Scalable Numerical Algorithms for Electronic Structure Calculations

Edgar Solomonik, James Demmel

Department of EECS, UC Berkeley

March 2013

Outline

Collaborators

- advising
  - James Demmel, Katherine Yelick
- Numerical linear algebra algorithms
  - (UC Berkeley) Grey Ballard, Erin Carson, Michael Driscoll, Evangelos Georganas, Penporn Koanantakool, Nick Knight, Benjamin Lipshitz, Hong-Diep Nguyen, Oded Schwartz
  - (Lawrence Berkeley National Laboratory) Aydın Buluç
  - (INRIA) Mathias Jacquelin, Laura Grigori
- Electronic structure calculations
  - (UT Austin) Devin Matthews
  - (Argonne National Laboratory) Jeff Hammond
  - (Lawrence Livermore National Laboratory) Abhinav Bhatele, Erik Draeger, Todd Gamblin, Martin Schulz

Electronic structure calculations

Quantum many-body methods

- allow study of chemical problems at the quantum level
- Attempt to find approximate solutions to the Schrödinger equation
$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle$$
- Density Functional Theory
  - The wave-function Ψ is modelled implicitly via a particle density
- Coupled Cluster
  - The wave-function Ψ is explicitly approximated via expansion and truncation of an exponential

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

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

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

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

where V_s has a exchange-correlation potential correction,

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

Density Function Theory (DFT), contd.

The exchange-correlation potential $\mathbf{V_{XC}}$ 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 $\mathbf{V_s}$ 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 dense linear algebra kernels

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

We aim to introduce scalable algorithms for these problems and integrate them into QBox

- QBox already uses our matrix multiplication at large scale on Sequoia
- Library versions of factorizations are in development

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)

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

Coupled Cluster derivation

To derive CC equations, a normal-ordered Hamiltonian is defined as the sum of one-particle and two-particle interaction terms

$$\hat{\mathbf{H}}_N = \hat{\mathbf{F}}_N + \hat{\mathbf{V}}_N$$

Solving the CC energy contribution can be done by computing eigenvectors of the similarity-transformed Hamiltonian

$$\bar{\mathbf{H}} = e^{-\hat{\mathbf{T}}}\hat{\mathbf{H}}_N e^{\hat{\mathbf{T}}}$$

Performing the CCSD truncation $\hat{\mathbf{T}} = \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2$ and applying the Hadamard lemma of the Campbell-Baker-Hausdorff formula,

$$\bar{\mathbf{H}} = \hat{\mathbf{H}}_N + [\hat{\mathbf{H}}_N, \hat{\mathbf{T}}_1] + [\hat{\mathbf{H}}_N, \hat{\mathbf{T}}_2] + \frac{1}{2}[[\hat{\mathbf{H}}_N \hat{\mathbf{T}}_1], \hat{\mathbf{T}}_1] \ldots$$

which simplifies to

$$\bar{\mathbf{H}} = \hat{\mathbf{H}}_N + \hat{\mathbf{H}}_N\hat{\mathbf{T}}_1 + \hat{\mathbf{H}}_N\hat{\mathbf{T}}_2 + \hat{\mathbf{H}}_N\hat{\mathbf{T}}_1^2 + \ldots$$

Coupled Cluster equations

Left projecting the eigenvector equation, we can obtain an explicit formula for the CC energy via Wick contraction

$$E_{\text{CCSD}} - E_0 = \langle \Phi_0 | \bar{\mathbf{H}} | \Phi_0 \rangle = \sum_{ia} f_{ia} t_i^a + \frac{1}{4} \sum_{abij} \langle ij || ab \rangle t_{ij}^{ab} + \frac{1}{2} \sum_{aibj} \langle ij || ab \rangle t_i^a t_j^b$$

The tensor amplitude equations are derived in a similar fashion but involve many more terms

$$0 = \langle \Phi_i^a | \bar{\mathbf{H}} | \Phi_0 \rangle = f_{ai} - \sum_{kc} f_{kc} t_i^c t_k^a + \ldots$$

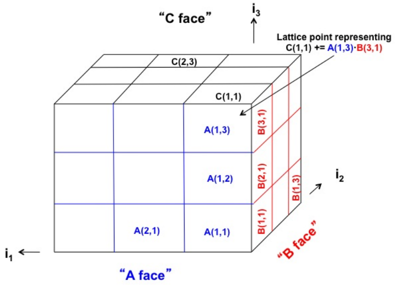
$$0 = \langle \Phi_{ij}^{ab} | \bar{\mathbf{H}} | \Phi_0 \rangle = \langle ab || ij \rangle + \sum_{bj} \langle ja || bi \rangle t_j^b + \ldots$$

These equations then need to be factorized into two-tensor contractions.

Matrix multiplication

For square n-by-n matrix $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, compute

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$



Communication lower bounds for matrix multiplication

In 1981, Hong and Kung derived a lower-bound on the sequential communication cost of matrix multiplication, given cache size M,

$$W_{\text{seq}} = \Omega\left(\frac{n^3}{M^{1/2}}\right) \quad S_{\text{seq}} = \Omega\left(\frac{n^3}{M^{3/2}}\right).$$

In 2004, Irony, Tiskin, and Toledo, extended this to the case of p processors with local memory of size M

$$W_{\text{par}} = \Omega\left(\frac{n^3}{p \cdot M^{1/2}}\right) \quad S_{\text{par}} = \Omega\left(\frac{n^3}{p \cdot M^{3/2}}\right).$$

These bounds hold for $M \in [3n^2/p, 3n^2/p^{2/3}]$.

Parallel matrix multiplication algorithms

Standard '2D' algorithms ([Cannon 69], [GW 97], [ABGJP 95]) assume $M = 3n^2/p$ and block **A**, **B**, and **C**. on a $\sqrt{p}$-by-$\sqrt{p}$ processor grid. They have a cost of

$$W_{2\text{D}} = O\left(\frac{n^2}{\sqrt{p}}\right)$$

'3D' algorithms ([Bernsten 89], [ACS 1990], [ABGJP 95], [MT 99]) assume $M = 3n^2/p^{2/3}$ and block the computation on a $p^{1/3}$-by-$p^{1/3}$-$p^{1/3}$ processor grid, yielding

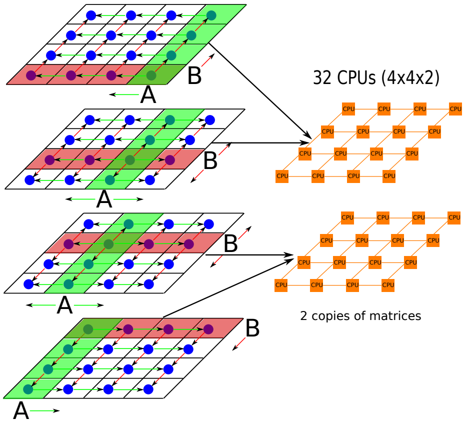
$$W_{3\text{D}} = O\left(\frac{n^2}{p^{2/3}}\right)$$

'2.5D' algorithms ([MT 99], [SD 2011]) generalize this and, for any $c \in [1, p^{1/3}]$ attain the lower bound with memory usage $M = cn^2/p$,

$$W_{2.5\text{D}} = O\left(\frac{n^2}{\sqrt{cp}}\right)$$

2.5D matrix multiplication

[McColl and Tiskin 99],
[Solomonik and Demmel 2011]



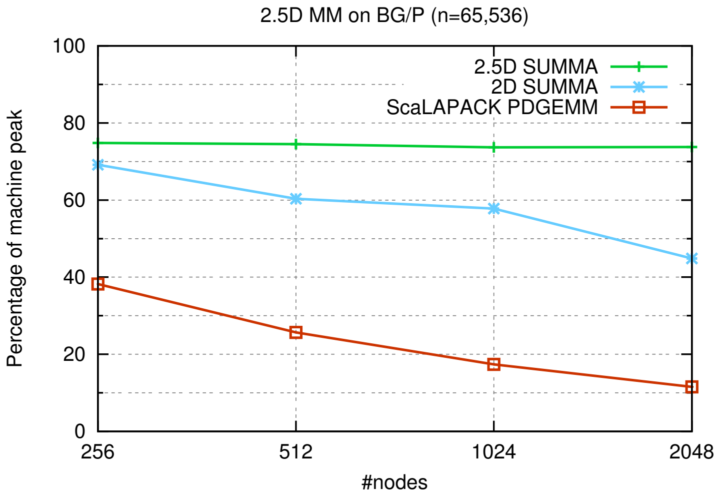
$O(n^3/p)$ flops

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

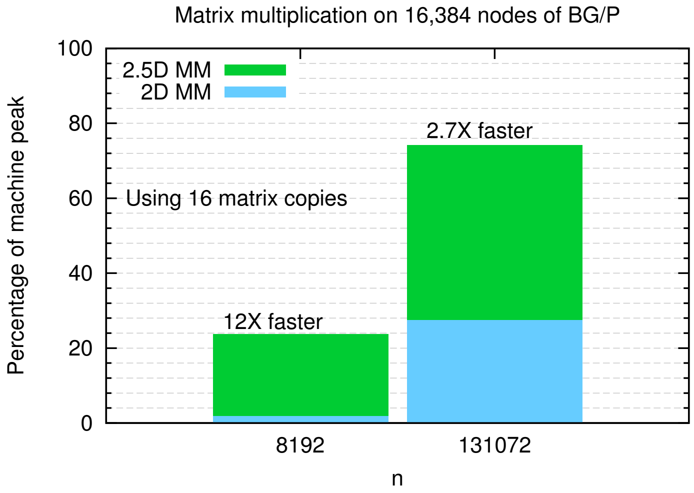
$O(\sqrt{p/c^3})$ messages

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

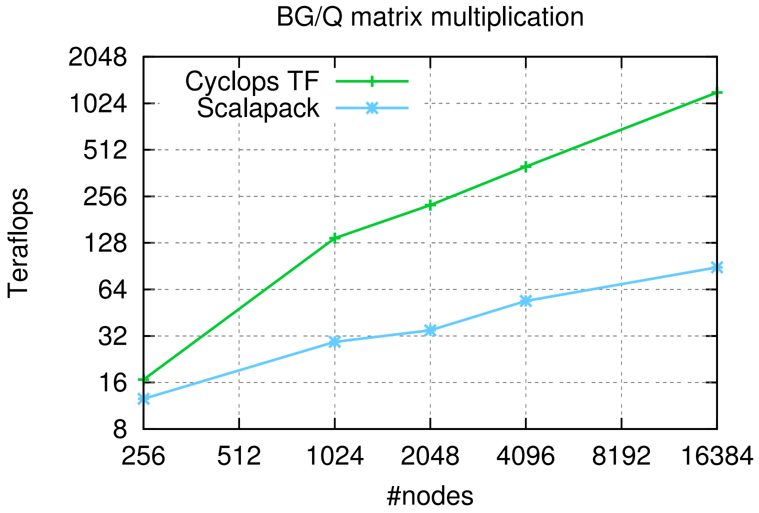
Strong scaling matrix multiplication



Strong scaling matrix multiplication



2.5D matrix multiplication on BG/Q



Tensor contractions

We define a tensor contraction between $\mathcal{A} \in \mathbb{R}^{\otimes k}$, $\mathcal{B} \in \mathbb{R}^{\otimes l}$ into $\mathcal{C} \in \mathbb{R}^{\otimes m}$ as

$$c_{i_1 \ldots i_m} = \sum_{j_1 \ldots j_{k+l-m}} a_{i_1 \ldots i_{m-l} j_1 \ldots j_{k+l-m}} \cdot b_{j_1 \ldots j_{k+l-m} i_{m-l+1} \ldots i_m}$$

Tensor contractions reduce to matrix multiplication via index folding (let $[ijk]$ denote a group of 3 indices folded into one),

$$c_{[i_1 \ldots i_{m-l}],[i_{m-l+1} \ldots i_m]} = \sum_{[j_1 \ldots j_{k+l-m}]} a_{[i_1 \ldots i_{m-l}],[j_1 \ldots j_{k+l-m}]} \cdot b_{[j_1 \ldots j_{k+l-m}],[i_{m-l+1} \ldots i_m]}$$

so here $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ can be treated simply as matrices.

Tensor symmetry

Tensors can have symmetry e.g.

$$a_{(ij)k} = a_{(ji)k} \quad \text{or} \quad a_{(ij)k} = -a_{(ji)k}$$

I am introducing more dubious notation, by denoting symmetric groups of indices as $(ab...)$. We now might face contractions like

$$c_{(ij)kl} = \sum_{pqr} a_{(ij)(pq)} \cdot b_{(pqk)(rl)}$$

where the computational graph G can be thought of as a 7D tensor with entries $g_{(ij)kl(pq)r} = (c_{(ij)kl}, a_{(ij)(pq)}, b_{(pqk)(rl)})$. There are two things that can happen to symmetries during a contraction:

- preserved, e.g. $g_{(ij)kl(pq)r} = g_{(ji)kl(pq)r}$
- broken, e.g. $b_{(pqk)(rl)} = b_{(pqk)(lr)}$ but $g_{(ij)kl(pq)r} \neq g_{(ij)kr(pq)l}$

Preserved symmetries in contractions

When a d-dimensional symmetry is preserved, a factor of $d!$ can be saved in memory and flops. This is simple to achieve, since the d-dimensional index group can be folded into one index in a packed layout, for instance

$$c_{kl} = 2 \cdot \sum_{[i<j]} a_{k[(i<j)]} \cdot b_{[(i<j)]l}$$

Since we are folding the packed index, the iteration space of this contraction is in effect equivalent to matrix multiplication, and therefore easy to handle.

Broken symmetries in contractions

When a symmetry is broken, no flops can be saved with respect to unpacking. However, memory can be saved as the tensors can remain stored in packed format. Matrix multiplication of two symmetric tensors features a broken symmetry, which can be computed in packed layout as

$$c_{kl} = \sum_i a_{(k<i)} \cdot b_{(i<l)} + a_{(i<k)} \cdot b_{(i<l)} + a_{(k<i)} \cdot b_{(l<i)} + a_{(i<k)} \cdot b_{(l<i)}$$

This requires four matrix multiplications, but each accesses only the lower triangle of the matrices, so only that portion need be stored.

If data replication is correctly utilized in the parallel algorithm unpacking and doing permutations of contractions have equivalent bandwidth costs.

Lower bounds for tensor contractions

Let the size of the index space spanned by the 3 tensors be G. Let the preserved symmetry factor be s_p. We conjecture the following lower-bounds

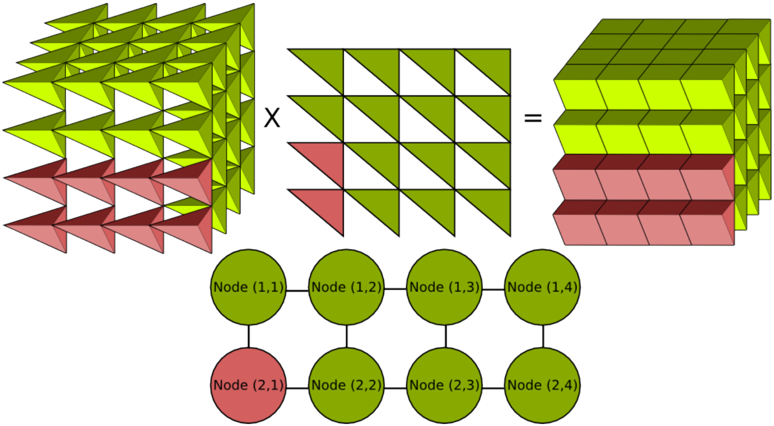
$$W = \Omega\left(\frac{G/s_p}{M^{1/2}}\right) \quad S = \Omega\left(\frac{G/s_p}{M^{3/2}}\right).$$

Cyclops Tensor Framework (CTF) approach to contractions

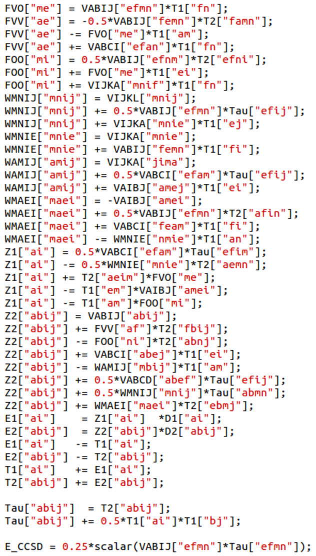
CTF is a massively-parallel framework for tensor contractions

- a cyclic parallel decomposition preserves symmetric structure
- a single transpose/redistribution for each tensor in a contraction
- preserved symmetries are folded
- for broken symmetries, unfold if enough memory, otherwise perform all permutations
- optimizations: threaded transposes, topology-aware mapping, replication (2.5D)

3D tensor mapping



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"]);
```

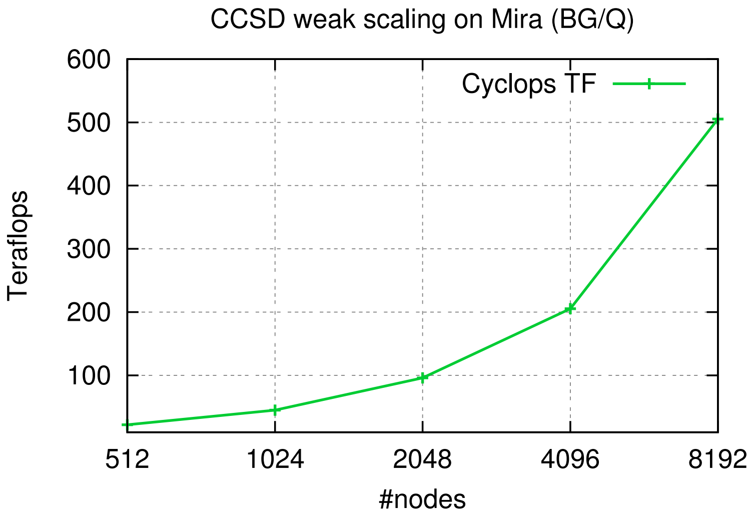
Comparison with NWChem on Cray XE6

CCSD iteration time on 64 nodes of Hopper:

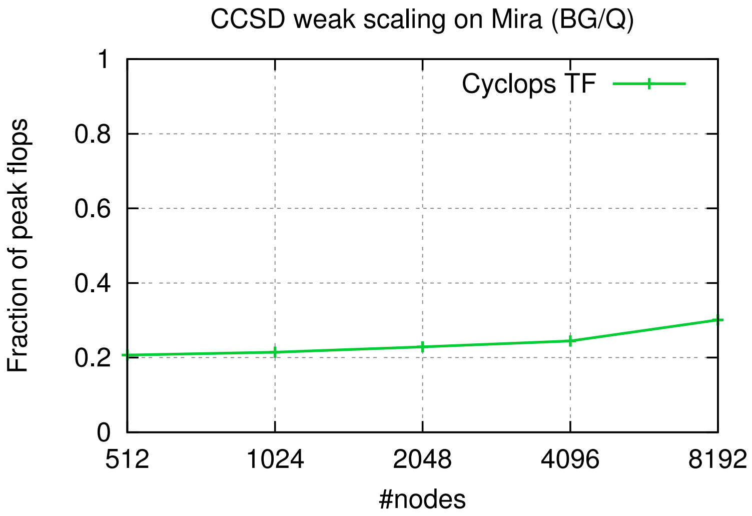
system	# electrons	# orbitals	CTF	NWChem
w5	25	205	14 sec	36 sec
w7	35	287	90 sec	178 sec
w9	45	369	127 sec	-
w12	60	492	336 sec	-

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



A general memory-size-based lower-bound

The matrix multiplication lower bound has been extended to LU, Cholesky, QR, and the SVD of *n*-by-*n* matrices, as well as the all-pairs-shortest-paths problem for a graph with *n* nodes [BDHS 2011]

$$W = \Omega\left(\frac{n^3}{p \cdot M^{1/2}}\right) \quad S = \Omega\left(\frac{n^3}{p \cdot M^{3/2}}\right).$$

Tighter latency lower-bounds for factorizations

Using expansion analysis of dependency graphs, we can prove that for LU, Cholesky, QR, and the SVD of n-by-n matrices, as well as the all-pairs-shortest-paths problem for a graph with n nodes

$$F \cdot S^2 = \Omega(n^3) \quad W \cdot S = \Omega(n^2),$$

and for a triangular solve

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

For k-step Krylov subspace methods on a d-dimensional stencil, we also have

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

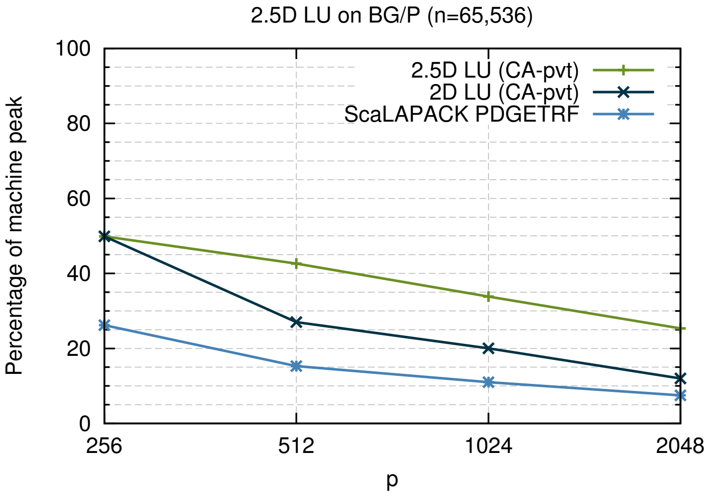
These lower-bounds are independent of the number of processors!

Summary of theoretical results for 2.5D algorithms

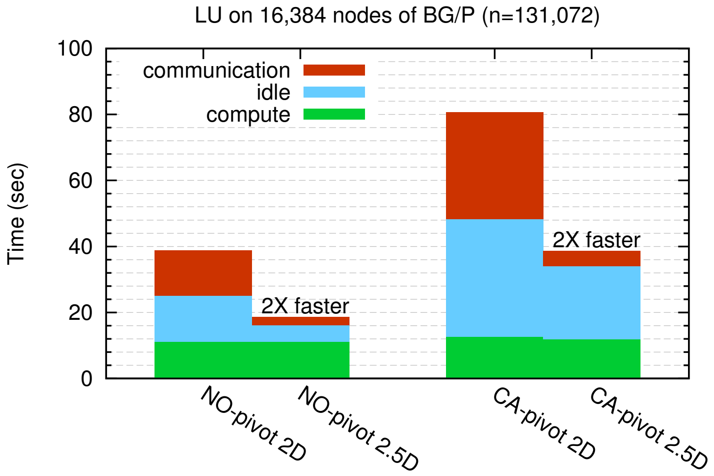
A comparison between asymptotic communication cost in ScaLAPACK (SCL) and in 2.5D algorithms ($\log(p)$ factors suppressed). All matrices are n-by-n. For 2.5D algorithms, $c \in [1, p^{1/3}]$, $M = O(c \cdot n^2/p)$

problem	lower bound	2.5D lat	2.5D bw	SCL lat	SCL bw
MM	$W = \Omega(n^2/\sqrt{cp})$	$\sqrt{p/c^3}$	$n^2/\sqrt{pc}$	$\sqrt{p}$	$n^2/\sqrt{p}$
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	$n^2/\sqrt{p}$

2.5D LU on 65,536 cores



2.5D LU on 65,536 cores



Further references

Websites with more information and papers

- `bebop.cs.berkeley.edu`
- `cs.berkeley.edu/~solomon`

- Memory-based lower-bounds
  - see [BDHS] (SIAM J. Mat. Anal. Appl. 2011)
- 2.5D numerical linear algebra algorithms
  - MM, Cholesky, and LU see [SD] (EuroPar 2011 and SC 2011)
  - APSP see [SBD] (IPDPS 2013)
  - 2.5D+overlap see [GGSTY] (SC 2012)
  - 1.5D molecular dynamics see [DGKSY] (IPDPS 2013)
- Cyclops Tensor Framework
  - see [SMHD] (IPDPS 2013)
- Latency-tradeoff lower-bounds (paper in preparation)

Summary and conclusion

- 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 and some other problems
- CTF is a parallel framework for symmetric tensor contractions
- Coupled Cluster and Density Functional Theory are electronic structure calculation methods implemented on top of CTF

Backup slides

Dependency bubble

Definition (Dependency bubble)

We consider the expansion of dependencies associated with a path $R = \{v_1, \ldots v_n\}$, where each v_i, for $i \in [2, n]$ has a dependency path from 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, $\{w, \ldots u_i \ldots z\}$ 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).

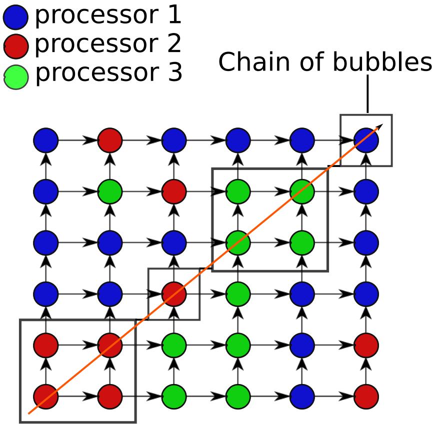
Latency lower bound based on bubble size

Conjecture (Bubble Neighborhood Theorem)

Consider a computation G which has a dependency path R, and any consecutive subsequence $R \subset P$ has a dependency bubble $B(R)$. Given a lower bound on the size of the neighborhood of the bubble $|N(B(R))| = \Omega(\eta(|R|))$, where $\eta(b) = b^k$ for any $b \in [1, |P|]$, the following bandwidth W and latency cost S must be incurred by some processor to compute G,

$$S = \Omega(|P|/b), \qquad W = \Omega(\eta(b)).$$

Dependency bubbles



Proof of Bubble Neighborhood Theorem

Sketch of Proof

Let the length of the longest consecutive subsequence of R computed by a single processor be b. That process must communicate the neighborhood around R, therefore

$$W = \Omega(\eta(b)).$$

Further, there must be $S = \Omega(|P|/b)$ synchronizations in the computation of R, since no chunk of size more than b is computed sequentially.

Example: solution to system of linear equations

Consider solving for **x** where L is lower-triangular in

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

Define vertices corresponding to computations as $v_{ij} = (l_{ij}, y_i)$ in addition to input vertices corresponding to elements of L and y. We can use the concept of the dependency bubble to prove the following conjecture

Conjecture (Latency-bandwidth Trade-off in TRSM)

The parallel computation of $x = L \backslash y$ where L is a lower-triangular n-by-n matrix, must incur latency cost S and bandwidth cost W, such that

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

TRSM latency lower bound

Sketch of Proof

We consider the dependency bubble formed along any dependency path $R = \{v_{jj}, \ldots v_{kk}\}$*, which corresponds to the divide operations which compute* x_j *through* x_k *. The dependency bubble* $B(R)$ *formed by this path includes vertices* v_{ac} *for* $\{a, c \in [j, k], a \geq c\}$*. Each* v_{ac} *has a unique neighbor of the input graph* l_{ac}*, therefore the neighborhood growth around* $B(R)$*, is lower bound by* $|N(B(R))| = \Omega(\eta(|R|)$ *where*

$$\eta(b) = \Omega(b^2)$$

By the Bubble Neighborhood Theorem we have $S = \Omega(n/b)$*,* $W = \Omega(b^2)$

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

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

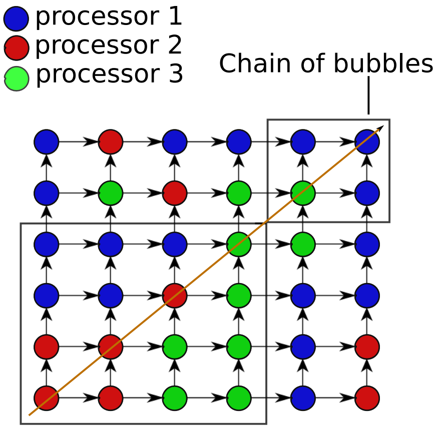
Conjecture (Bubble Expansion Theorem)

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 bubbles



Proof of general latency lower bound

Definition

A parallelization corresponds to a coloring of the vertices, Let $V = \cup V_i$ be a disjoint union of sets V_i where process i computes vertices V_i. Define R_i inductively as the smallest consecutive subsequence of $R_i = P - \cup_{j=1}^{i-1} R_j$, so that

- some process $p_i \in \{1, \ldots p\}$ computes the first entry of R_i
- process p_i computes $|V_{p_i} \cap B(R_i)| \geq \frac{1}{4}|B(R_i)|$ elements and does not compute $|B(R_i) - V_{p_i}| \geq \frac{1}{2}|B(R_i)|$ elements

Due to load balance $|\sum_i R_j| = \Omega(|P|)$.

Proof of general latency lower bound

Sketch of Proof

To compute each $B(R_i)$ at least one synchronization is required. Further, any communication schedule for $V_{p_i} \cap B(R_i)$ must correspond to a set Q of vertices ("communicated values") which separate $V_{p_i} \cap B(R_i)$ from $V_{p_i} - B(R_i)$. Therefore, Q corresponds to a balanced vertex separator on $B(R_i)$,

$$F = \Omega\left(\sum_i \sigma(|R_i|)\right), \qquad W = \Omega\left(\sum_i \epsilon(|R_i|)\right).$$

These costs are minimized when each subsequence R_i is of the same length b, therefore

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

Example: LU factorization

We can use bubble expansion to prove better latency lower bounds for LU, as well as Cholesky, and QR factorizations. LU factorization of square matrices gives a cubic DAG $v_{ijk} = (l_{ik}, u_{kj})$, where

$$a_{ij} = \sum_{k \leq \min(i,j)} l_{ik} \cdot u_{kj}.$$

Conjecture (Latency-bandwidth Trade-off in LU Factorization)

The parallel computation of lower-triangular L and upper-triangular U such that $A = LU$ where all matrices are n-by-n, must incur flops cost F, latency cost S, and bandwidth cost W, such that

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

LU latency lower bound

Sketch of Proof

We consider the dependency bubble $B(R)$ formed around any path $R = \{v_{jjj}, \ldots v_{kkk}\}$, where each entry v_{iii} corresponds to the divide operation used to compute l_{ii}. We see that $|B(R)| = \Omega(|R|)$ vertices, for $\eta(b) = b^3$, which are v_{acd} for $a, c, d \in [j, k]$. Each such bubble has a smallest separator size of $E(R) = \Omega(\epsilon(|R|))$ where $\epsilon(b) = b^2$. By application of the Bubble Expansion Theorem, we then get that for any b

$$F = \Omega(b^2 \cdot n), \qquad W = \Omega(b \cdot n), \qquad S = \Omega(n/b)$$

therefore

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

Krylov subspace methods

Definition (Krylov subspace methods)

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

Conjecture

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

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

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

Latency lower bound for s-step methods

Sketch of Proof

For n-by-n A based on a d dimensional mesh, we consider the path $P = \{x_{n/2}, (Ax)_{n/2}, \dots (A^k x)_{n/2}\}$. *The bubble* $B(R)$ *formed along a subsequence of length* $|R|$ *of this path is of size* $|B(R)| = \Omega(\sigma(|R|))$, *where* $\sigma(b) = b^{d+1}$ *(it is all vertices within* $b/2$ *hops in the mesh) and has bubble expansion* $E(R) = \Omega(\epsilon(|R|))$, *where* $\epsilon(b) = \Omega(b^d)$ *(corresponding to a vertex separator cut plane). Using the Bubble Expansion Theorem, we attain,*

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

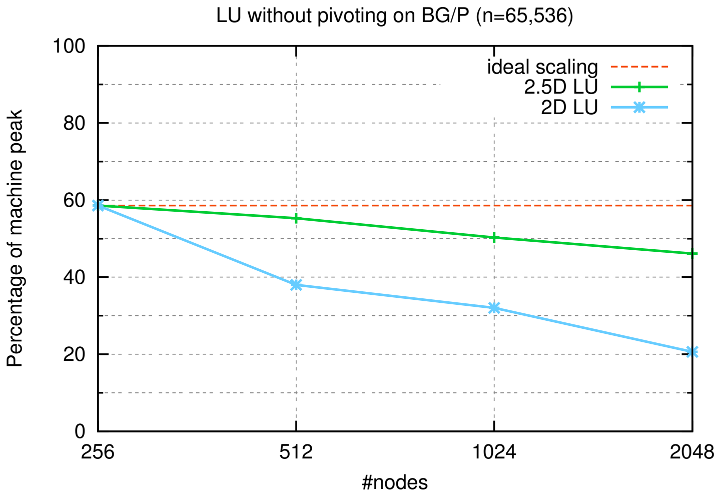
for any b.

3D recursive non-pivoted LU and Cholesky

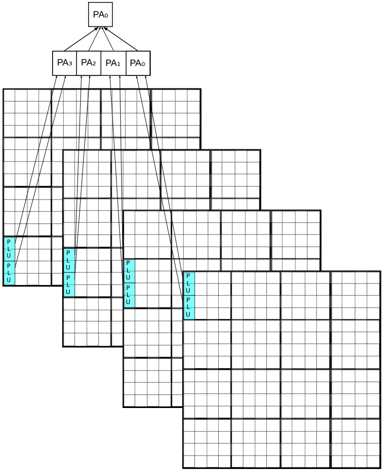
A 3D recursive algorithm with no pivoting [A. Tiskin 2002]

- 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

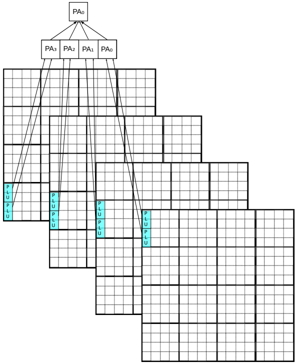
2.5D LU strong scaling (without pivoting)

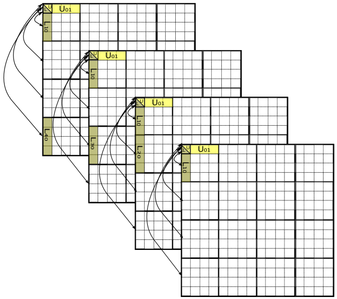


2.5D LU factorization with tournament pivoting

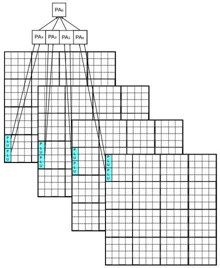


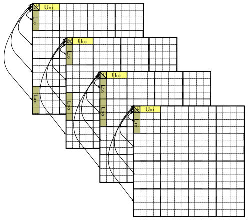
2.5D LU factorization with tournament pivoting

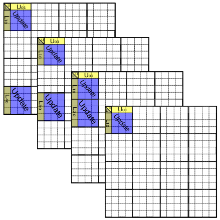




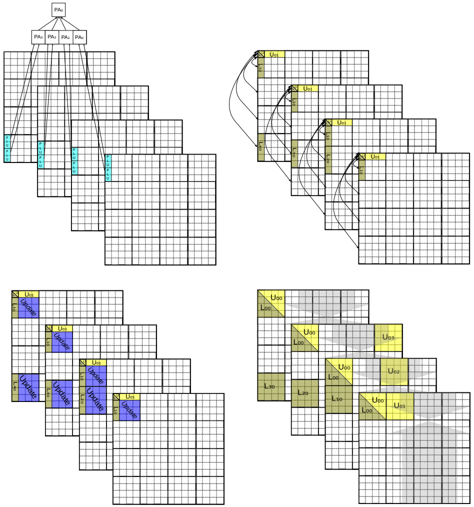
2.5D LU factorization with tournament pivoting







2.5D LU factorization with tournament pivoting



3D QR factorization

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

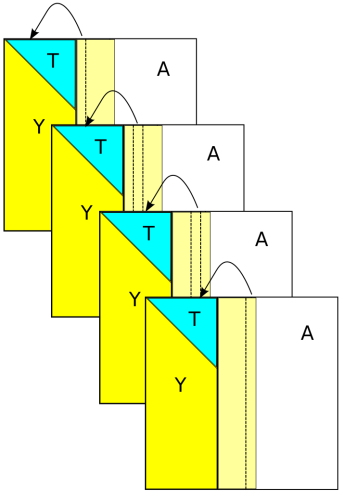
- 3D QR using Givens rotations (generic pairwise elimination) is given by [A. Tiskin 2007]
- Tiskin minimizes latency and bandwidth by working on slanted panels
- 3D QR cannot be done with right-looking updates as 2.5D LU due to non-commutativity of orthogonalization updates

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

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

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.

Cyclic decomposition in CTF

Cyclical distribution is fundamental to CTF, hence the name Cyclops (cyclic-operations).

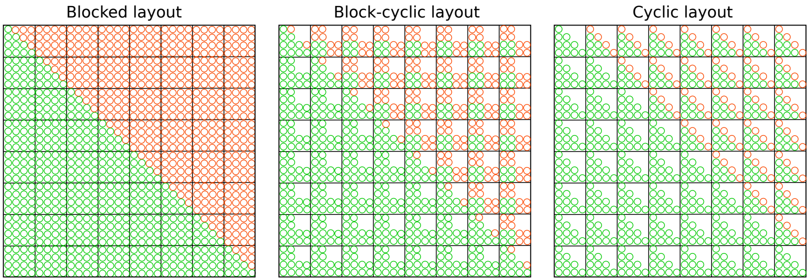
Given a vector **v** of length n on p processors

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

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.

Blocked vs block-cyclic vs cyclic decompositions



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

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

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'

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

$$
\begin{aligned}
\tilde{W}^{mn}_{ei} &= v^{mn}_{ei} + \sum_f v^{mn}_{ef} t^f_i , \\
\tilde{W}^{mn}_{ij} &= v^{mn}_{ij} + P^i_j \sum_e v^{mn}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{mn}_{ef} \tau^{ef}_{ij} , \\
\tilde{W}^{am}_{ie} &= v^{am}_{ie} - \sum_n \tilde{W}^{mn}_{ei} t^a_n + \sum_f v^{ma}_{ef} t^f_i + \frac{1}{2} \sum_{nf} v^{mn}_{ef} t^{af}_{in} , \\
\tilde{W}^{am}_{ij} &= v^{am}_{ij} + P^i_j \sum_e v^{am}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{am}_{ef} \tau^{ef}_{ij} , \\
z^a_i &= f^a_i - \sum_m \tilde{F}^m_i t^a_m + \sum_e f^a_e t^e_i + \sum_{em} v^{ma}_{ei} t^e_m + \sum_{em} v^{ae}_{im} \tilde{F}^m_e + \frac{1}{2} \sum_{efm} v \\
z^{ab}_{ij} &= v^{ab}_{ij} + P^i_j \sum_e v^{ab}_{ie} t^e_j + P^a_b P^i_j \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{mj} - P^a_b \sum_m \tilde{W}^{am}_{ij} t^b_m + P^a_b
\end{aligned}
$$

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
v-orbitals, *o*-electrons

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