

# Algorithms as Multilinear Tensor Equations

Edgar Solomonik

Department of Computer Science  
ETH Zurich

**University of Toronto**

March 15, 2016

*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
- **algebraic descriptions of datasets and relations**

# Tensor computations as programming abstractions

Tensors (scalars, vectors, matrices, etc.) 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

Matrix computations  $\subset$  tensor computations

- = 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, ...)

# What is the power of a parallel tensor library?

The ability to **optimally** orchestrate

- *algebraic transformations*
- *data movement*
- *synchronization*

for a **universal** class of algebraic 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  $\otimes$  angle  $\otimes$  time
- compression (tensor factorizations, sparsity)

## Machine learning

- sparse multi-feature discrete datasets
- reduced-order models, recommendation systems (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  $\rightarrow$  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  $\otimes$  electron  $\otimes$  orbital  $\otimes$  orbital
- nonlinear equations of partially (anti)symmetric tensors
- interactions diminish with distance  $\rightarrow$  sparsity, low rank

- 1 Massively-parallel electronic structure calculations
  - Cyclops Tensor Framework (CTF): first distributed-memory tensor contraction framework
  - codes using CTF for wavefunction methods: Aquarius, QChem, VASP, Psi4
  - coupled cluster faster than NWChem by  $> 10X$ , nearly 1 petaflop/s
- 2 Sparse and discrete tensor computations
  - CTF supports arbitrary sparse multidimensional arrays
  - sparsity used to accelerate algebraic all-pairs shortest-paths
- 3 Communication-optimal algorithms for linear solvers
  - 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
- 4 Preserving symmetry in tensor contractions
  - 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

# Coupled cluster methods

Coupled cluster provides a systematically improvable approximation to the manybody 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  $F, V$

$$0 = V_{ij}^{ab} + \mathcal{P}(a, b) \sum_e T_{ij}^{ae} F_e^b - \frac{1}{2} \mathcal{P}(i, j) \sum_{mnef} T_{im}^{ab} V_{ef}^{mn} T_{jn}^{ef} + \dots$$

where  $\mathcal{P}$  is an antisymmetrization operator



# A library for tensor computations

## Cyclops Tensor Framework<sup>1</sup>

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

---

<sup>1</sup>S., Hammond, Demmel, UCB, 2011. S., Matthews, Hammond, Demmel, IPDPS, 2013

# A library for tensor computations

## 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

```
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){
    ... // split A = R + 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;
}
```

# A library for tensor computations

## Cyclops Tensor Framework

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

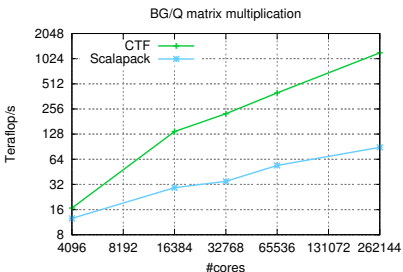
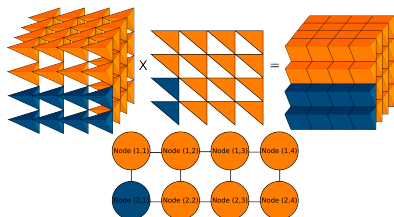
Jacobi iteration (solves  $Ax = b$  iteratively) example code snippet

```
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;
}
```

# 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



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

```
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"];

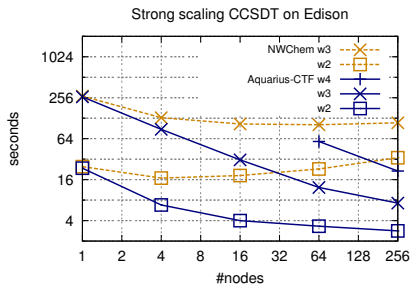
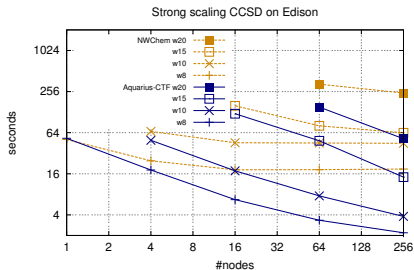
Z2["abij"]    = WMNEF["ijab"];
Z2["abij"]    += FAE["af"]*T2["fbij"];
Z2["abij"]    -= FMI["ni"]*T2["abnj"];
Z2["abij"]    += 0.5*WABEF["abef"]*T2["efij"];
Z2["abij"]    += 0.5*WMNIJ["mnij"]*T2["abmn"];
Z2["abij"]    -= WAMEI["amei"]*T2["ebmj"];
```

CTF is used within **Aquarius**, **QChem**, **VASP**, and **Psi4**

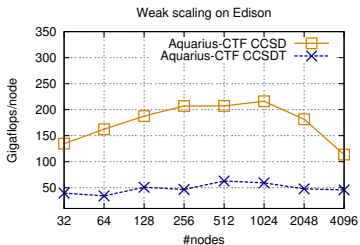
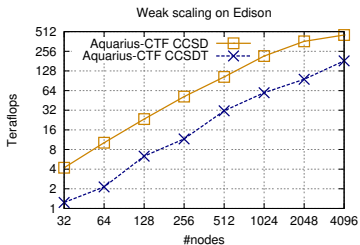
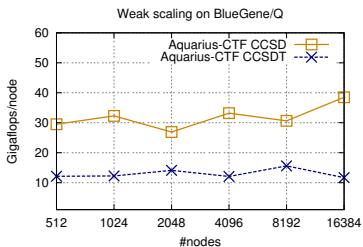
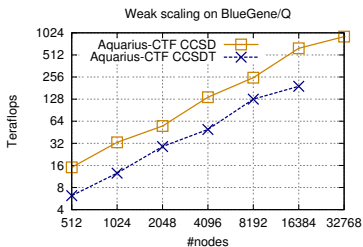
# Comparison with NWChem

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<sup>a</sup>



<sup>a</sup>S., Matthews, Hammond, Demmel, JPDC, 2014

Møller-Plesset perturbation theory (MP3) code snippet

```
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"];
```

Consider sparse two-electron integrals: **Vabcd**, **Vijkl**, **Vaibj**



# Algebraic shortest path computations

Tropical (geodetic) semiring

- additive (idempotent) operator:  $a \oplus b := \min(a, b)$ , identity:  $\infty$
- multiplicative operator:  $a \otimes b := a + b$ , identity:  $0$
- matrix multiplication defined accordingly,

$$C = A \otimes B \quad := \quad \forall i, j, C_{ij} = \min_k (A_{ik} + B_{kj})$$

Bellman-Ford algorithm (SSSP) for  $n \times n$  adjacency matrix  $A$ :

- 1 initialize  $v^{(1)} = (0, \infty, \infty, \dots)$
- 2 compute  $v^{(n)}$  via recurrence

$$v^{(i+1)} = v^{(i)} \oplus (v^{(i)} \otimes A)$$

# Algebraic shortest path computations

All-pairs shortest-paths (APSP):

- distance matrix is the closure of  $A$ ,

$$A^* = I \oplus A \oplus A^2 \oplus \dots \oplus A^n$$

- Floyd–Warshall = Gauss–Jordan elimination  $\approx$  Gaussian elimination
  - $O(n^3)$  cost, but contains length  $n \log n$  dependency path<sup>2</sup>
- path doubling:  $\log n$  steps,  $O(n^3 \log n)$  cost:

$$B = I \oplus A, \quad B^{2^k} = B^k \otimes B^k, \quad B^n = A^*$$

- sparse path doubling<sup>3</sup>:
  - let  $C$  be subset of  $B^k$  corresponding to paths containing *exactly*  $k$  edges,

$$B^{2^k} = B^k \oplus (C \otimes B^k)$$

- $O(n^3)$  cost, dependency paths length  $O(\log^2 n)$

---

<sup>2</sup>S., Buluc, Demmel, IPDPS, 2013

<sup>3</sup>Tiskin, Springer LNCS, 2001

# Bellman–Ford Algorithm using CTF

CTF code for  $n$  node single-source shortest-paths (SSSP) calculation:

```
World w(MPI_COMM_WORLD);
Semiring<int> s(INT_MAX/2,
               [](int a, int b){ return min(a,b); },
               MPI_MIN,
               0,
               [](int a, int b){ return a+b; });

Matrix<int> A(n,n,SP,w,s); // Adjacency matrix
Vector<int> v(n,w,s); // Distances from starting vertex

... // Initialize A and v

//Bellman-Ford SSSP algorithm
for (int t=0; t<n; t++){
    v["i"] += v["j"]*A["ji"];
}
```

# Betweenness centrality

Betweenness centrality code snippet, for  $k$  of  $n$  nodes

```
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);
        }, ...);

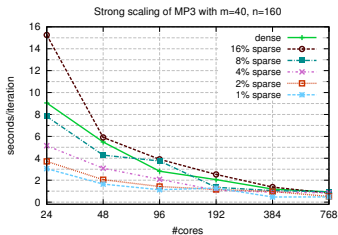
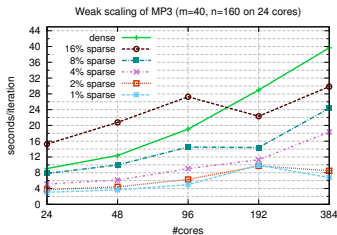
    Matrix<path> Q(n,k,mon); // shortest path matrix
    Q["ij"] = P["ij"];

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

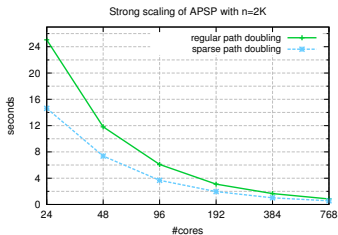
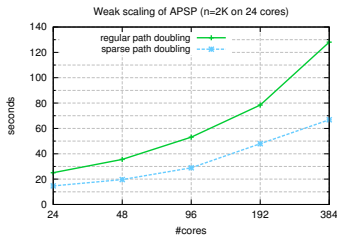
    for (int i=0; i<n; i++)
        Q["ij"] = append(A["ik"],Q["kj"]);
    ...
}
```

# Performance of CTF for sparse computations

## MP3 leveraging sparse-dense tensor contractions<sup>a</sup>



## All-pairs shortest-paths based on path doubling with sparsification<sup>a</sup>



<sup>a</sup>S., Hoefler, Demmel, arXiv, 2015

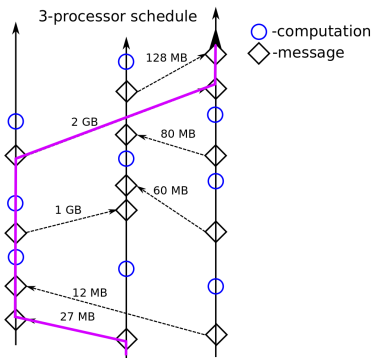
# Cost model for parallel algorithms

Algorithms should minimize communication, not just computation

- data movement and synchronization cost more energy than flops

Given a schedule 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<sup>4</sup>

$$W_{MM} = \Omega\left(\frac{n^2}{p^{2/3}}\right)$$

- memory-dependent horizontal communication lower bound<sup>5</sup>

$$W_{MM} = \Omega\left(\frac{n^3}{p\sqrt{M}}\right)$$

- with  $M = cn^2/p$  memory, hope to obtain communication cost

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

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

---

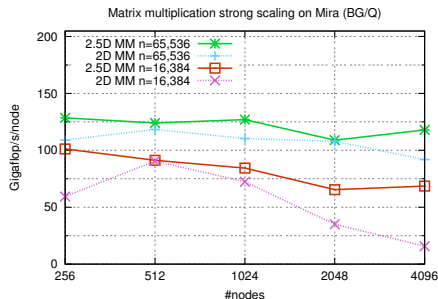
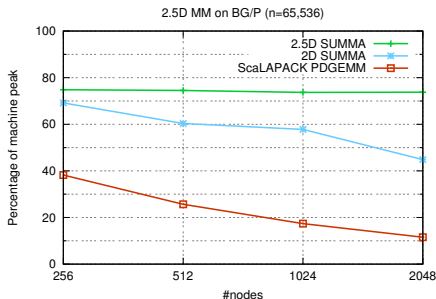
<sup>4</sup> Aggarwal, Chandra, Snir, TCS, 1990

<sup>5</sup> Irony, Toledo, Tiskin, JPDC, 2004

# Communication-efficient matrix multiplication

Communication-avoiding algorithms for matrix multiplication have been studied extensively<sup>6</sup>

They continue to be attractive on modern architectures<sup>7</sup>



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

<sup>6</sup> Berntsen, Par. Comp., 1989; Agarwal, Chandra, Snir, TCS, 1990; Agarwal, Balle, Gustavson, Joshi, Palkar, IBM, 1995; McColl, Tiskin, Algorithmica, 1999; ...

<sup>7</sup> S., Bhatele, Demmel, SC, 2011



# Synchronization cost lower bounds

Unlike matrix multiplication, many algorithms in numerical linear algebra have polynomial depth (contain a long dependency path)

- matrix multiplication synchronization cost bound<sup>8</sup>

$$S_{\text{MM}} = \Theta \left( \frac{n^3}{\rho M^{3/2}} \right)$$

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

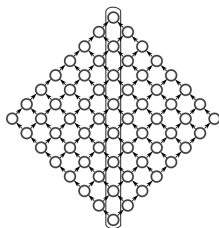
---

<sup>8</sup>Ballard, Demmel, Holtz, Schwartz, SIAM JMAA, 2011

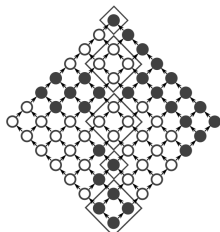
# Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the  $n \times n$  diamond DAG,<sup>9</sup>

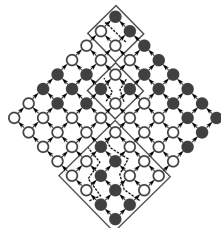
$$F \cdot S = \Omega(n^2)$$



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

We generalize this idea<sup>10</sup>

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

<sup>9</sup>Papadimitriou, Ullman, SIAM JC, 1987

<sup>10</sup>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:<sup>a</sup>

For triangular solve with an  $n \times n$  matrix,

$$F_{\text{TRSV}} \cdot S_{\text{TRSV}} = \Omega(n^2)$$

For Cholesky of an  $n \times n$  matrix,

$$F_{\text{CHOL}} \cdot S_{\text{CHOL}}^2 = \Omega(n^3) \quad W_{\text{CHOL}} \cdot S_{\text{CHOL}} = \Omega(n^2)$$

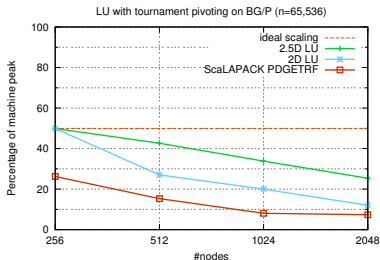
---

<sup>a</sup>S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)

# Communication-efficient LU factorization

For any  $c \in [1, p^{1/3}]$ , use  $cn^2/p$  memory per processor and obtain

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



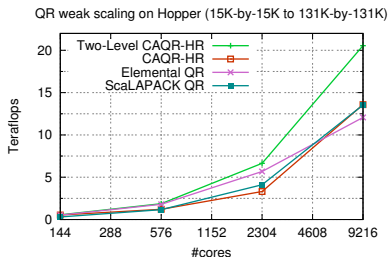
- LU with pairwise pivoting<sup>11</sup> extended to tournament pivoting<sup>12</sup>
- first implementation of a communication-optimal LU algorithm<sup>12</sup>

<sup>11</sup>Tiskin, FGCS, 2007

<sup>12</sup>S., Demmel, Euro-Par, 2011

# Communication-efficient QR factorization

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



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

<sup>a</sup>Tiskin, FGCS, 2007

<sup>b</sup>Ballard, Demmel, Grigori, Jacquelin, Nguyen, S., IPDPS, 2014

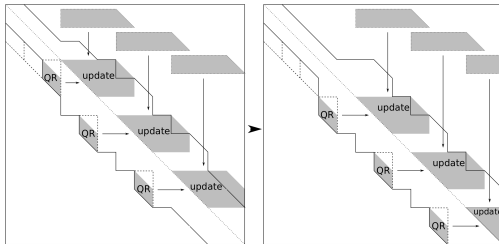
<sup>c</sup>S., UCB, 2014

# Communication-efficient eigenvalue computation

For the dense symmetric matrix eigenvalue problem<sup>a</sup>

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

- above costs obtained by left-looking algorithm with Householder aggregation, however, with increased vertical communication
- successive band reduction minimizes both communication costs



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

<sup>a</sup>S., UCB, 2014. S., Hoefler, Demmel, in preparation

# Synchronization tradeoffs in stencils

Our lower bound analysis extends to sparse iterative methods:<sup>13</sup>  
For computing  $s$  applications of a  $(2m + 1)^d$ -point stencil,

$$F_{\text{St}} \cdot S_{\text{St}}^d = \Omega \left( m^{2d} \cdot s^{d+1} \right), \quad W_{\text{St}} \cdot S_{\text{St}}^{d-1} = \Omega \left( m^d \cdot s^d \right)$$

- 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

---

<sup>13</sup>S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)

# Symmetry preserving algorithms

Tensor symmetry (e.g.  $A_{ij} = A_{ji}$ ) reduces memory and cost<sup>14</sup>

- for order  $d$  tensor,  $d!$  less memory
- matrix-vector multiplication ( $A_{ij} = A_{ji}$ )<sup>1</sup>

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

- $A_{ij} b_j \neq A_{ji} b_i$  but  $A_{ij} (b_i + b_j) = A_{ji} (b_j + b_i) \rightarrow (1/2)n^2$  multiplies
- for symmetrized contraction of symmetric order  $s + v$  and  $v + t$  tensors

$$\frac{(s + t + v)!}{s!t!v!} \quad \text{fewer multiplies}$$

- lower overall cost for partially symmetric contractions
- up to 9X for select contractions, 1.3X/2.1X for CCSD/CCSDT
- Hermitian BLAS/LAPACK operations with 25% less cost
- Ongoing: relationship to fast structured matrix multiplication

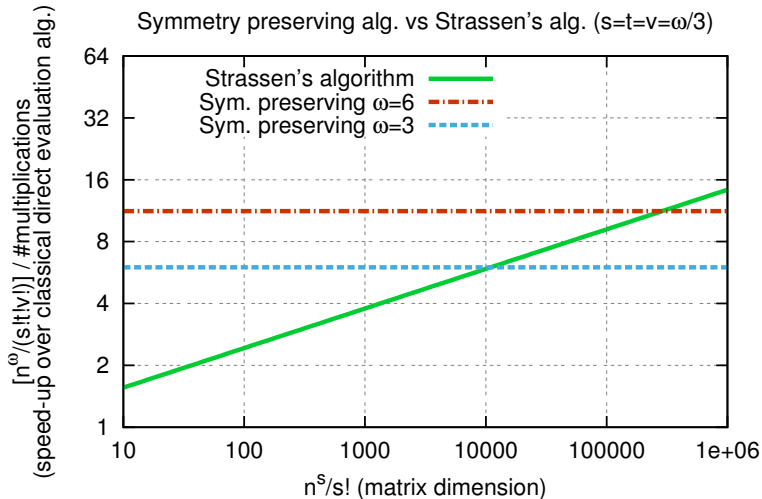
<sup>14</sup>S., Demmel; Technical Report, ETH Zurich, 2015.



- Cyclops Tensor Framework
  - already widely-adapted in quantum chemistry, many requests for features
  - study algorithms for tensor expressions → 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
- 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
- symmetry in tensor computations
  - cost improvements → fast library implementations → application speed-ups
  - study symmetries in tensor equations and factorizations
  - consider other symmetries and relation to fast matrix multiplication



# Symmetry preserving algorithm vs Strassen's algorithm



# Nesting of bilinear algorithms

Given two bilinear algorithms:

$$\Lambda_1 = (\mathbf{F}_1^{(A)}, \mathbf{F}_1^{(B)}, \mathbf{F}_1^{(C)})$$

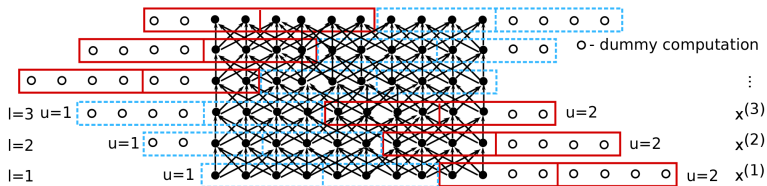
$$\Lambda_2 = (\mathbf{F}_2^{(A)}, \mathbf{F}_2^{(B)}, \mathbf{F}_2^{(C)})$$

We can nest them by computing their tensor product

$$\Lambda_1 \otimes \Lambda_2 := (\mathbf{F}_1^{(A)} \otimes \mathbf{F}_2^{(A)}, \mathbf{F}_1^{(B)} \otimes \mathbf{F}_2^{(B)}, \mathbf{F}_1^{(C)} \otimes \mathbf{F}_2^{(C)})$$

$$\text{rank}(\Lambda_1 \otimes \Lambda_2) = \text{rank}(\Lambda_1) \cdot \text{rank}(\Lambda_2)$$

# Block-cyclic algorithm for $s$ -step methods

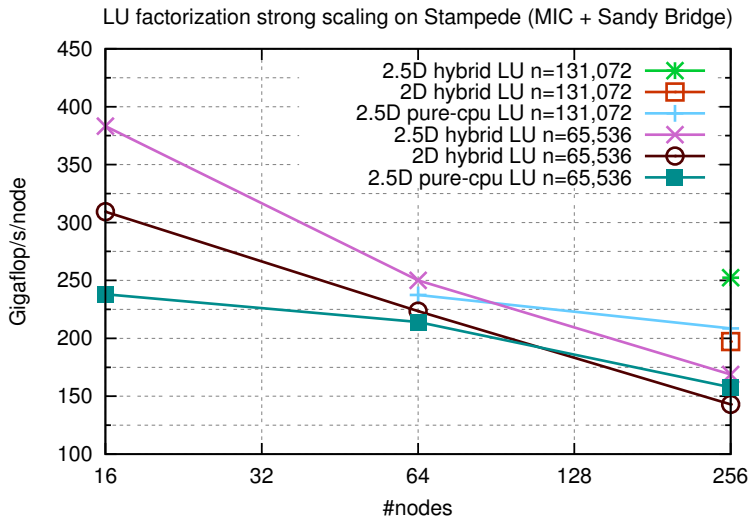


For  $s$ -steps of a  $(2m + 1)^d$ -point stencil with block-size of  $H^{1/d}/m$ ,

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

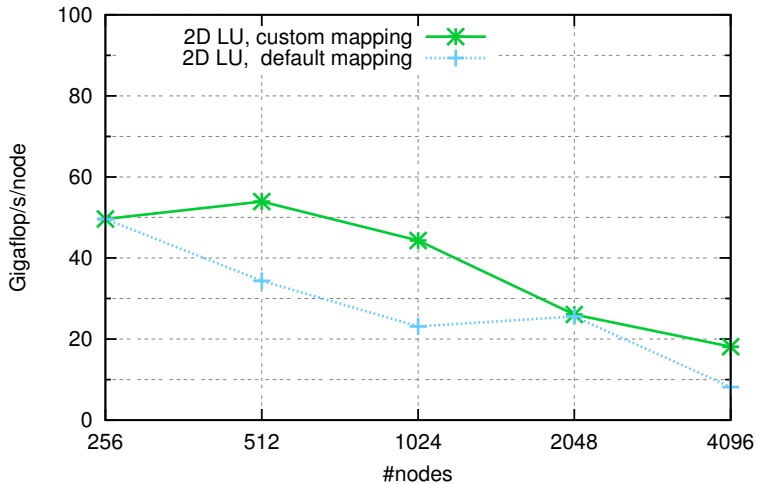
which are good when  $H = \Theta(n^d/p)$ , so the algorithm is useful when the cache size is a bit smaller than  $n^d/p$

# 2.5D LU on MIC



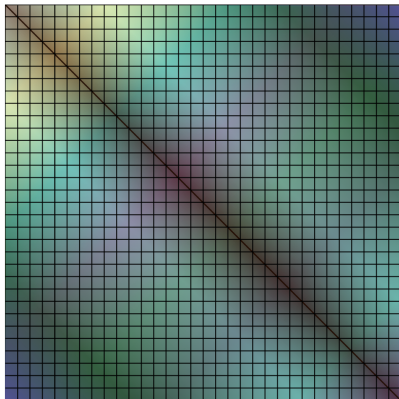
# Topology-aware mapping on BG/Q

LU factorization strong scaling on Mira (BG/Q),  $n=65,536$

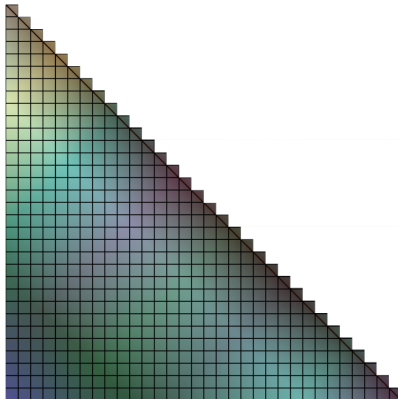


# Symmetric matrix representation

Symmetric matrix



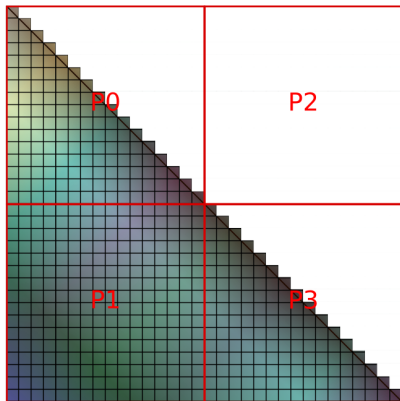
Unique part of symmetric matrix



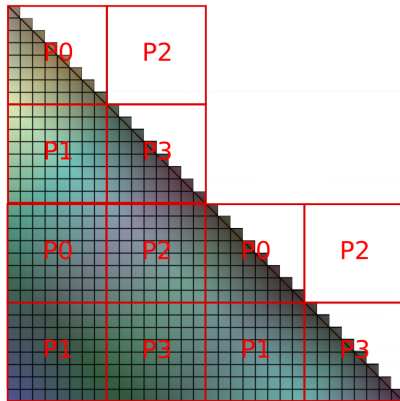


# Blocked distributions of a symmetric matrix

Naive blocked layout



Block-cyclic layout

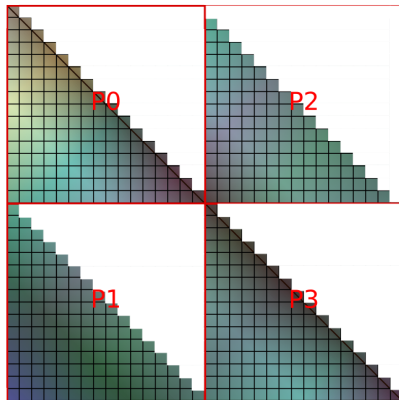
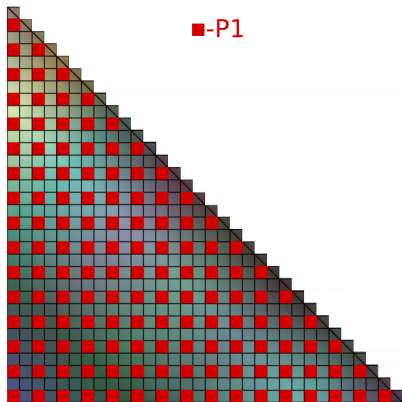


# Cyclic distribution of a symmetric matrix

Cyclic layout

~

Improved blocked layout



Credit to John F. Stanton and Jurgen Gauss

$$\tau_{ij}^{ab} = t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b,$$

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

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

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

# Our CCSD factorization

$$\tilde{W}_{ei}^{mn} = v_{ei}^{mn} + \sum_f v_{ef}^{mn} t_i^f,$$

$$\tilde{W}_{ij}^{mn} = v_{ij}^{mn} + P_j^i \sum_e v_{ie}^{mn} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef},$$

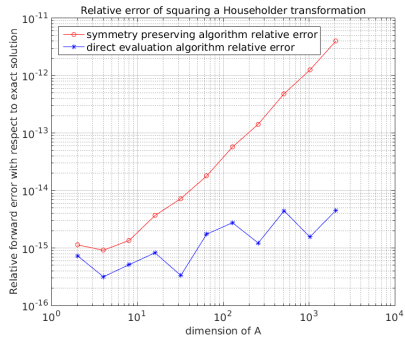
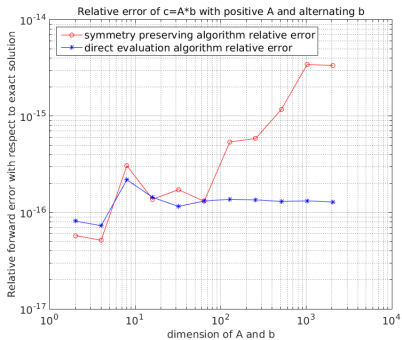
$$\tilde{W}_{ie}^{am} = v_{ie}^{am} - \sum_n \tilde{W}_{ei}^{mn} t_n^a + \sum_f v_{ef}^{ma} t_i^f + \frac{1}{2} \sum_{nf} v_{ef}^{mn} t_{in}^{af},$$

$$\tilde{W}_{ij}^{am} = v_{ij}^{am} + P_j^i \sum_e v_{ie}^{am} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef},$$

$$\begin{aligned} z_i^a &= f_i^a - \sum_m \tilde{F}_i^m t_m^a + \sum_e f_e^a t_i^e + \sum_{em} v_{ei}^{ma} t_m^e + \sum_{em} v_{im}^{ae} \tilde{F}_e^m + \frac{1}{2} \sum_{efm} v_{ef}^{am} \tau_{im}^{ef} \\ &\quad - \frac{1}{2} \sum_{emn} \tilde{W}_{ei}^{mn} t_{mn}^{ea}, \end{aligned}$$

$$\begin{aligned} z_{ij}^{ab} &= v_{ij}^{ab} + P_j^i \sum_e v_{ie}^{ab} t_j^e + P_b^a P_j^i \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_b^a \sum_m \tilde{W}_{ij}^{am} t_m^b \\ &\quad + P_b^a \sum_e \tilde{F}_e^a t_{ij}^{eb} - P_j^i \sum_m \tilde{F}_i^m t_{mj}^{ab} + \frac{1}{2} \sum_{ef} v_{ef}^{ab} \tau_{ij}^{ef} + \frac{1}{2} \sum_{mn} \tilde{W}_{ij}^{mn} \tau_{mn}^{ab}, \end{aligned}$$

# Stability of symmetry preserving algorithms



# 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%       | $O(v^4 o^2 / p)$          | flops/mem bandwidth  |
| broadcasts      | 20%       | $O(v^4 o^2 / p \sqrt{M})$ | multicast bandwidth  |
| prefix sum      | 10%       | $O(p)$                    | allreduce bandwidth  |
| data packing    | 7%        | $O(v^2 o^2 / p)$          | integer ops          |
| all-to-all- $v$ | 7%        | $O(v^2 o^2 / p)$          | bisection bandwidth  |
| tensor folding  | 4%        | $O(v^2 o^2 / p)$          | memory bandwidth     |

# Tiskin's path doubling algorithm

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

$$\mathbf{A}^k = \mathbf{I} \oplus \mathbf{A}^k(1) \oplus \mathbf{A}^k(2) \oplus \dots \oplus \mathbf{A}^k(k)$$

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

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

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

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

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  $k$ .