What commonalities exist in simulation and data analysis applications?

- multidimensional datasets (observations, discretizations)
- higher-order relations: equations, maps, graphs, hypergraphs
- **sparsity** and **symmetry** in structure of relations
- relations lead to solution directly or as an iterative criterion
- algebraic descriptions of datasets and relations
What abstractions are needed in high performance computing?

- data abstractions reflecting native dimensionality and structure
- functions orchestrating communication and synchronization
- provably efficient building-block algorithms
Tensors are convenient abstractions for multidimensional data
- one type of object for any homogeneous dataset
- enable expression of symmetries
- reveal sparsity structure of relations in multidimensional space

Tensor computations naturally extend numerical linear algebra
- often reduce to or employ matrix algorithms
  - can leverage high performance matrix libraries
- high-order tensors can ‘act’ as many matrix unfoldings
- symmetries lower memory footprint and cost
- tensor factorizations (CP, Tucker, tensor train, ...)

Matrix computations $\subset$ tensor computations
Applications of high-order tensor representations

Numerical solution to differential equations
- higher-order Taylor series expansion terms
- nonlinear terms and differential operators

Computer vision and graphics
- 2D image ⊗ angle ⊗ time
- compression (tensor factorizations, sparsity)

Machine learning
- sparse multi-feature discrete datasets
- reduced-order models (tensor factorizations)

Graph computations
- hypergraphs, time-dependent graphs
- clustering/partitioning/path-finding (eigenvector computations)

Divide-and-conquer algorithms representable by tensor folding
- bitonic sort, FFT, scans
Applications to quantum systems

Manybody Schrödinger equation

- “curse of dimensionality” – exponential state space

Condensed matter physics

- tensor network models (e.g. DMRG), tensor per lattice site
- highly symmetric multilinear tensor representation
- exponential state space localized → factorized tensor form

Quantum chemistry (electronic structure calculations)

- models of molecular structure and chemical reactions
- methods for calculating electronic correlation:
  - “Post Hartree-Fock”: configuration interaction, coupled cluster, Møller-Plesset perturbation theory
- multi-electron states as tensors, e.g. electron ⊗ electron ⊗ orbital ⊗ orbital
- nonlinear equations of partially (anti)symmetric tensors
- interactions diminish with distance → sparsity, low rank
Outline and highlights

1. Symmetry-preserving tensor algorithms
   - contraction of order $2s$ symmetric tensors in $\frac{(3s)!}{(s!)^3}$ fewer multiplies
   - up to 9X speed-up for partially-symmetric contractions in coupled cluster

2. Communication-avoiding parallel algorithms
   - novel tradeoffs: synchronization vs communication in Cholesky and stencils
   - algorithms with $p^{1/6}$ less communication on $p$ processors for LU, QR, eigs
   - topology-aware implementations: 12X speed-up for MM, 2X for LU

3. Cyclops Tensor Framework (CTF)
   - first distributed-memory tensor framework supporting arbitrary contractions
   - symmetry, sparsity, multitype functions, redistributions, high-level language

4. Applications to electronic structure calculations
   - codes using CTF for wavefunction methods: Aquarius, QChem, VASP, Psi4
   - coupled cluster faster than NWChem by $>10$X, nearly 1 petaflop/s
Exploiting symmetry in tensors

Tensor symmetry (e.g. $A_{ij} = A_{ji}$) reduces memory and cost\(^1\)

- for order $d$ tensor, $d!$ less memory
- dot product $\sum_{i,j} A_{ij} B_{ij} = 2 \sum_{i<j} A_{ij} B_{ij} + \sum_i A_{ii} B_{ii}$
- matrix-vector multiplication ($A_{ij} = A_{ji}$)\(^1\)

$$c_i = \sum_j A_{ij} b_j = \sum_j A_{ij} (b_i + b_j) - \left( \sum_j A_{ij} \right) b_i$$

- $A_{ij} b_j \neq A_{ji} b_i$ but $A_{ij} (b_i + b_j) = A_{ji} (b_j + b_i) \rightarrow (1/2) n^2$ multiplies
- partially-symmetric case: $A_{ij}^{km} = A_{ji}^{km}$

$$c_i^{kl} = \sum_{j,m} A_{ij}^{km} b_j^{ml} = \sum_j \left( \sum_m A_{ij}^{km} (b_i^{ml} + b_j^{ml}) \right) - \sum_m \left( \sum_j A_{ij}^{km} \right) b_i^{ml}$$

- let $Z_{ij}^{kl} = \sum_m A_{ij}^{km} (b_i^{ml} + b_j^{ml})$ and observe $Z_{ij}^{kl} = Z_{ji}^{kl}$
- $Z_{ij}^{kl}$ can be computed using $(1/2) n^5$ multiplies and $(1/2) n^5$ adds

\(^1\) S., Demmel; Technical Report, ETH Zurich, 2015.
Symmetry preserving algorithms

By exploiting symmetry, reduce multiplies (but increase adds)\(^2\)

- rank-2 vector outer product

\[
C_{ij} = a_i b_j + a_j b_i = (a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j
\]

- squaring a symmetric matrix \(A\) (or \(AB + BA\))

\[
C_{ij} = \sum_k A_{ik} A_{kj} = \sum_k (A_{ik} + A_{kj} + A_{ij})^2 - \ldots
\]

- for symmetrized contraction of symmetric order \(s + v\) and \(v + t\) tensors

\[
\frac{(s + t + v)!}{s! t! v!} \text{ fewer multiplies}
\]

e.g. cases above are

- \(s = 1, t = 1, v = 0 \rightarrow\) reduction by 2X
- \(s = 1, t = 1, v = 1 \rightarrow\) reduction by 6X

\(^2\) S., Demmel; Technical Report, ETH Zurich, 2015.
Applications of symmetry preserving algorithms

Extensions and applications:

- algorithms generalize to antisymmetric and Hermitian tensors
- cost reductions in partially-symmetric coupled cluster contractions: 2X-9X for select contractions, 1.3X-2.1X for methods
- for Hermitian tensors, multiplies cost 3X more than adds
  - Hermitian matrix multiplication and tridiagonal reduction (BLAS and LAPACK routines) with 25% fewer operations
- \((2/3)n^3\) bilinear rank for squaring a nonsymmetric matrix
- decompose symmetric contractions into smaller symmetric contractions

Further directions:

- high performance implementation
- symmetry in tensor equations (e.g. Cholesky factors)
- generalization to other group actions
- relationships to fast matrix multiplication and structured matrices
Algorithms should minimize communication, not just computation

- data movement and synchronization cost more energy than flops
- two types of data movement:
  - vertical (intranode memory–cache)
  - horizontal (internode network transfers)
- parallel algorithm design involves tradeoffs: computation vs communication vs synchronization
- lower bounds and parameterized algorithms provide optimal solutions within a well-defined tuning space
Cost model for parallel algorithms

Given a schedule of work and communication tasks on $p$ processors, consider the following costs, accumulated along chains of tasks (as in $\alpha - \beta$, BSP, and LogGP models),

- $F$ – computation cost
- $Q$ – vertical communication cost
- $W$ – horizontal communication cost
- $S$ – synchronization cost
Communication lower bounds: previous work

Multiplication of $n \times n$ matrices

- horizontal communication lower bound$^3$
  \[ W_{\text{MM}} = \Omega \left( \frac{n^2}{p^{2/3}} \right) \]

- memory-dependent horizontal communication lower bound$^4$
  \[ W_{\text{MM}} = \Omega \left( \frac{n^3}{p \sqrt{M}} \right) \]

  with $M = cn^2/p$ memory, hope to obtain communication cost
  \[ W = O(n^2/\sqrt{cp}) \]

- libraries like ScaLAPACK, Elemental optimal only for $c = 1$

---

$^3$ Aggarwal, Chandra, Snir, TCS, 1990

$^4$ Irony, Toledo, Tiskin, JPDC, 2004
Communication-efficient matrix multiplication

Communication-avoiding algorithms for matrix multiplication have been studied extensively\(^5\)

They continue to be attractive on modern architectures\(^6\)

12X speed-up, 95% reduction in comm. for \(n = 8K\) on 16K nodes of BG/P

---

\(^5\) Berntsen, Par. Comp., 1989; Agarwal, Chandra, Snir, TCS, 1990; Agarwal, Balle, Gustavson, Joshi, Palkar, IBM, 1995; McColl, Tiskin, Algorithmica, 1999; ...

\(^6\) S., Bhavele, Demmel, SC, 2011
Unlike matrix multiplication, many algorithms in numerical linear algebra have polynomial depth (contain a long dependency path)

- matrix multiplication synchronization cost bound

\[ S_{MM} = \Theta \left( \frac{n^3}{pM^{3/2}} \right) \]

- algorithms for Cholesky, LU, QR, SVD do not attain this bound
- low granularity computation increases synchronization cost

---

Ballard, Demmel, Holtz, Schwartz, SIAM JMAA, 2011
Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the $n \times n$ diamond DAG,

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

We generalize this idea

- additionally consider horizontal communication
- allow arbitrary (polynomial or exponential) interval expansion

---

8 Papadimitriou, Ullman, SIAM JC, 1987
9 S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)
Tradeoffs involving synchronization

We apply tradeoff lower bounds to dense linear algebra algorithms, represented via dependency hypergraphs:\(^a\)

For triangular solve with an \( n \times n \) matrix,

\[
F_{\text{TRSV}} \cdot S_{\text{TRSV}} = \Omega \left( n^2 \right)
\]

For Cholesky of an \( n \times n \) matrix,

\[
F_{\text{CHOL}} \cdot S_{\text{CHOL}}^2 = \Omega \left( n^3 \right) \quad W_{\text{CHOL}} \cdot S_{\text{CHOL}} = \Omega \left( n^2 \right)
\]

Proof employs classical Loomis-Whitney inequality:

for any \( R \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \), three projections of \( R \) onto \( \mathbb{N} \times \mathbb{N} \) have total size at least \( |R|^{2/3} \)

\(^a\)S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)
For any \( c \in [1, p^{1/3}] \), use \( cn^2/p \) memory per processor and obtain

\[
W_{LU} = O(n^2/\sqrt{cp}), \quad S_{LU} = O(\sqrt{cp})
\]

- LU with pairwise pivoting\(^{10}\) extended to tournament pivoting\(^{11}\)
- first implementation of a communication-optimal LU algorithm\(^{11}\)

---

\(^{10}\) Tiskin, FGCS, 2007  
\(^{11}\) S., Demmel, Euro-Par, 2011
Communication-efficient QR factorization

- $W_{QR} = O(n^2/\sqrt{cp})$, $S_{QR} = O(\sqrt{cp})$ using Givens rotations$^a$
- Householder form can be reconstructed quickly from TSQR$^b$
  \[ Q = I - YTY^T \quad \Rightarrow \quad LU(I - Q) \rightarrow (Y, TY^T) \]
- enables communication-optimal Householder QR$^c$
- Householder aggregation yields performance improvements

Further directions: 2.5D QR implementation, lower bounds, pivoting

---

$^a$ Tiskin, FGCS, 2007

$^b$ Ballard, Demmel, Grigori, Jacquelin, Nguyen, S., IPDPS, 2014

$^c$ S., UCB, 2014
For the dense symmetric matrix eigenvalue problem\textsuperscript{a}

\[ W_{SE} = O(n^2/\sqrt{cp}), S_{QR} = O(\sqrt{cp} \log^2 p) \]

\textbullet above costs obtained by left-looking algorithm with Householder aggregation, however, with increased vertical communication

\textbullet successive band reduction minimizes both communication costs

Further directions: implementations (ongoing), eigenvector computation, SVD

\textsuperscript{a}S., UCB, 2014. S., Hoefler, Demmel, in preparation
Our lower bound analysis extends to sparse iterative methods:\textsuperscript{12}

For computing $s$ applications of a $(2m + 1)^d$-point stencil

\[
F_{St} \cdot S_{St}^d = \Omega \left( m^{2d} \cdot s^{d+1} \right) \\
W_{St} \cdot S_{St}^{d-1} = \Omega \left( m^d \cdot s^d \right)
\]

- proof requires generalization of Loomis-Whitney inequality to order $d$
  - set and order $d - 1$ projections
- time-blocking lowers synchronization and vertical communication costs, but raises horizontal communication
- we suggest alternative approach that minimizes vertical and horizontal communication, but not synchronization
- further directions:
  - implementation of proposed algorithm
  - lower bounds for graph traversals

\textsuperscript{12} S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)
Loomis-Whitney inequalities are not sufficient for all computations

- symmetry preserving tensor contraction algorithms have arbitrary order projections from order $d$ set
- bilinear algorithms\textsuperscript{13} provide a more general framework
- a bilinear algorithm is defined by matrices $F^{(A)}$, $F^{(B)}$, $F^{(C)}$,

$$c = F^{(C)}[(F^{(A)^T}a) \odot (F^{(B)^T}b)]$$

where $\odot$ is the Hadamard (pointwise) product

$$c = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}^T \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \odot \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}^T \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$$

- communication lower bounds derived based on matrix rank\textsuperscript{14}

\textsuperscript{13} Pan, Springer, 1984
\textsuperscript{14} S., Hoefler, Demmel, in preparation
For contraction of order $s + v$ tensor with order $v + t$ tensor\textsuperscript{15}

- symmetry preserving algorithm requires $\frac{(s+v+t)!}{s!v!t!}$ fewer multiplies
- matrix-vector-like algorithms ($\min(s, v, t) = 0$)
  - vertical communication dominated by largest tensor
  - horizontal communication asymptotically greater if only unique elements are stored and $s \neq v \neq t$
- matrix-matrix-like algorithms ($\min(s, v, t) > 0$)
  - vertical and horizontal communication costs asymptotically greater for symmetry preserving algorithm when $s \neq v \neq t$
- further work: bounds for nested and iterative bilinear algorithms

\textsuperscript{15}S., Hoefler, Demmel; Technical Report, ETH Zurich, 2015.
Tensor algebra as a programming abstraction

Cyclops Tensor Framework\(^{16}\)

- contraction/summation/functions of tensors
- distributed symmetric-packed/sparse storage via cyclic layout
- parallelization via MPI+OpenMP(+CUDA)

Cyclops Tensor Framework

- contraction/summation/functions of tensors
- distributed symmetric-packed/sparse storage via cyclic layout
- parallelization via MPI+OpenMP(+CUDA)

Jacobi iteration (solves $Ax = b$ iteratively) example code snippet

```cpp
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){
    ... // split $A = R + \text{diag}(1/d)$
    do {
        x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
        r["i"] = b["i"]-A["ij"]*x["j"]; // compute residual
    } while (r.norm2() > 1.E-6); // check for convergence
    return x;
}
```
Tensor algebra as a programming abstraction

Cyclops Tensor Framework

- contraction/summation/functions of tensors
- distributed symmetric-packed/sparse storage via cyclic layout
- parallelization via MPI+OpenMP(+CUDA)

Jacobi iteration (solves $Ax = b$ iteratively) example code snippet

```cpp
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){
    Matrix<> R(A);
    R["ii"] = 0.0;
    Vector<> x(n), d(n), r(n);
    Function<> inv([](double & d){ return 1./d; });
    d["i"] = inv(A["ii"]); // set d to inverse of diagonal of A
    do {
        x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
        r["i"] = b["i"]-A["ij"]*x["j"]; // compute residual
    } while (r.norm2() > 1.E-6); // check for convergence
    return x;
}
```
Tensor algebra as a programming abstraction

Cyclops Tensor Framework

- contraction/summation/functions of tensors
- distributed symmetric-packed/sparse storage via cyclic layout
- parallelization via MPI+OpenMP(+CUDA)

Møller-Plesset perturbation theory (MP3) code snippet

```plaintext
Z["abij"] += Fab["af"]*T["fbij"];
Z["abij"] -= Fij["ni"]*T["abnj"];  
Z["abij"] += 0.5*Vabcd["abef"]*T["efij"]; 
Z["abij"] += 0.5*Vijkl["mnij"]*T["abmn"]; 
Z["abij"] -= Vaibj["amei"]*T["ebmj"]; 
```
Betweenness centrality code snippet, for $k$ of $n$ nodes

```cpp
void btwn_central(Matrix<int> A, Matrix<path> P, int n, int k)
{
    Monoid<path> mon(...,
        [](path a, path b){
            if (a.w<b.w) return a;
            else if (b.w<a.w) return b;
            else return path(a.w, a.m+b.m);
        }, ...);

    Matrix<path> Q(n,k,mon); // shortest path matrix
    Q["ij"] = P["ij"];

    Function<int,path> append([](int w, path p){
        return path(w+p.w, p.m);
    });

    for (int i=0; i<n; i++)
        Q["ij"] = append(A["ik"],Q["kj"]);

    ...
}
```
Performance of CTF for dense computations

CTF is highly tuned for massively-parallel machines

- virtualized multidimensional processor grids
- topology-aware mapping and collective communication
- performance-model-driven decomposition done at runtime
- optimized redistribution kernels for tensor transposition

![Diagram of CTF architecture](Image)

![Graph showing performance comparison](Image)

Edgar Solomonik
Algorithms as Multilinear Tensor Equations 26/33
Performance of CTF for sparse computations

MP3 leveraging sparse-dense tensor contractions\(^a\)

Weak scaling of MP3 (m=40, n=160 on 24 cores)

Strong scaling of MP3 with m=40, n=160

All-pairs shortest-paths based on path doubling with sparsification\(^a\)

Weak scaling of APSP (n=2K on 24 cores)

Strong scaling of APSP with n=2K

\(^a\)S., Hoefler, Demmel, arXiv, 2015
Coupled cluster methods

Coupled cluster provides a systematically improvable approximation to the many-body time-independent Schrödinger equation $H|\psi\rangle = E|\psi\rangle$

- the Hamiltonian has one- and two-electron components $H = F + V$
- Hartree-Fock (SCF) computes mean-field Hamiltonian: $F$, $V$
- Coupled-cluster methods (CCSD, CCSDT, CCSDTQ) consider transitions of (doubles, triples, and quadruples) of electrons to unoccupied orbitals, encoded by tensor operator, $T = T_1 + T_2 + T_3 + T_4$
- they use an exponential ansatz for the wavefunction, $\Psi = e^{T} \phi$
  where $\phi$ is a Slater determinant
- expanding $0 = \langle \phi' | H | \Psi \rangle$ yields nonlinear equations for $\{T_i\}$ in $F, V$

\[
0 = V_{ij}^{ab} + P(a, b) \sum_e T_{ij}^{ae} F_{e}^{b} - \frac{1}{2} P(i, j) \sum_{mnef} T_{im}^{ab} V_{ef}^{mn} T_{jn}^{ef} + \ldots
\]

where $P$ is an antisymmetrization operator
CCSD using CTF

Extracted from Aquarius (Devin Matthews’ code, https://github.com/devinamatthews/aquarius)

```plaintext
FMI["mi"] += 0.5*WMNEF["mnef"]*T2["efin"];
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T2["efij"];
FAE["ae"] -= 0.5*WMNEF["mnef"]*T2["afmn"];
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T2["afin"];
Z2["abij"] = WMNEF["ijab"];  
Z2["abij"] += FAE["af"]*T2["fbij"];  
Z2["abij"] -= FMI["ni"]*T2["abnj"];  
Z2["abij"] += 0.5*WABEF["abef"]*T2["efij"];  
Z2["abij"] += 0.5*WMNIJ["mnij"]*T2["abmn"];  
Z2["abij"] -= WAMEI["amei"]*T2["ebmj"];  
```

CTF-based CCSD codes exist in Aquarius, QChem, VASP, and Psi4
Comparison with NWChem

NWChem is the most commonly-used distributed-memory quantum chemistry method suite

- provides CCSD and CCSDT
- derives equations via Tensor Contraction Engine (TCE)
- generates contractions as blocked loops leveraging Global Arrays
CCSD up to 55 (50) water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ

Weak scaling on BlueGene/Q
Aquarius-CTF CCSD
Aquarius-CTF CCSDT

Weak scaling on Edison
Aquarius-CTF CCSD
Aquarius-CTF CCSDT

Gigaflops/node
#nodes
Weak scaling on BlueGene/Q
Weak scaling on Edison
Novel results described in this talk:

- **symmetry preserving algorithms**
  - reduce number of multiplications in symmetric contractions by $\omega$!
  - reduce cost of basic Hermitian matrix operations by 25%
  - reduce cost of some contractions in coupled cluster by 2X in CCSD (1.3X overall), 4X in CCSDT (2.1X overall), 9X in CCSDTQ

- **communication and synchronization lower bounds**
  - tradeoffs: synchronization vs computation or communication in TRSV, Cholesky, and stencils
  - rank-based lower bounds to analyze symmetric contractions

- **communication-avoiding matrix factorizations**
  - new algorithms and implementations with up to $p^{1/6}$ less communication for LU, QR, symmetric eigenvalue problem
  - speed-ups of up to 2X for LU and QR over vendor-optimized libraries

- **Cyclops Tensor Framework**
  - first fully robust distributed-memory tensor contraction library
  - supports symmetry, sparsity, general algebraic structures
  - coupled cluster performance more than 10X faster than state-of-the-art, reaching 1 petaflop/s performance
Impact and future work

- symmetry in tensor computations
  - cost improvements $\rightarrow$ fast library implementations $\rightarrow$ application speed-ups
  - study symmetries in tensor equations and factorizations
  - consider other symmetries and relation to fast matrix multiplication

- communication-avoiding algorithms
  - existing fast implementations already used by applications (e.g. QBox)
  - find efficient methods of searching larger tuning spaces
  - algorithms for computing eigenvectors, SVD, tensor factorizations
  - study (randomized) algorithms for sparse matrix factorization

- Cyclops Tensor Framework
  - already widely-adapted in quantum chemistry, many requests for features
  - study algorithms for tensor expressions $\rightarrow$ factorization, scheduling, ...
  - engage new application domains (via sparsity and algebraic structures)
    - tensor networks for condensed matter-physics, particle methods
    - graph algorithms, discrete data analysis
    - graphics, computer vision, machine learning
Symmetry preserving algorithm vs Strassen’s algorithm

\[
\frac{[n^\omega/(s!t!v!)]/\text{#multiplications}}{\text{(speed-up over classical direct evaluation alg.)}}
\]

Strassen’s algorithm
Sym. preserving \(\omega=6\)
Sym. preserving \(\omega=3\)

Edgar Solomonik
Algorithms as Multilinear Tensor Equations 35/33
Nesting of bilinear algorithms

Given two bilinear algorithms:

\[ \Lambda_1 = (F_1^{(A)}, F_1^{(B)}, F_1^{(C)}) \]
\[ \Lambda_2 = (F_2^{(A)}, F_2^{(B)}, F_2^{(C)}) \]

We can nest them by computing their tensor product

\[ \Lambda_1 \otimes \Lambda_2 := (F_1^{(A)} \otimes F_2^{(A)}, F_1^{(B)} \otimes F_2^{(B)}, F_1^{(C)} \otimes F_2^{(C)}) \]

\[ \text{rank}(\Lambda_1 \otimes \Lambda_2) = \text{rank}(\Lambda_1) \cdot \text{rank}(\Lambda_2) \]
Block-cyclic algorithm for $s$-step methods

For $s$-steps of a $(2m + 1)^d$-point stencil with block-size of $H^{1/d}/m$,

$$W_{Kr} = O \left( \frac{msn^d}{H^{1/d}p} \right) \quad S_{Kr} = O(\frac{sn^d}{(pH)}) \quad Q_{Kr} = O \left( \frac{msn^d}{H^{1/d}p} \right)$$

which are good when $H = \Theta(n^d/p)$, so the algorithm is useful when the cache size is a bit smaller than $n^d/p$.
LU factorization strong scaling on Stampede (MIC + Sandy Bridge)

- 2.5D hybrid LU $n=131,072$
- 2D hybrid LU $n=131,072$
- 2.5D pure-cpu LU $n=131,072$
- 2.5D hybrid LU $n=65,536$
- 2D hybrid LU $n=65,536$
- 2.5D pure-cpu LU $n=65,536$
LU factorization strong scaling on Mira (BG/Q), n=65,536

- 2D LU, custom mapping
- 2D LU, default mapping
Symmetric matrix representation

Symmetric matrix

Unique part of symmetric matrix
Blocked distributions of a symmetric matrix

Naive blocked layout

Block-cyclic layout
Cyclic distribution of a symmetric matrix

Cyclic layout  ~  Improved blocked layout

P1

P0

P2

P1

P3
Our CCSD factorization

Credit to John F. Stanton and Jurgen Gauss

\[ \tau_{ij}^{ab} = t_{ij}^{ab} + \frac{1}{2} P^a_i P^b_j t_i^a t_j^b, \]

\[ \tilde{F}_e^m = f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \]

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

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

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

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

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

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

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

\[- \frac{1}{2} \sum_{emn} \tilde{W}_{ei} t_{mn}^{ea},\]

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

\[+ P_{b} \sum_{e} \tilde{F}_{e}^{a} t_{ij}^{eb} - P_{j} \sum_{m} \tilde{F}_{i}^{m} t_{ij}^{ab} + \frac{1}{2} \sum_{ef} v_{ef} \tau_{ij}^{ef} + \frac{1}{2} \sum_{mn} \tilde{W}_{ij} t_{mn}^{ab},\]
Stability of symmetry preserving algorithms

Relative error of $c = A^t b$ with positive $A$ and alternating $b$

Relative error of squaring a Householder transformation
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
ν-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>$O(ν^4o^2/p)$</td>
<td>flops/mem bandwidth</td>
</tr>
<tr>
<td>broadcasts</td>
<td>20%</td>
<td>$O(ν^4o^2/p\sqrt{M})$</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>$O(ν^2o^2/p)$</td>
<td>integer ops</td>
</tr>
<tr>
<td>all-to-all-ν</td>
<td>7%</td>
<td>$O(ν^2o^2/p)$</td>
<td>bisection bandwidth</td>
</tr>
<tr>
<td>tensor folding</td>
<td>4%</td>
<td>$O(ν^2o^2/p)$</td>
<td>memory bandwidth</td>
</tr>
</tbody>
</table>
Tiskin’s path doubling algorithm

Tiskin gives a way to do path-doubling in $F = O(n^3/p)$ operations. We can partition each $A^k$ by path size (number of edges)

$$A^k = I \oplus A^k(1) \oplus A^k(2) \oplus \ldots \oplus A^k(k)$$

where each $A^k(l)$ contains the shortest paths of up to $k \geq l$ edges, which have exactly $l$ edges. We can see that

$$A^l(l) \leq A^{l+1}(l) \leq \ldots \leq A^n(l) = A^*(l),$$

in particular $A^*(l)$ corresponds to a sparse subset of $A^l(l)$. The algorithm works by picking $l \in [k/2, k]$ and computing

$$(I \oplus A)^{3k/2} \leq (I \oplus A^k(l)) \otimes A^k,$$

which finds all paths of size up to $3k/2$ by taking all paths of size exactly $l \geq k/2$ followed by all paths of size up to $k$. 