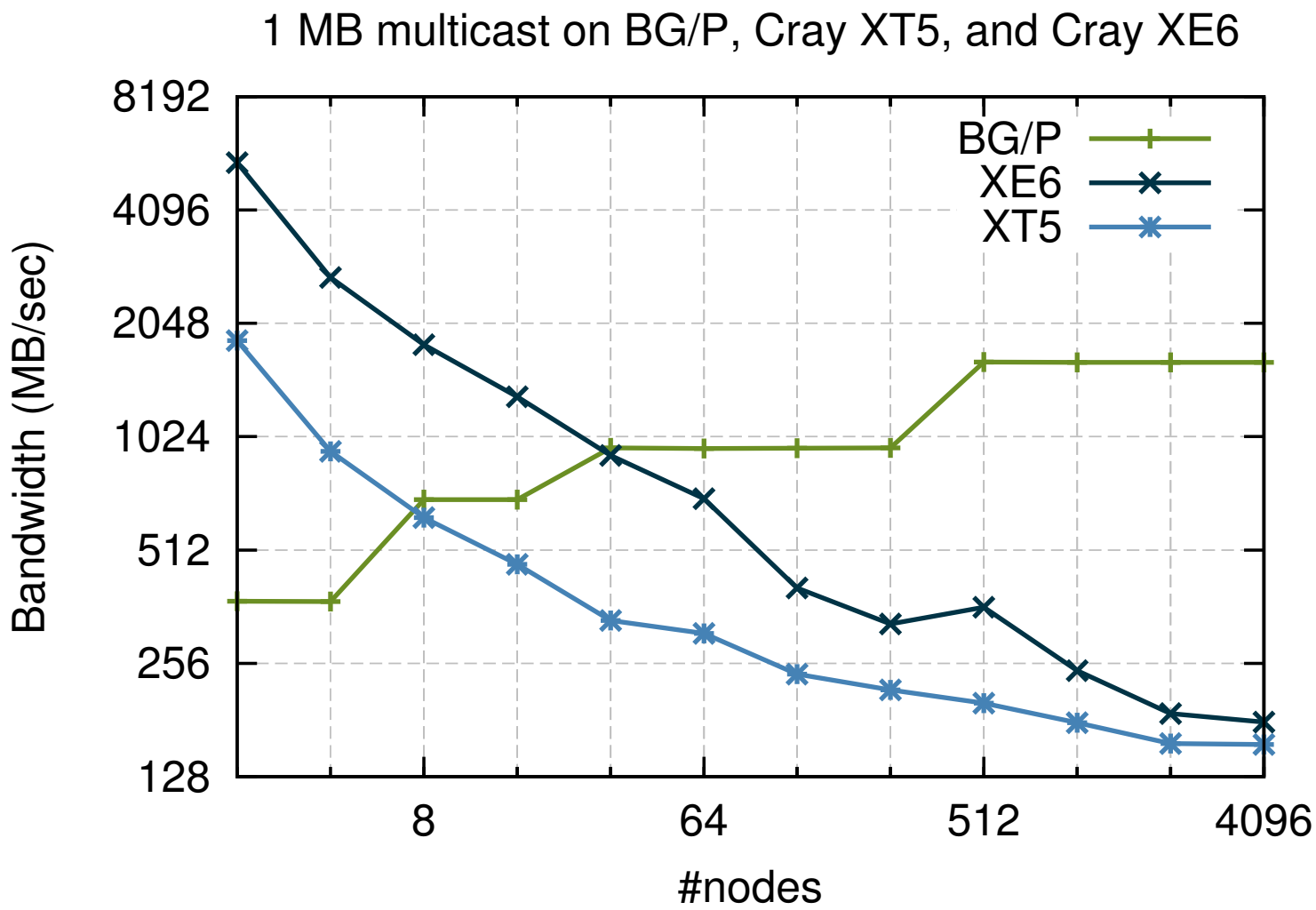
- Working on formalization and error proof of YT reconstruction
- Plan to implement 3D QR and 3D symmetric eigensolve
- Integrate with QBox

Cyclops Tensor Framework

- Implement CCSD, CSSD(T), CCSDT, CSSDTQ
- Merge with SCF and eigensolver codes
- Sparse tensors
- Consider multi-term factorization/other tensor computations

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