Communication-avoiding parallel algorithms for dense linear algebra

Edgar Solomonik

Department of EECS, UC Berkeley

June 2013

Outline

Communication costs more than computation

Communication happens off-chip and on-chip and incurs two costs

- latency - time per message
- bandwidth - amount of data per unit time

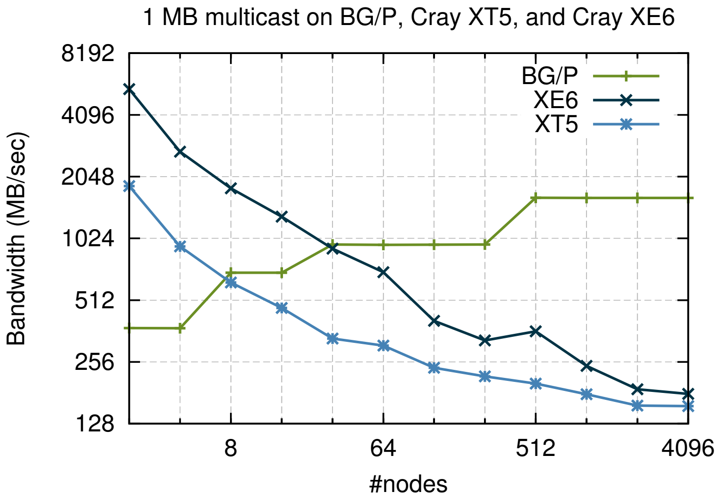
These costs are becoming more expensive relative to flops

Table: Annual improvements

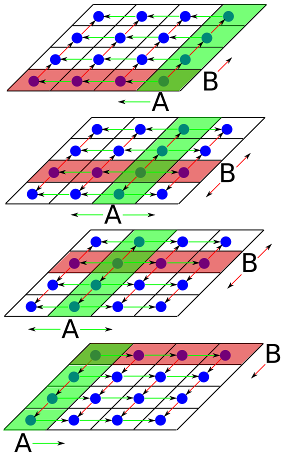
time per flop		bandwidth	latency
59%	network	26%	15%
	DRAM	23%	5%

Source: James Demmel [FOSC]

Topology-aware collective communication

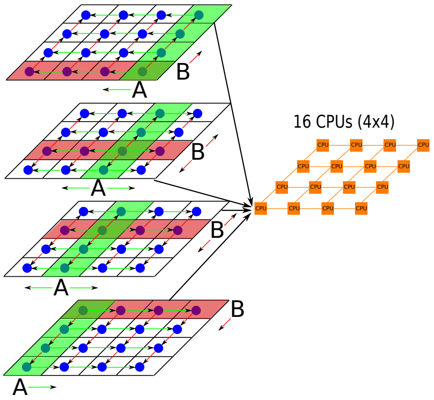


Blocking matrix multiplication



2D matrix multiplication

[Cannon 69],
[Van De Geijn and Watts 97],
[Agarwal et al 95]



$O(n^3/p)$ flops

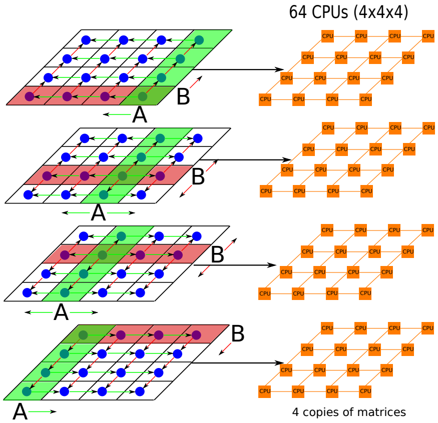
$O(n^2/\sqrt{p})$ words moved

$O(\sqrt{p})$ 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

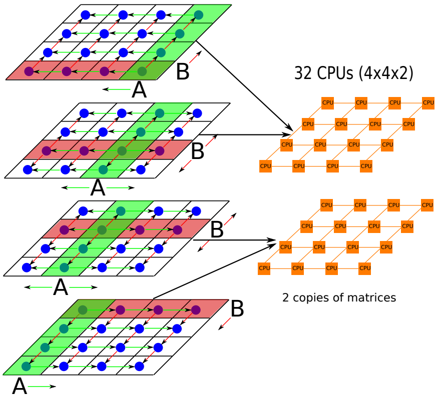
$O(n^2/p^{2/3})$ words moved

$O(1)$ messages

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

2.5D matrix multiplication

[McColl and Tiskin 99]
[Solomonik and Demmel 11]



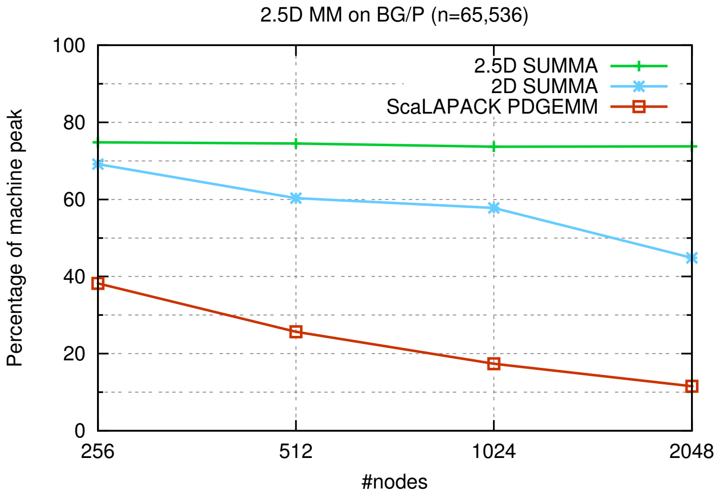
$O(n^3/p)$ flops

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

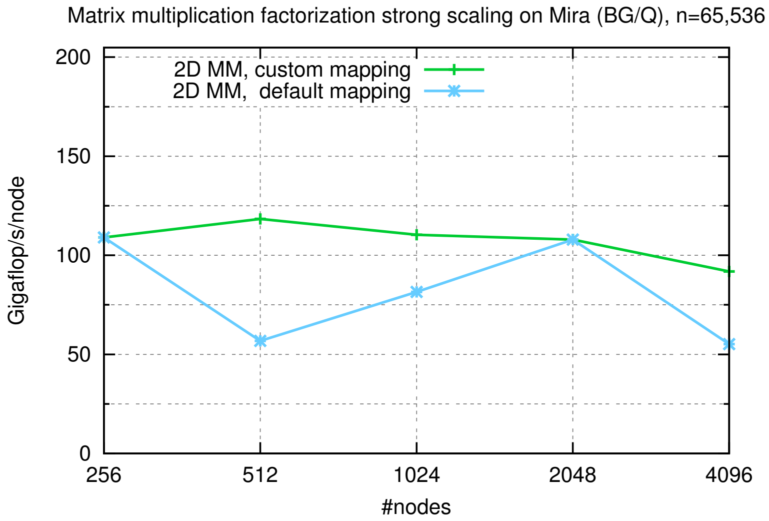
$O(\sqrt{p/c^3})$ messages

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

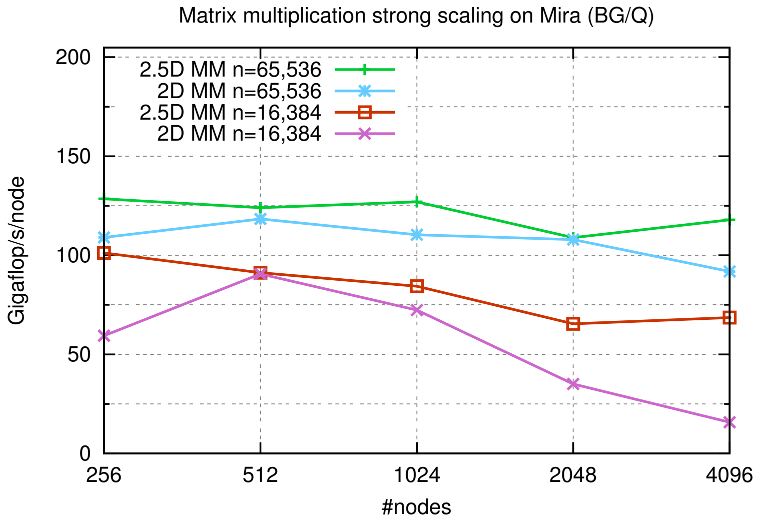
Strong scaling matrix multiplication



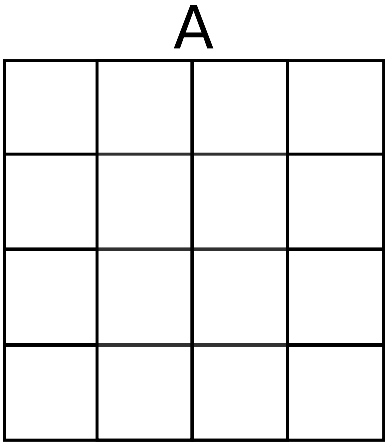
Topology-aware mapping on BG/Q



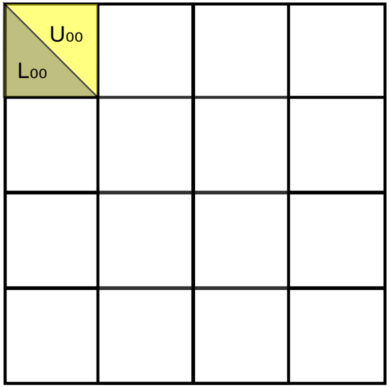
Benefit of replication on BG/Q



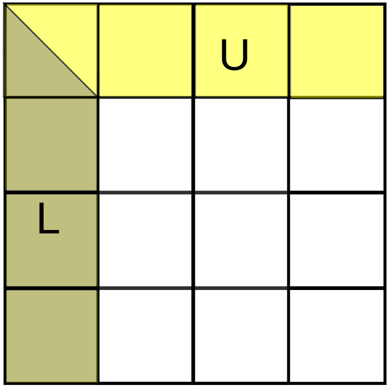
2D blocked LU factorization



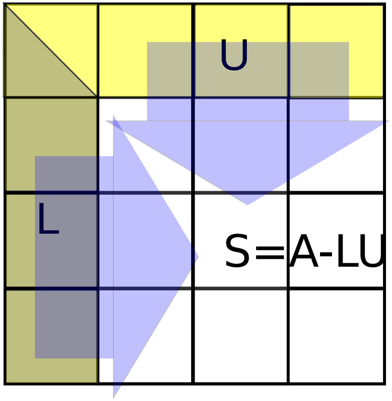
2D blocked LU factorization



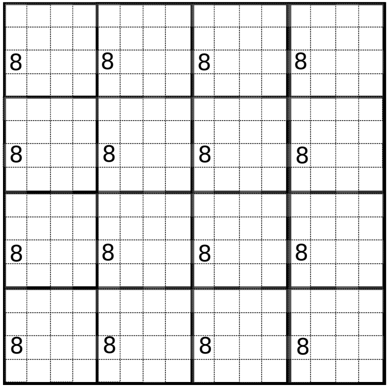
2D blocked LU factorization



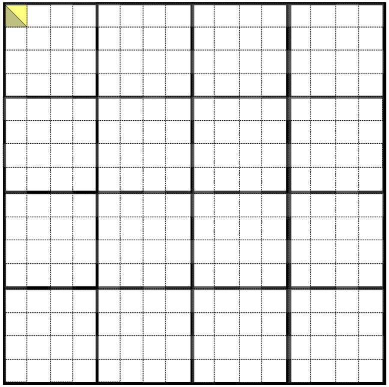
2D blocked LU factorization



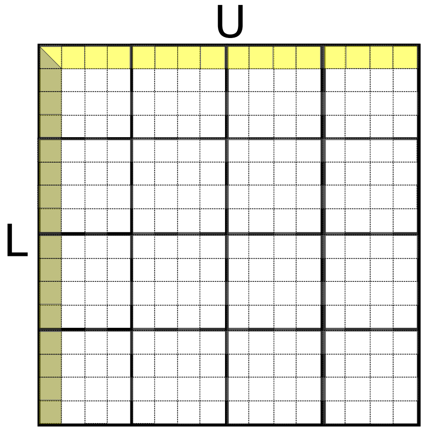
2D block-cyclic decomposition



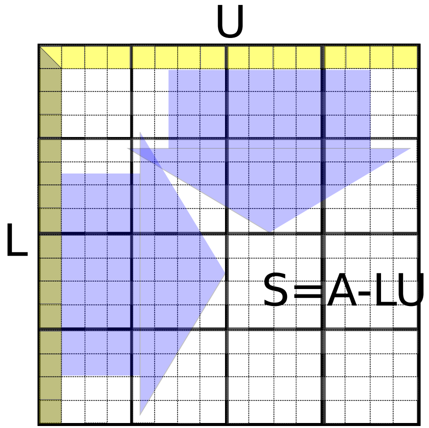
2D block-cyclic LU factorization



2D block-cyclic LU factorization



2D block-cyclic LU factorization

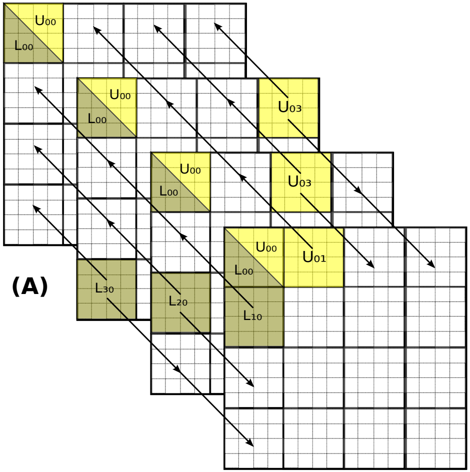


3D recursive non-pivoted LU and Cholesky

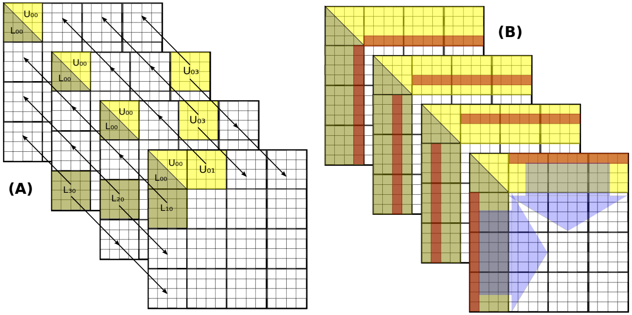
A 3D recursive algorithm with no pivoting [A. Tiskin 2002]

- Tiskin gives algorithm under the BSP model
  - Bulk Synchronous Parallel
  - considers communication and synchronization
- We give an alternative distributed-memory version and implementation
- Also, we have a new lower-bound for the latency cost

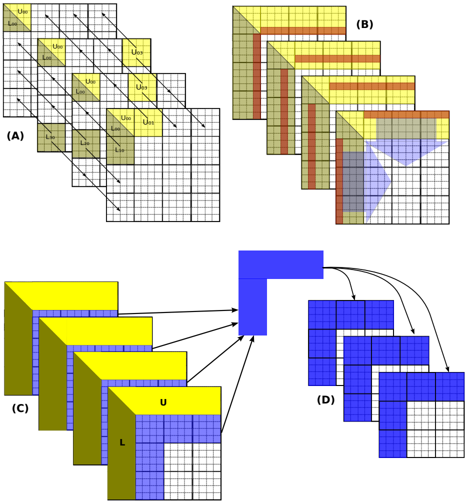
2.5D LU factorization



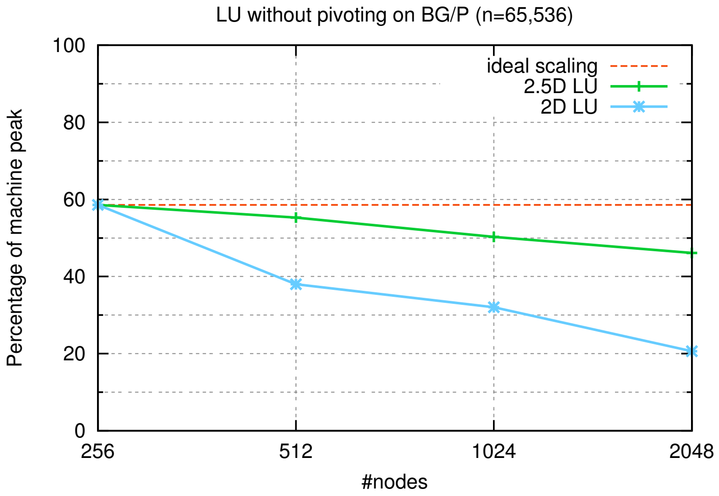
2.5D LU factorization



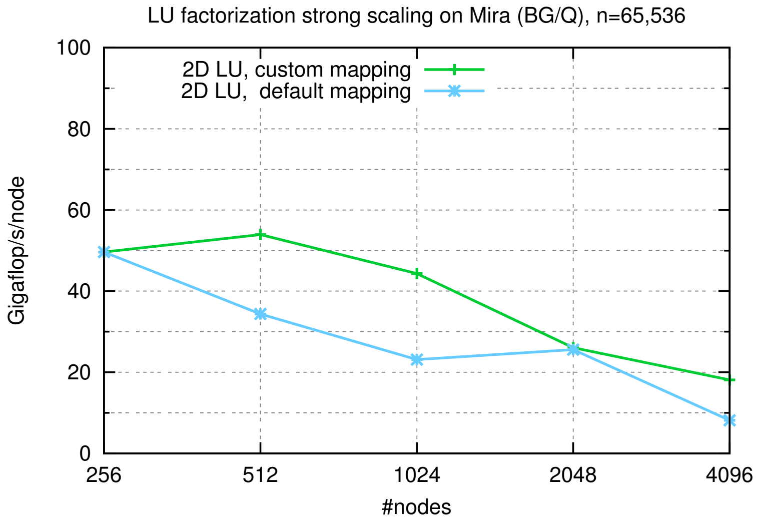
2.5D LU factorization



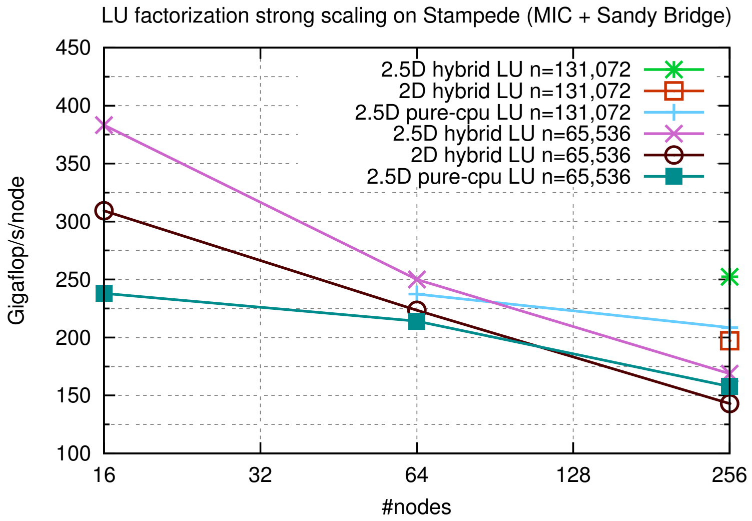
2.5D LU strong scaling (without pivoting)



Topology-aware mapping on BG/Q



Hybrid 2.5D LU factorization



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

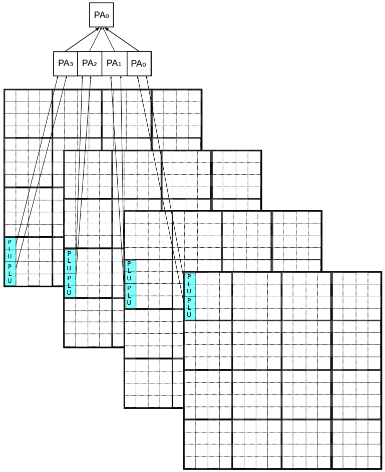
Partial pivoting is not communication-optimal on a blocked matrix

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

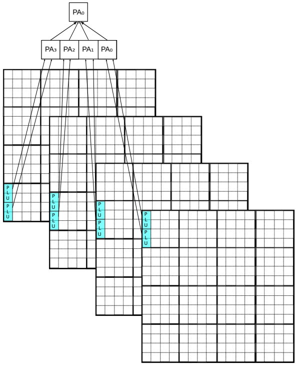
Tournament pivoting is communication-optimal

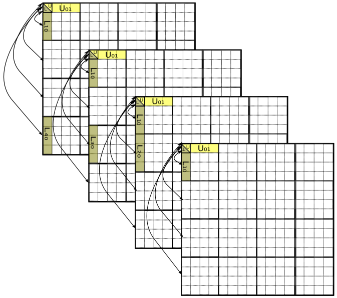
- performs a tournament to determine best pivot row candidates
- passes up 'best rows' of A

2.5D LU factorization with tournament pivoting

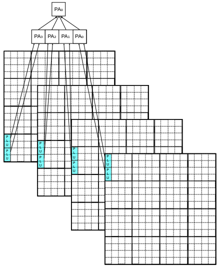


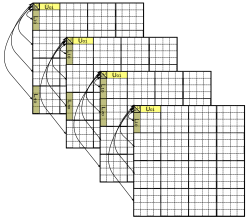
2.5D LU factorization with tournament pivoting

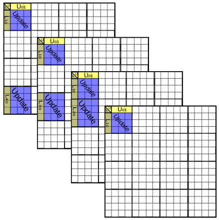




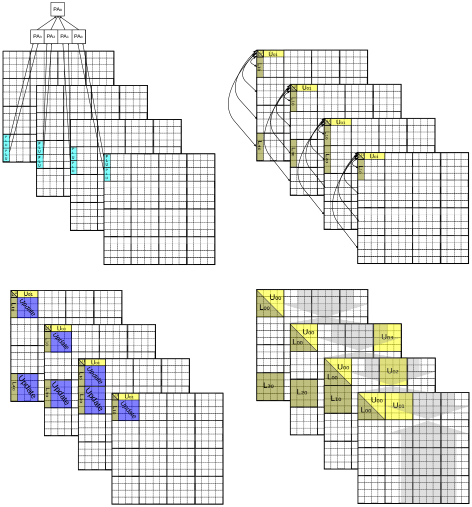
2.5D LU factorization with tournament pivoting



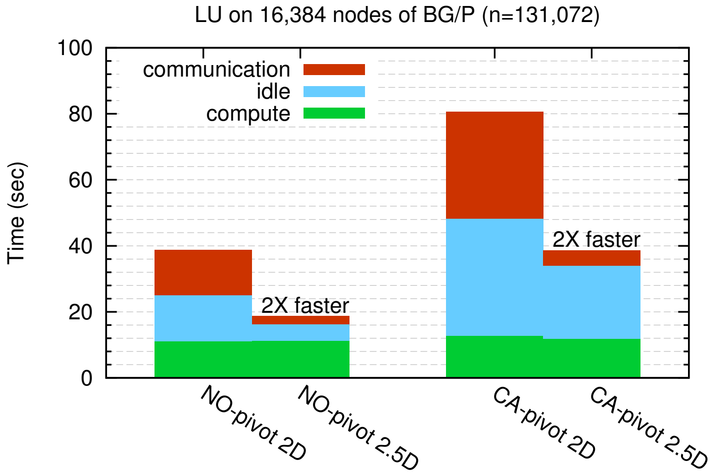




2.5D LU factorization with tournament pivoting



2.5D LU on 65,536 cores



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

2.5D QR factorization

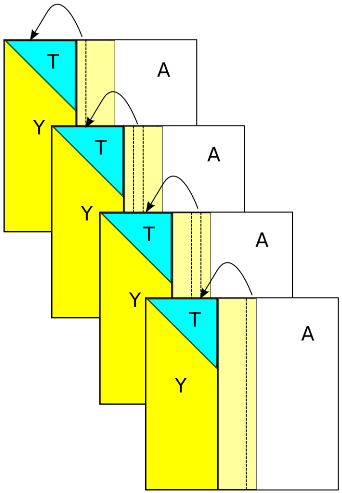
- The orthogonalization updates $(I - 2yy^T)$ do not commute so aggregate them into $(I - YTY)^T$.
- To minimize latency perform recursive TSQR on the panel
- Must reconstruct Householder Y from TSQR Q, R

Householder reconstruction

Yamamoto's algorithm

- Given $A = QR$ for tall-skinny A,
- perform LU on $(Q_1 - I)$ to get $LU([Q_1 - I, Q_2]) = Y \cdot (TY^T)$.
- as stable as QR in practice

3D QR using YT representation



Symmetric eigensolve via QR

Need to apply two sided updates to reduce to tridiagonal T

$$T = (I - YTY^T)A(I - YT^TY^T)$$
$$V = AYT^T - \frac{1}{2}YTY^TAYT^T$$
$$T = A - YV^T - VY^T$$

We use TSQR to compute panels of Y and reduce A to banded form.

2.5D symmetric eigensolve

Algorithm outline

- Compute TSQR on each subpanel $A_i = Q_i \cdot R_i$ to reduce A to band size $n/\sqrt{pc}$
- Recover Y_i from Q_i and A_i via Yamamoto's method
- Accumulate $Y = [Y_1, Y_2 \ldots Y_i]$ on processor layers and apply in parallel to next panel A_{i+1}
- Reduce from banded to tridiagonal using symmetric band reduction with $\sqrt{pc}$ processors
- Use MRRR to compute eigenvalues of the tridiagonal matrix

Summary of theoretical results for 2.5D algorithms

A comparison between asymptotic communication cost in ScaLAPACK (SCL) and in 2.5D algorithms ($\log(p)$ factors suppressed). All matrices are n-by-n. For 2.5D algorithms, $c \in [1, p^{1/3}]$

problem	lower bound	2.5D lat	2.5D bw	SCL lat	SCL bw
MM	$W = \Omega(n^2/p^{2/3})$	$\sqrt{p/c^3}$	$n^2/\sqrt{pc}$	$\sqrt{p}$	$n^2/\sqrt{p}$
TRSM	$W \cdot S^2 = \Omega(n^2)$	$\sqrt{p/\sqrt{c}}$	$n^2/\sqrt{pc}$	$\sqrt{p}$	$n^2/\sqrt{p}$
Cholesky	$W \cdot S = \Omega(n^2)$	$\sqrt{pc}$	$n^2/\sqrt{pc}$	$\sqrt{p}$	$n^2/\sqrt{p}$
LU	$W \cdot S = \Omega(n^2)$	$\sqrt{pc}$	$n^2/\sqrt{pc}$	n	$n^2/\sqrt{p}$
QR	$W \cdot S = \Omega(n^2)$	$\sqrt{pc}$	$n^2/\sqrt{pc}$	n	$n^2/\sqrt{p}$
sym eig	$W \cdot S = \Omega(n^2)$	$\sqrt{pc}$	$n^2/\sqrt{pc}$	n	$n^2/\sqrt{p}$

Dependency bubble

Q: Why must we sacrifice latency to lower bandwidth cost in LU/Cholesky?
A: graph expansion!

Definition (Dependency bubble)

We consider the expansion of dependencies associated with a path $R = \{v_1, \dots v_n\}$, where each v_i, for $i \in [2, n]$ is dependent on v_{i-1}. We define the dependency bubble around P as $B(R) \subset V$ where each vertex $u_i \in B(R)$ lays on a dependency path, $\{w, \dots u_i \dots z\}$ in G where $w, z \in R$. This bubble corresponds to vertices which must be computed between the computations of v_1 and v_n (the start and end of the path).

Dependency bubble expansion

Recall that a balanced vertex separator Q of a graph $G = (V, E)$, splits $V - Q = W_1 + W_2$ so that $min(|W_1|, |W_2|) \geq \frac{1}{4}|V|$ and $E = W_1 \times (Q + W_1) + W_2 \times (Q + W_2)$.

Definition (Dependency bubble cross-section expansion)

If $B(R)$ is the dependency bubble formed around path R, the **bubble cross-section expansion**, $E(R)$ is the minimum size of a balanced vertex separator of $B(R)$.

General latency lower-bound based on bubble expansion

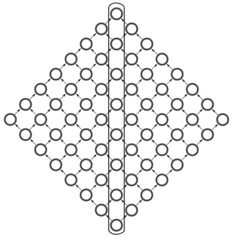
Theorem (Bubble Expansion Theorem)

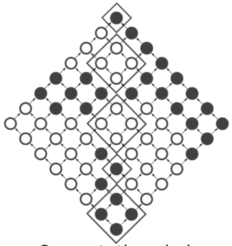
Let P be a dependency path in G, such that any subsequence $R \subset P$, has bubble cross-section expansion $E(R) = \Omega(\epsilon(|R|))$ and bubble size $|B(R)| = \Omega(\sigma(|R|))$, where $\epsilon(b) = b^{d_1}$, and $\sigma(b) = b^{d_2}$ for positive integers d_1, d_2 The bandwidth and latency costs of any parallelization of G must obey the relations

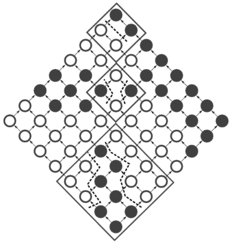
$$F = \Omega(\sigma(b) \cdot |P|/b), \qquad W = \Omega(\epsilon(b) \cdot |P|/b), \qquad S = \Omega(|P|/b)$$

for all $b \in [1, |P|]$.

Dependency bubble expansion along path


Dependency path R


Computation chain


Communication chain

Solution to triangular system of linear equations (TRSV)

Consider solving for $\mathbf{x}$ where L is lower-triangular in

$$y_i = \sum_{j \le i}^{n} l_{ij} \cdot x_j.$$

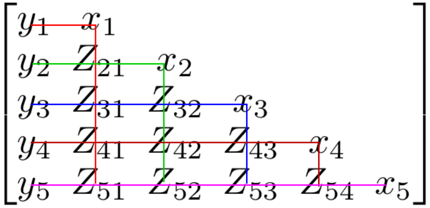
Define vertices corresponding to computations as $v_{ij} = (l_{ij}, y_i)$ in addition to input vertices corresponding to elements of L and y.

Theorem (Latency-bandwidth Trade-off in TRSV)

The parallel computation of x in $y = L \cdot x$ where L is a lower-triangular n-by-n matrix, incurs latency cost S and bandwidth cost W,

$$W \cdot S^2 = \Omega(n^2)$$

TRSV dependency hypergraph


y_1 x_1
y_2 Z_{21} x_2
y_3 Z_{31} Z_{32} x_3
y_4 Z_{41} Z_{42} Z_{43} x_4
y_5 Z_{51} Z_{52} Z_{53} Z_{54} x_5

Cholesky factorization

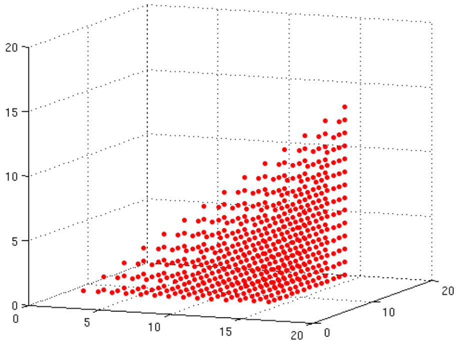
We can use bubble expansion to prove better latency lower bounds for LU, as well as Cholesky, and QR factorizations.

Theorem (Latency-bandwidth Trade-off in Cholesky Factorization)

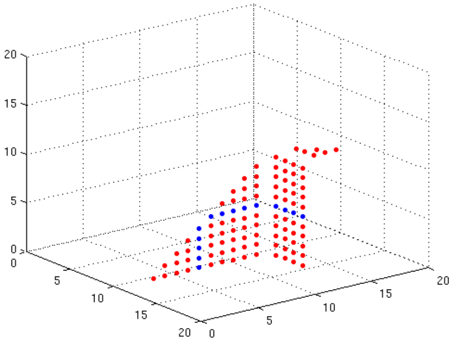
The parallel computation of lower-triangular L for symmetric positive definite A such that $A = LL^T$ where all matrices are n-by-n, must incur flops cost F, latency cost S, and bandwidth cost W, such that

$$W \cdot S = \Omega(n^2) \quad \textit{and} \quad F \cdot S^2 = \Omega(n^3)$$

Cholesky computational structure



Cholesky dependency hypergraph



Krylov subspace methods

Definition (Krylov subspace methods)

Compute $A^k x$, where A typically corresponds to a sparse graph.

Theorem

To compute $A^k x$, where A corresponds to a 3^d-point stencil, the bandwidth W and latency S costs are lower-bounded by

$$F = \Omega(k \cdot b^d), \quad W = \Omega(k \cdot b^{d-1}), \quad S = \Omega(k/b),$$

for any b. We can rewrite these relations as

$$W \cdot S^{d-1} = \Omega(k^d), \quad F \cdot S^d = \Omega(k^{d+1}).$$

Electronic structure theory

Electronic structure calculations attempt to model the ground-state (and sometimes excited-state) energies of chemical systems, taking into account of quantum effects.

Density Functional Theory is the most common method

- cost is typically $O(n^3)$ for n electrons
- models system as a density functional, corrects for correlation
- good for metals and regular systems
- bad at molecules due to correlation effects on boundary

Coupled Cluster models electronic correlation explicitly

- cost is typically $O(n^{4+d})$, where $d \in \{2, 4, 6\}$
- the most accurate method used in practice

Coupled Cluster definition

Coupled Cluster (CC) is a method for computing an approximate solution to the time-independent Schrödinger equation of the form

$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle,$$

CC rewrites the wave-function $|\Psi\rangle$ as an excitation operator $\hat{\mathbf{T}}$ applied to the Slater determinant $|\Phi_0\rangle$

$$|\Psi\rangle = e^{\hat{\mathbf{T}}}|\Phi_0\rangle$$

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

$$\begin{aligned}\hat{\mathbf{T}}_{\mathrm{CCSD}} &= \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2\\ \hat{\mathbf{T}}_{\mathrm{CCSDT}} &= \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3\\ \hat{\mathbf{T}}_{\mathrm{CCSDTQ}} &= \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3 + \hat{\mathbf{T}}_4\end{aligned}$$

Coupled Cluster with Double excitations (CCD) equations

$e^{\hat{\mathbf{T}}_2}|\Phi_0\rangle$ turns into:

$$R^{ab}_{ij} = V^{ab}_{ij} + P(ia,jb)\Big[T^{ae}_{ij}I^{b}_{e} - T^{ab}_{im}I^{m}_{j} + \frac{1}{2}V^{ab}_{ef}T^{ef}_{ij} + \frac{1}{2}T^{ab}_{mn}I^{mn}_{ij} - T^{ae}_{mj}I^{mb}_{ie} - I^{ma}_{ie}T^{eb}_{mj} + (2T^{ea}_{mi} - T^{ea}_{im})I^{mb}_{ej}\Big]$$

$$\begin{aligned}
I^{a}_{b} &= (-2V^{mn}_{eb} + V^{mn}_{be})T^{ea}_{mn} \\
I^{i}_{j} &= (2V^{mi}_{ef} - V^{im}_{ef})T^{ef}_{mj} \\
I^{ij}_{kl} &= V^{ij}_{kl} + V^{ij}_{ef}T^{ef}_{kl} \\
I^{ia}_{jb} &= V^{ia}_{jb} - \frac{1}{2}V^{im}_{eb}T^{ea}_{jm} \\
I^{ia}_{bj} &= V^{ia}_{bj} + V^{im}_{be}(T^{ea}_{mj} - \frac{1}{2}T^{ae}_{mj}) - \frac{1}{2}V^{mi}_{be}T^{ae}_{mj}
\end{aligned}$$

NWChem approach to contractions

A high-level description of NWChem's algorithm for tensor contractions:

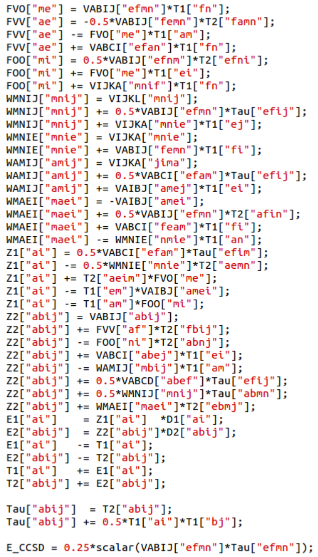
- data layout is abstracted away by the Global Arrays framework
- Global Arrays uses one-sided communication for data movement
- packed tensors are stored in blocks
- for each contraction, each process does a subset of the block contractions
- each block is transposed and unpacked prior to contraction
- dynamic load balancing is employed among processors

Cyclops Tensor Framework (CTF) approach to contractions

A high-level description of CTF's algorithm for tensor contractions:

- packed tensors are decomposed cyclically among toroidal processor grids
- MPI collectives are used for all communication
- for each contraction, a distributed layout is selected based on internal performance models
- performance model considers all possible execution paths
- before contraction, tensors are redistributed to a new layout
- if there is enough memory, the tensors are (partially) unpacked
- all preserved symmetries and non-symmetric indices are folded in preparation for matrix multiplication
- nested distributed matrix multiply algorithms are used to perform the contraction in a load-balanced manner

CCSD code using our domain specific language



```
FVO["me"] = VABIJ["efmn"]*T1["fn"];
FVV["ae"] = -0.5*VABIJ["femn"]*T2["famn"];
FVV["ae"] -= FVO["me"]*T1["am"];
FVV["ae"] += VABCI["efan"]*T1["fn"];
FOO["mi"] = 0.5*VABIJ["efnm"]*T2["efni"];
FOO["mi"] += FVO["me"]*T1["ei"];
FOO["mi"] += VIJKA["mnif"]*T1["fn"];
WMNIJ["mnij"] = VIJKL["mnij"];
WMNIJ["mnij"] += 0.5*VABIJ["efmn"]*Tau["efij"];
WMNIJ["mnij"] += VIJKA["mnie"]*T1["ej"];
WMNIE["mnie"] = VIJKA["mnie"];
WMNIE["mnie"] += VABIJ["femn"]*T1["fi"];
WAMIJ["amij"] = VIJKA["jima"];
WAMIJ["amij"] += 0.5*VABCI["efam"]*Tau["efij"];
WAMIJ["amij"] += VAIBJ["amej"]*T1["ei"];
WMAEI["maei"] = -VAIBJ["amei"];
WMAEI["maei"] += 0.5*VABIJ["efmn"]*T2["afin"];
WMAEI["maei"] += VABCI["feam"]*T1["fi"];
WMAEI["maei"] -= WMNIE["nmie"]*T1["an"];
Z1["ai"] = 0.5*VABCI["efam"]*Tau["efim"];
Z1["ai"] -= 0.5*WMNIE["mnie"]*T2["aemn"];
Z1["ai"] += T2["aeim"]*FVO["me"];
Z1["ai"] -= T1["em"]*VAIBJ["amei"];
Z1["ai"] -= T1["am"]*FOO["mi"];
Z2["abij"] = VABIJ["abij"];
Z2["abij"] += FVV["af"]*T2["fbij"];
Z2["abij"] -= FOO["ni"]*T2["abnj"];
Z2["abij"] += VABCI["abej"]*T1["ei"];
Z2["abij"] -= WAMIJ["mbij"]*T1["am"];
Z2["abij"] += 0.5*VABCD["abef"]*Tau["efij"];
Z2["abij"] += 0.5*WMNIJ["mnij"]*Tau["abmn"];
Z2["abij"] += WMAEI["maei"]*T2["ebmj"];
E1["ai"]    = Z1["ai"]  *D1["ai"];
E2["abij"]  = Z2["abij"]*D2["abij"];
E1["ai"]   -= T1["ai"];
E2["abij"] -= T2["abij"];
T1["ai"]   += E1["ai"];
T2["abij"] += E2["abij"];

Tau["abij"]  = T2["abij"];
Tau["abij"] += 0.5*T1["ai"]*T1["bj"];

E_CCSD = 0.25*scalar(VABIJ["efmn"]*Tau["efmn"]);
```

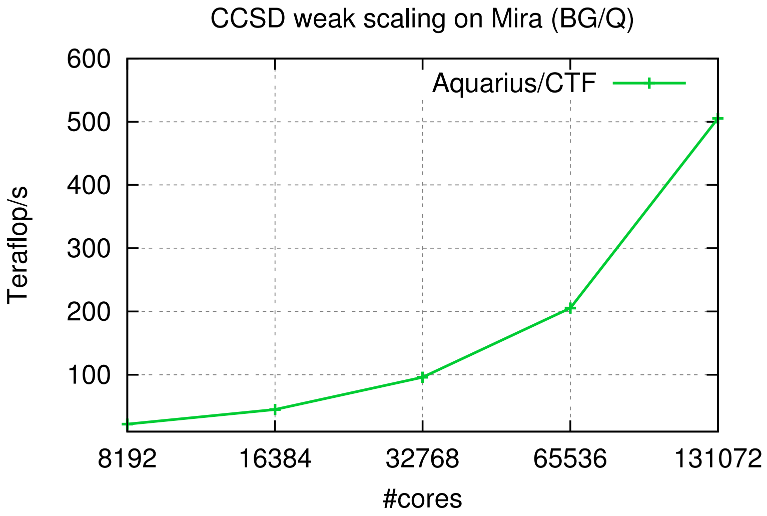
Comparison with NWChem on Cray XE6

CCSD iteration time on 64 nodes of Hopper:

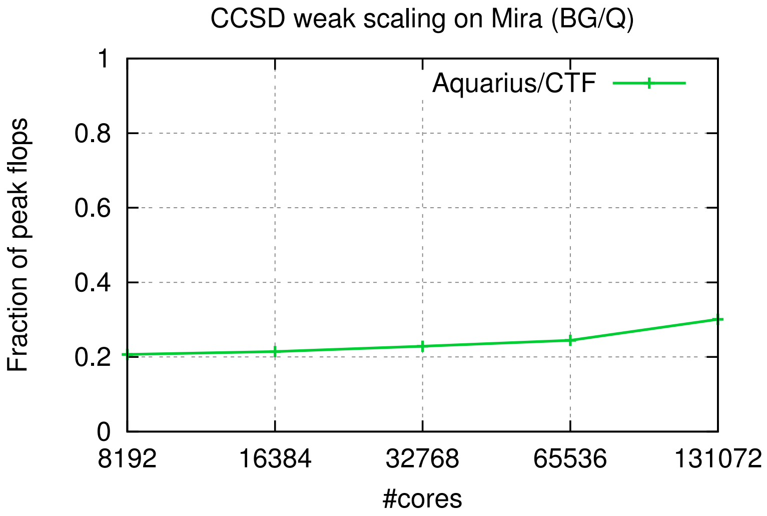
system	# electrons	# orbitals	CTF	NWChem
w5	25	205	14 sec	36 sec
w7	35	287	90 sec	178 sec
w9	45	369	127 sec	-
w12	60	492	336 sec	-

On 128 nodes, NWChem completed w9 in 223 sec, CTF in 73 sec.

Blue Gene/Q up to 1250 orbitals, 250 electrons



Coupled Cluster efficiency on Blue Gene/Q



Summary and conclusion

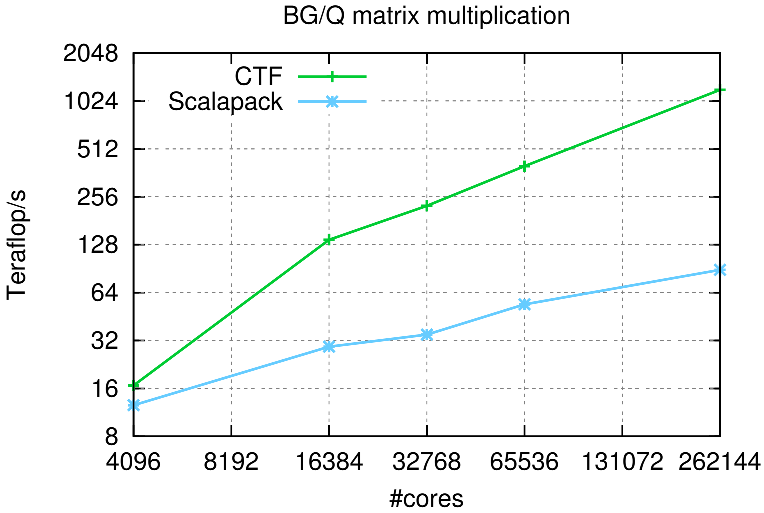
- Communication cost and load balance matter, especially in parallel
- We can lower bound bandwidth based on projections and latency based on dependencies and graph expansion
- 2.5D algorithms present a communication-optimal algorithm family for dense linear algebra
- 2.5D matrix multiplication and LU factorization work well in practice
- CTF is a parallel framework for symmetric tensor contractions

Acknowledgements

- Collaborators:
  - James Demmel (adviser)
  - Grey Ballard (2.5D QR, 2.5D TRSM, 2.5D sym eig)
  - Erin Carson, Nick Knight (latency lower bounds)
  - Evangelos Georganas (2.5D with overlap, 1.5D MD)
  - Katherine Yelick, Michael Driscoll, Penporn Koanantakool (1.5D MD)
  - (INRIA) Mathias Jacquelin, Laura Grigori (2.5D QR)
  - Hong-Diep Nguyen (2.5D QR)
  - (LBNL) Aydın Buluç (2.5D all-pairs shortest-paths)
  - (UT Austin) Devin Matthews (CTF)
  - (Argonne) Jeff Hammond (CTF)
- funding and support
  - DOE Computational Science Graduate Fellowship (CSGF)
  - access to Argonne, NERSC, LLNL, and TACC resources

Backup slides

Comparison with ScaLAPACK on BG/Q



Communication lower bound for tensor contractions

The computational graph corresponding to a tensor contraction can be higher dimensional, but there are still only three projections corresponding to $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$. So, if the contraction necessitates F floating point operations, the bandwidth lower bound is still just

$$W_p = \Omega\left(\frac{F}{p \cdot \sqrt{M}}\right).$$

Therefore. folding contractions into matrix multiplication and running a good multiplication algorithm is communication-optimal.

Cyclic decomposition in CTF

Cyclical distribution is fundamental to CTF, hence the name Cyclops (cyclic-operations).
Given a vector **v** of length n on p processors

- in a blocked distribution process p_i owns $\{v_{i \cdot n/p+1}, \ldots v_{(i+1) \cdot n/p}\}$
- in a cyclic distribution process p_i owns $\{v_i, v_{2i}, \ldots v_{(n/p)i}\}$

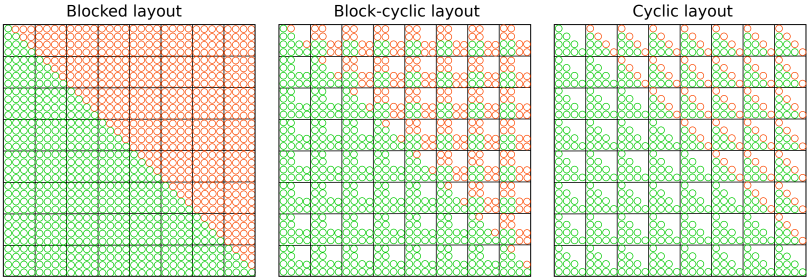
A cyclic distribution is associated with a phase along each dimension (for the vector above this was p). The main advantage from this distribution is that each subtensor can retain packed structure with only minimal padding.
CTF assumes all subtensor symmetries have index relations of the form $\leq$ and not $<$, so in effect, diagonals are stored for skew-symmetric tensors.

Sequential tensor contractions

A cyclic distribution provides a vital level of abstraction, because each subtensor contraction becomes a packed contraction of the same sort as the global tensor contraction but of smaller size. Given a sequential packed contraction kernel, CTF can parallelize it automatically. Further, because each subcontraction is the same, the workload of each processor is the same. The actual sequential kernel used by CTF employs the following steps

1. if there is enough memory, unpack broken symmetries
2. perform a nonsymmetric transpose, to make all indices of non-broken symmetry be the leading dimensions
3. use a naive kernel to iterate though indices with broken symmetry and call BLAS GEMM for the leading dimensions

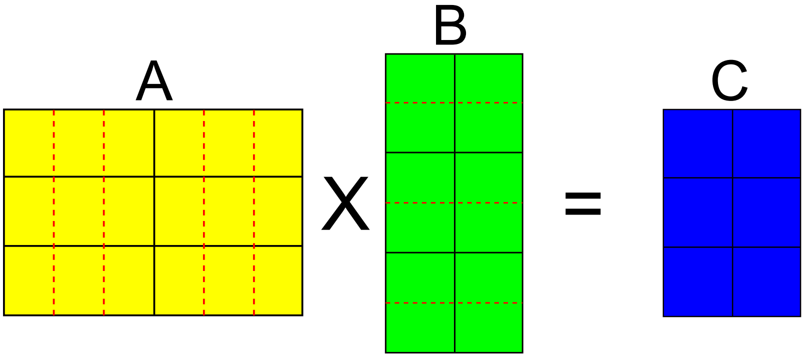
Blocked vs block-cyclic vs cyclic decompositions



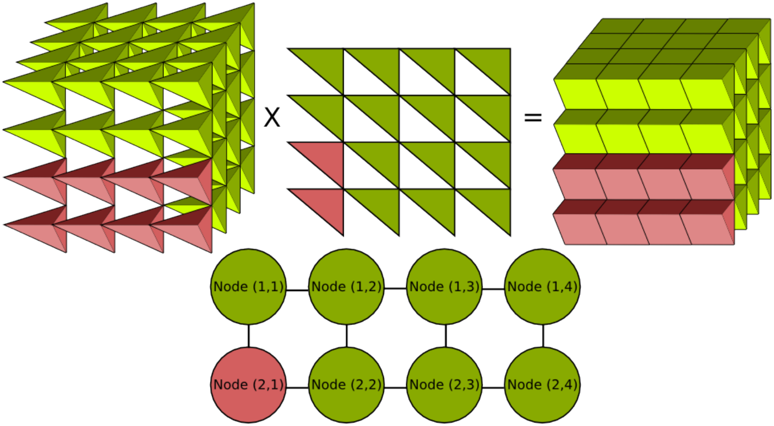
○ Green denotes fill (unique values) ○ Red denotes padding / load imbalance

Virtualization

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



3D tensor mapping



A simple data layout

Replicated, distributed nonsymmetric tensor (processor grid)
of nonsymmetric tensors (virtual grid)
of symmetric tensors (folded broken symmetries)
of matrices (unfolded broken and folded preserved symmetries)

The layout typically changes for each tensor between each contraction.

Tensor redistribution

Our symmetric tensor data layout has a global ordering and a local ordering

- the local data is not in global order
- cannot compute local data index from global index
- cannot compute global data index from local index
- can iterate over local data and obtain global index
- can iterate over global data and obtain local index

Given these constraints, it is simplest to compute the global index of each piece of data and sort.

General data redistribution

We use an algorithm faster than sorting for redistribution

1. iterate over the local data and count where the data must be sent
2. communicate counts and compute prefix sums to obtain offsets
3. iterate over the local data **in global order** and bin it
4. exchange the data (MPI all to all v)
5. iterate over the new local data **in global order** and retrieve it from bins

This method is much faster, because it does not explicitly form and communicate keys for the data.

Threaded general redistribution

In order to hide memory latency and reduce integer operations it is imperative to thread the redistribution kernel

- prefix sums and counts are trivial to thread
- to thread the iterator over data, we must give each thread different global indices
- each thread moves the local data corresponding to a global index partition, preserving the ordering

Interface and code organization

- the CTF codebase is currently 31,345 lines of C++ code
- CTF provides functionality for general tensor contractions, including a contraction domain-specific language (DSL)
- Aquarius is a quantum chemistry package being developed by Devin Matthews
  - uses CTF for parallel tensor contraction execution
  - provides a DSL for spin-integrated tensor contractions
  - gives implementations of CC methods including other necessary components (e.g. SCF)
- efforts are underway to also integrate CTF into the QChem package

Multidimensional processor grids

CTF supports tensors and processor grids of any dimension because mapping a symmetric tensor to a processor grid of the same dimension preserves symmetric structure with minimal virtualization and padding. Processor grids are defined by

- a base grid, obtained from the physical topology or from factorizing the number of processors
- folding all possible combinations of adjacent processor grid dimensions

Tensors are contracted on higher dimensional processor grids by

- mapping an index shared by two tensors in the contraction to different processor grid dimensions
- running a distributed matrix multiplication algorithm for each such 'mismatched' index
- replicating data along some processor dimensions 'a la 2.5D'

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.

2.5D algorithms for DFT

2.5D 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

Our CCSD factorization

Credit to John F. Stanton and Jurgen Gauss

$$\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\
\tilde{F}_i^m &= (1 - \delta_{mi}) f_i^m + \sum_e \tilde{F}_e^m t_i^e + \frac{1}{2} \sum_{nef} v_{ef}^{mn} t_{in}^{ef} + \sum_{fn} v_{if}^{mn} t_n^f,
\end{aligned}$$

Our CCSD factorization

$$\begin{aligned}
\tilde{W}^{mn}_{ei} &= v^{mn}_{ei} + \sum_{f} v^{mn}_{ef} t^{f}_{i}, \\
\tilde{W}^{mn}_{ij} &= v^{mn}_{ij} + P^{i}_{j} \sum_{e} v^{mn}_{ie} t^{e}_{j} + \frac{1}{2} \sum_{ef} v^{mn}_{ef} \tau^{ef}_{ij}, \\
\tilde{W}^{am}_{ie} &= v^{am}_{ie} - \sum_{n} \tilde{W}^{mn}_{ei} t^{a}_{n} + \sum_{f} v^{ma}_{ef} t^{f}_{i} + \frac{1}{2} \sum_{nf} v^{mn}_{ef} t^{af}_{in}, \\
\tilde{W}^{am}_{ij} &= v^{am}_{ij} + P^{i}_{j} \sum_{e} v^{am}_{ie} t^{e}_{j} + \frac{1}{2} \sum_{ef} v^{am}_{ef} \tau^{ef}_{ij}, \\
z^{a}_{i} &= f^{a}_{i} - \sum_{m} \tilde{F}^{m}_{i} t^{a}_{m} + \sum_{e} f^{a}_{e} t^{e}_{i} + \sum_{em} v^{ma}_{ei} t^{e}_{m} + \sum_{em} v^{ae}_{im} \tilde{F}^{m}_{e} + \frac{1}{2} \sum_{efm} \\
z^{ab}_{ij} &= v^{ab}_{ij} + P^{i}_{j} \sum_{e} v^{ab}_{ie} t^{e}_{j} + P^{a}_{b} P^{i}_{j} \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{mj} - P^{a}_{b} \sum_{m} \tilde{W}^{am}_{ij} t^{b}_{m} + P^{a}_{b}
\end{aligned}$$

Performance breakdown on BG/Q

Performance data for a CCSD iteration with 200 electrons and 1000 orbitals on 4096 nodes of Mira
4 processes per node, 16 threads per process
Total time: 18 mins
n_v-orbitals, n_o-electrons, p-processors, M-local memory size

kernel	% of time	complexity	architectural bounds
DGEMM	45%	$O(n_v^4 n_o^2/p)$	flops/mem bandwidth
broadcasts	20%	$O(n_v^4 n_o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	10%	$O(p)$	allreduce bandwidth
data packing	7%	$O(n_v^2 n_o^2/p)$	integer ops
all-to-all-v	7%	$O(n_v^2 n_o^2/p)$	bisection bandwidth
tensor folding	4%	$O(n_v^2 n_o^2/p)$	memory bandwidth

Performance breakdown on Cray XE6

Performance data for a CCSD iteration with 100 electrons and 500 orbitals on 256 nodes of Hopper
4 processes per node, 6 threads per process
Total time: 9 mins
v-orbitals, o-electrons

kernel	% of time	complexity	architectural bounds
DGEMM	21% $\Downarrow$ 24%	$O(v^4o^2/p)$	flops/mem bandwidth
broadcasts	32% $\Uparrow$ 12%	$O(v^4o^2/p\sqrt{M})$	multicast bandwidth
prefix sum	7% $\Downarrow$ 3%	$O(p)$	allreduce bandwidth
data packing	10% $\Uparrow$ 3%	$O(v^2o^2/p)$	integer ops
all-to-all-v	8%	$O(v^2o^2/p)$	bisection bandwidth
tensor folding	4%	$O(v^2o^2/p)$	memory bandwidth