Scalable Numerical Algorithms for Electronic Structure Calculations

Edgar Solomonik, James Demmel

Department of EECS, UC Berkeley

March 2013
Outline

1 Electronic structure calculations
   - Density Functional Theory
   - Coupled Cluster

2 Tensor contractions
   - Matrix multiplication
   - High-dimensional tensor contractions

3 Matrix factorizations
   - Communication lower bounds
   - 2.5D algorithms
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 Bhavele, Erik Draeger, Todd Gamblin, Martin Schulz
Quantum many-body methods
- allow study of chemical problems at the quantum level
- Attempt to find approximate solutions to the Schrödinger equation

$$H |\psi\rangle = E |\psi\rangle$$

- Density Functional Theory
  - The wave-function $\Psi$ is modelled implicitly via a particle density
- Coupled Cluster
  - The wave-function $\Psi$ is explicitly approximated via expansion and truncation of an exponential
Density Function Theory (DFT)

DFT uses the fact that the ground-state wave-function $\Psi_0$ is a unique functional of the particle density $n(\vec{r})$

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

Since $\hat{U} = \hat{T} + \hat{V} + \hat{U}$, 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,

$$V_s(\vec{r}) = V(\vec{r}) + \int \frac{e^2 n_s(\vec{r}')}{|\vec{r} - \vec{r}'|} d^3r' + V_{XC}[n_s(\vec{r})]$$
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 $\phi_i$.
3. Compute a new guess at $n(\vec{r})$ based on $\phi_i$.

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

$$\mathbf{H}\vert\psi\rangle = E\vert\psi\rangle,$$

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

$$\vert\psi\rangle = e^{\hat{T}}\vert\Phi_0\rangle$$

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 derivation

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

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

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

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

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

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

which simplifies to

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

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

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

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

\[
0 = \langle \Phi_i^a | \vec{H} | \Phi_0 \rangle = f_{ai} - \sum_{kc} f_{kc} t_i^c t_k^a + \ldots
\]

\[
0 = \langle \Phi_{ij}^{ab} | \vec{H} | \Phi_0 \rangle = \langle ab || ij \rangle + \sum_{bj} \langle ja || bi \rangle t_j^b + \ldots
\]

These equations then need to be factorized into two-tensor contractions.
For square $n$-by-$n$ matrix $A$, $B$, and $C$, compute

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$
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 \( \mathbf{A}, \mathbf{B}, \text{ and } \mathbf{C}. \) on a \( \sqrt{p} \)-by-\( \sqrt{p} \) processor grid. They have a cost of
\[
W_{2D} = O \left( \frac{n^2}{\sqrt{p}} \right)
\]

'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-\( p^{1/3} \)-by-\( p^{1/3} \) processor grid, yielding
\[
W_{3D} = O \left( \frac{n^2}{p^{2/3}} \right)
\]

'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 \( M = cn^2/p \),
\[
W_{2.5D} = O \left( \frac{n^2}{\sqrt{cp}} \right)
\]
2.5D matrix multiplication

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

$O\left(\frac{n^3}{p}\right)$ flops
$O\left(\frac{n^2}{\sqrt{c \cdot p}}\right)$ words moved
$O\left(\frac{\sqrt{p}}{c^3}\right)$ messages
$O\left(c \cdot \frac{n^2}{p}\right)$ bytes of memory
Strong scaling matrix multiplication

2.5D MM on BG/P (n=65,536)

- 2.5D SUMMA
- 2D SUMMA
- ScaLAPACK PDGEMM

Percentage of machine peak vs. #nodes

Edgar Solomonik  Communication-avoiding parallel algorithms  15/ 35
Matrix multiplication on 16,384 nodes of BG/P

- 2.5D MM
- 2D MM

Using 16 matrix copies

- 12X faster
- 2.7X faster
2.5D matrix multiplication on BG/Q

![Graph showing BG/Q matrix multiplication with two lines representing Cyclops TF and Scalapack. The x-axis represents the number of nodes (256, 512, 1024, 2048, 4096, 8192, 16384), and the y-axis represents Teraflops (8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096). The graph illustrates the performance of matrix multiplication on BG/Q with different numbers of nodes.]
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 \((ab\ldots)\). We now might face contractions like

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

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 \( g_{(ij)kl(pq)r} \neq g_{(ij)kl(pq)r} \)
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.
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

$$W = \Omega \left( \frac{G}{s_p} \right) \quad S = \Omega \left( \frac{G}{s_p} \right).$$
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

Node (1,1)  Node (1,2)  Node (1,3)  Node (1,4)

Node (2,1)  Node (2,2)  Node (2,3)  Node (2,4)
CCSD code using our domain specific language

```
FVO["me"] = VABIJ["efmn"]*T1["fn"];  
FVO["ae"] = -0.5*VABIJ["fenn"]*T2["fann"];  
FVO["ae"] = FVO["me"]*T1["am"];  
FVO["ae"] += VABC["efan"]*T1["fn"];  
F00["ml"] = 0.5*VABIJ["efnn"]*T2["efnl"];  
F00["ml"] = FVO["me"]*T1["et"];  
F00["ml"] += VIJK["nnlf"]*T1["fn"];  
WMNJ["mni"] = VIJKL["mni"];  
WMNJ["mni"] += 0.5*VABIJ["efmn"]*Tau["efij"];  
WMNJ["mni"] += VIJK["mnie"]*T1["ej"];  
WMNIE["mnie"] = VIJK["nie"];  
WMNIE["mnie"] = VABIJ["mnie"]*T1["ft"];  
WMNIE["mnie"] += VABIJ["fenn"]*T1["ef"];  
WAMJ["amj"] = VIJK["jina"];  
WAMJ["amj"] += 0.5*VABC["efam"]*Tau["eflj"];  
WAMJ["amj"] += VABIJ["amej"]*T1["et"];  
WMMT["mael"] = -VABIJ["anel"]*T1["fin"];  
WMAE["mael"] += 0.5*VABIJ["efnn"]*T2["afin"];  
WMAE["mael"] += VABIJ["feam"]*T1["efi"];  
WMAE["mael"] = WMNIE["mnie"]*T1["an"];  
Z1["ai"] = 0.5*VABC["efam"]*Tau["efin"];  
Z1["ai"] -= 0.5*WMNIE["mnie"]*T2["aemn"];  
Z1["ai"] += T2["aelm"]*FVO["me"];  
Z1["ai"] -= T1["em"]*VABIJ["anel"];  
Z1["ai"] -= T1["am"]*F00["mi"];  
Z2["ablj"] = VABIJ["ablj"];  
Z2["ablj"] += FV["af"]*T2["fbij"];  
Z2["ablj"] -= F00["nl"]*T2["abnj"];  
Z2["ablj"] += VABC["abej"]*T1["et"];  
Z2["ablj"] -= WAMJ["mbij"]*T1["an"];  
Z2["ablj"] += 0.5*VABCD["abef"]*Tau["efij"];  
Z2["ablj"] += 0.5*WMNIE["mnie"]*Tau["abmn"];  
Z2["ablj"] += WMAE["mael"]*T2["ebmj"];  
E1["ai"] = Z1["ai"]*D1["ai"];  
E2["ablj"] = Z2["ablj"]*E2["ablj"];  
E1["ai"] -= T1["ai"];  
E2["ablj"] -= T2["ablj"];  
T1["ai"] += E1["ai"];  
T2["ablj"] += E2["ablj"];  

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

E_CCSD = 0.25*scalar(VABIJ["efnn"]*Tau["efnn"])  

```
Comparison with NWChem on Cray XE6

CCSD iteration time on 64 nodes of Hopper:

<table>
<thead>
<tr>
<th>system</th>
<th># electrons</th>
<th># orbitals</th>
<th>CTF</th>
<th>NWChem</th>
</tr>
</thead>
<tbody>
<tr>
<td>w5</td>
<td>25</td>
<td>205</td>
<td>14 sec</td>
<td>36 sec</td>
</tr>
<tr>
<td>w7</td>
<td>35</td>
<td>287</td>
<td>90 sec</td>
<td>178 sec</td>
</tr>
<tr>
<td>w9</td>
<td>45</td>
<td>369</td>
<td>127 sec</td>
<td>-</td>
</tr>
<tr>
<td>w12</td>
<td>60</td>
<td>492</td>
<td>336 sec</td>
<td>-</td>
</tr>
</tbody>
</table>

On 128 nodes, NWChem completed w9 in 223 sec, CTF in 73 sec.
Blue Gene/Q up to 1250 orbitals, 250 electrons

CCSD weak scaling on Mira (BG/Q)

Teraflops vs. #nodes

Edgar Solomonik
Communication-avoiding parallel algorithms
CCSD weak scaling on Mira (BG/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]

\[
W = \Omega \left( \frac{n^3}{p \cdot M^{1/2}} \right) \quad S = \Omega \left( \frac{n^3}{p \cdot M^{3/2}} \right).
\]
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!
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, $c \in [1, p^{1/3}]$, $M = O(c \cdot n^2 / p)$

<table>
<thead>
<tr>
<th>problem</th>
<th>lower bound</th>
<th>2.5D lat $\sqrt{p/c^3}$</th>
<th>2.5D bw $n^2/\sqrt{pc}$</th>
<th>SCL lat $\sqrt{p}$</th>
<th>SCL bw $n^2/\sqrt{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>$W = \Omega(n^2/\sqrt{cp})$</td>
<td>$\sqrt{p/c^3}$</td>
<td>$n^2/\sqrt{pc}$</td>
<td>$\sqrt{p}$</td>
<td>$n^2/\sqrt{p}$</td>
</tr>
<tr>
<td>Cholesky</td>
<td>$W \cdot S = \Omega(n^2)$</td>
<td>$\sqrt{pc}$</td>
<td>$n^2/\sqrt{pc}$</td>
<td>$\sqrt{p}$</td>
<td>$n^2/\sqrt{p}$</td>
</tr>
<tr>
<td>LU</td>
<td>$W \cdot S = \Omega(n^2)$</td>
<td>$\sqrt{pc}$</td>
<td>$n^2/\sqrt{pc}$</td>
<td>$n$</td>
<td>$n^2/\sqrt{p}$</td>
</tr>
<tr>
<td>QR</td>
<td>$W \cdot S = \Omega(n^2)$</td>
<td>$\sqrt{pc}$</td>
<td>$n^2/\sqrt{pc}$</td>
<td>$n$</td>
<td>$n^2/\sqrt{p}$</td>
</tr>
<tr>
<td>sym eig</td>
<td>$W \cdot S = \Omega(n^2)$</td>
<td>$\sqrt{pc}$</td>
<td>$n^2/\sqrt{pc}$</td>
<td>$n$</td>
<td>$n^2/\sqrt{p}$</td>
</tr>
</tbody>
</table>
2.5D LU on 65,536 cores
2.5D LU on 65,536 cores

LU on 16,384 nodes of BG/P (n=131,072)

- NO-pivot 2D
- NO-pivot 2.5D
- CA-pivot 2D
- CA-pivot 2.5D

Time (sec)

- compute
- idle
- communication

2X faster
Websites with more information and papers

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

Memory-based lower-bounds

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)
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.
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).
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 $G$,

$$S = \Omega(|P|/b), \quad W = \Omega(\eta(b)).$$
Dependency bubbles

processor 1
processor 2
processor 3

Chain of 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.
Consider solving for \( x \) where \( L \) is lower-triangular in

\[
y_i = \sum_{j \leq i} l_{ij} \cdot x_j.
\]

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

\[
W \cdot S^2 = \Omega(n^2)
\]
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 \( S = \Omega(n/b) \), \( W = \Omega(b^2) \)

\[ W \cdot S^2 = \Omega(n^2). \]
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 $B(R)$. 
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

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

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

- Processor 1
- Processor 2
- Processor 3

Chain of bubbles
Proof of general latency lower bound

Definition

A parallelization corresponds to a coloring of the vertices, Let \( V = \bigcup V_i \) 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 \( R_i = P - \bigcup_{j=1}^{i-1} R_j \), so that

- some process \( p_i \in \{1, \ldots, p\} \) computes the first entry of \( R_i \)
- process \( p_j \) 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 $B(R_i)$,

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

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

$$W \cdot S = \Omega(n^2) \text{ and } F \cdot S^2 = \Omega(n^3).$$
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 $b$

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

therefore

$$W \cdot S = \Omega(n^2) \quad \text{and} \quad F \cdot S^2 = \Omega(n^3)$$
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 $P = \{x_{n/2}, (Ax)_{n/2}, \ldots (A^k x)_{n/2}\}$. 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 $b$. 

Edgar Solomonik
Communication-avoiding parallel algorithms 51/35
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
Coupled Cluster formalism

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
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
To reduce latency, we can employ the TSQR algorithm

1. Given \( n \)-by-\( b \) panel partition into \( 2b \)-by-\( b \) blocks
2. Perform QR on each \( 2b \)-by-\( b \) block
3. Stack computed \( R \)s into \( n/2 \)-by-\( b \) panel and 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,

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

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
  \[ \{v_i \cdot n/p + 1, \ldots, v(i+1) \cdot n/p\} \]
- 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.
 Blocked vs block-cyclic vs cyclic decompositions

- **Blocked layout**
  - Red denotes padding / load imbalance
  - Green denotes fill (unique values)

- **Block-cyclic layout**
  - Red denotes padding / load imbalance
  - Green denotes fill (unique values)

- **Cyclic layout**
  - Red denotes padding / load imbalance
  - Green denotes fill (unique values)
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
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

\[ \tau_{ij}^{ab} = t_{ij}^{ab} + \frac{1}{2} P_b^a P_i^j t_i^a t_j^b, \]

\[ \tilde{F}_m^e = f_e^m + \sum_{fn} \nu_{ef}^m t_n^f, \]

\[ \tilde{F}_e^a = (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} \nu_{ef}^m t_{mn}^a f_n + \sum_{fn} \nu_{ef}^a t_n^f, \]

\[ \tilde{F}_i^m = (1 - \delta_{mi}) f_i^m + \sum_e \tilde{F}_e^m t_i^e + \frac{1}{2} \sum_{nef} \nu_{ef}^m t_{in}^e f_n + \sum_{fn} \nu_{if}^m t_n^f, \]
Our CCSD factorization

\[
\tilde{W}_{ei}^{mn} = v_{ei}^{mn} + \sum_{f} v_{ef}^{mn} t_{i}^{f},
\]

\[
\tilde{W}_{ij}^{mn} = v_{ij}^{mn} + P_{j}^{i} \sum_{e} v_{ie}^{mn} t_{j}^{e} + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef},
\]

\[
\tilde{W}_{ie}^{am} = v_{ie}^{am} - \sum_{n} \tilde{W}_{ei}^{mn} t_{n}^{a} + \sum_{f} v_{ef}^{ma} t_{i}^{f} + \frac{1}{2} \sum_{nf} v_{ef}^{mn} t_{in}^{af},
\]

\[
\tilde{W}_{ij}^{am} = v_{ij}^{am} + P_{j}^{i} \sum_{e} v_{ie}^{am} t_{j}^{e} + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef},
\]

\[
z_{i}^{a} = f_{i}^{a} - \sum_{m} \tilde{F}_{i}^{m} t_{m}^{a} + \sum_{e} f_{e}^{a} t_{i}^{e} + \sum_{em} v_{ei}^{ma} t_{m}^{e} + \sum_{em} v_{im}^{ae} \tilde{F}_{e}^{m} + \frac{1}{2} \sum_{efm} v_{ef}^{pm},
\]

\[
z_{ij}^{ab} = v_{ij}^{ab} + P_{j}^{i} \sum_{e} v_{ie}^{ab} t_{j}^{e} + P_{b}^{a} P_{j}^{i} \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_{b} \sum_{m} \tilde{W}_{ij}^{am} t_{m}^{b} + P_{a}^{b} t_{j}^{b},
\]
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

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