Provably Efficient Algorithms as Tensor Equations

Edgar Solomonik

Department of Computer Science
ETH Zurich

Cornell University

February 2, 2016

Tensors and algebraic structures

We consider the expression of data as indexable collections of elements and algorithms as applications of algebraic operators.

Definition (Algebraic structure)

A set of elements (type), potentially equipped with operators and identities

Examples: set, monoid, group, semiring, ring

Definition (Tensor)

A collection of elements of a single type, $\mathbf{T}$, with some order k and dimensions $(n_1, \ldots, n_k)$, with elements $T_{i_1 \ldots i_k}$

Examples: scalar, vector, matrix

An algebraic structure defines summation and contraction of tensors.

Numerical tensor computations

Classical matrix-based computations over the $(+, \cdot)$ ring

- stencil computations (iterative methods for sparse linear systems)

$$\mathbf{x}^{(l)} := \mathbf{A}\mathbf{x}^{(l-1)}$$

- dense matrix factorizations (direct solvers for dense linear systems)

$$\mathbf{A} \approx \mathbf{LU} \qquad \mathbf{A} \approx \mathbf{QR} \qquad \mathbf{A} \approx \mathbf{UDV}^{\mathsf{T}}$$

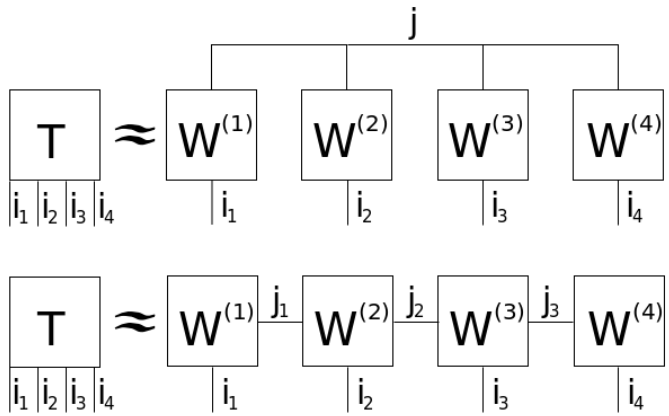
- tensor contractions (perturbation theory, solvers for nonlinear systems)

$$\sum_f F_f^a T_{ij}^{fb} - \sum_n F_i^n T_{nj}^{ab} + \frac{1}{2}\sum_{e,f} V_{ef}^{ab} T_{ij}^{ef} + \frac{1}{2}\sum_{m,n} V_{ij}^{mn} T_{mn}^{ab} - \sum_{e,m} V_{ei}^{am} T_{mj}^{eb}$$

- tensor decompositions (compression)

$$T_{i_1 \ldots i_k} \approx \sum_j W_{i_1 j}^{(1)} \cdots W_{i_k j}^{(k)}$$

$$T_{i_1 \ldots i_k} \approx \sum_{j_1 \ldots j_{k-1}} W_{i_1 j_1}^{(1)} W_{j_1 i_2 j_2}^{(2)} \cdots W_{j_{k-2} i_{k-1} j_{k-1}}^{(k-1)} W_{j_{k-1} i_k}^{(k)}$$



Discrete tensor algorithms

Alternative algebraic structures expand potential of tensor computations

- graph algorithms via tropical (geodetic) semiring $(\min, +)$
  - single-source shortest-paths via Bellman-Ford (stencil-like)
  - all-pairs shortest-paths (APSP) via Floyd-Warshall (LU-like)
  - APSP via path doubling (matrix-multiplication-like)
  - betweenness centrality
  - hypergraphs are representable by tensors
- recursion via higher order tensors
  - prefix sum, scan
  - FFT or other butterfly networks
  - bitonic sort
- particle methods
  - direct particle–particle force evaluation
  - particle–mesh (PM, P3M, SPME, PIC)

Cost model for parallel algorithms

Given a schedule that specifies all work and communication tasks on p processors, we consider the following costs, measured along dependent sequences of tasks (as in $\alpha - \beta$, BSP, and LogGP models).

Definition (F – computation cost)

Number of operations performed

Definition (Q – vertical communication cost)

Amount of data moved between memory and cache

Definition (W – horizontal communication cost)

Amount of data moved between processors

Definition (S – synchronization cost)

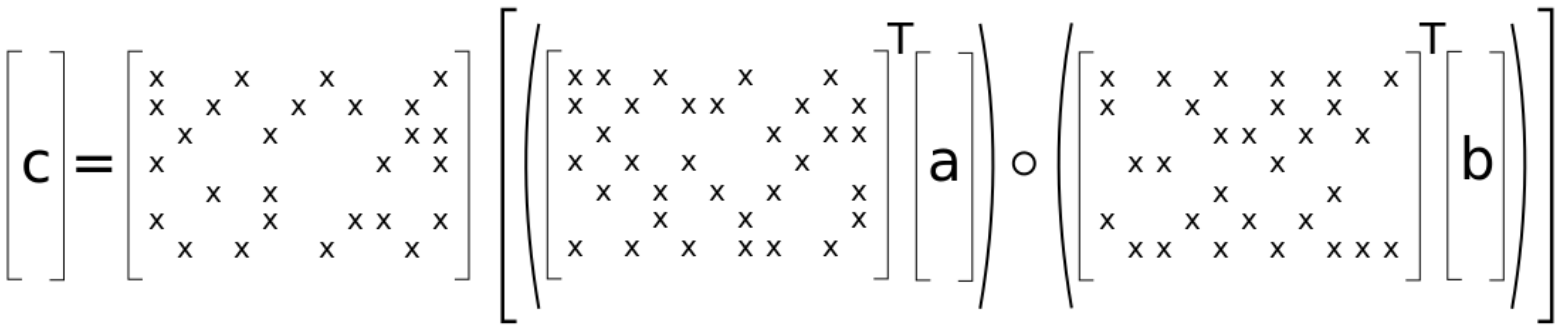
Number of distinct messages sent between processors

Bilinear algorithms

A bilinear algorithm Λ is defined by three matrices, $\Lambda = (\mathbf{F^{(A)}}, \mathbf{F^{(B)}}, \mathbf{F^{(C)}})$
Given input vectors $\mathbf{a}$ and $\mathbf{b}$, it computes vector,

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

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

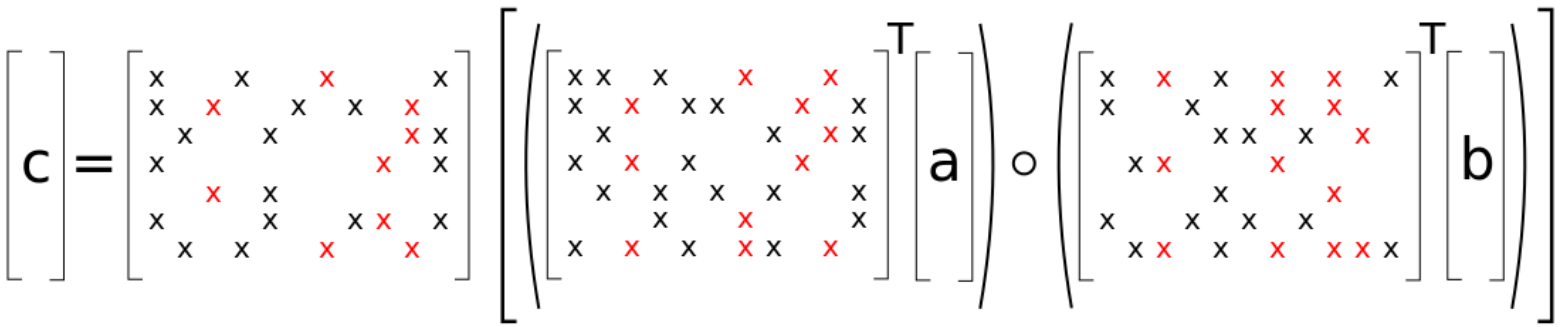


- the number of columns in the three matrices is equal and is the *bilinear algorithm rank*, denoted rank(Λ)
- the number of rows in each matrix corresponds to the number of inputs (dimensions of $\mathbf{a}$ and $\mathbf{b}$) and outputs (dimension of $\mathbf{c}$)

Bilinear algorithm expansion

A bilinear algorithm $\Lambda = (\mathbf{F}^{(\mathbf{A})}, \mathbf{F}^{(\mathbf{B})}, \mathbf{F}^{(\mathbf{C})})$ has expansion bound $\mathcal{E}_\Lambda : \mathbb{N}^3 \to \mathbb{N}$, if for all projection matrices $\mathbf{P}$,

$$\Lambda_{\text{sub}} = (\mathbf{F}^{(\mathbf{A})}\mathbf{P}, \mathbf{F}^{(\mathbf{B})}\mathbf{P}, \mathbf{F}^{(\mathbf{C})}\mathbf{P})$$



has rank bounded by $\mathcal{E}_\Lambda$,

$$\text{rank}(\Lambda_{\text{sub}}) \leq \mathcal{E}_\Lambda \left(\text{rank}(\mathbf{F}^{(\mathbf{A})}\mathbf{P}), \text{rank}(\mathbf{F}^{(\mathbf{B})}\mathbf{P}), \text{rank}(\mathbf{F}^{(\mathbf{C})}\mathbf{P}) \right)$$

Communication lower bounds

Consider any algorithm $\Lambda = (\mathbf{F^{(A)}}, \mathbf{F^{(B)}}, \mathbf{F^{(C)}})$ and expansion bound $\mathcal{E}_\Lambda$.
For a cache size H, Λ requires total vertical communication cost,

$$Q \geq \left\lceil 2H \frac{\text{rank}(\Lambda)}{\mathcal{E}_\Lambda^{\max}(H)} \right\rceil$$

where $\mathcal{E}_\Lambda^{\max}(H) := \max_{c^{(A)}+c^{(B)}+c^{(C)}=3H} \mathcal{E}_\Lambda(c^{(A)}, c^{(B)}, c^{(C)})$.
Given p processors, Λ requires horizontal communication cost,

$$W \geq \min_{\mathcal{E}_\Lambda\left(c^{(A)}+\frac{r^{(A)}}{p}, c^{(B)}+\frac{r^{(B)}}{p}, c^{(B)}+\frac{r^{(C)}}{p}\right) \geq \frac{\text{rank}(\Lambda)}{p}} \left\lceil c^{(A)} + c^{(B)} + c^{(C)} \right\rceil$$

where $r^{(A)}$, $r^{(B)}$, and $r^{(C)}$ are the number of rows in $\mathbf{F^{(A)}}$, $\mathbf{F^{(B)}}$, and $\mathbf{F^{(C)}}$, respectively.

Dependency interval expansion

Consider a bilinear algorithm that computes a set of multiplications V with a partial ordering, we denote a dependency interval between $a, b \in V$ as

$$[a, b] = \{a, b\} \cup \{c : a < c < b, c \in V\}$$

If there exists $\{v_1, \ldots, v_n\} \in V$ with $v_i < v_{i+1}$ and $\left|[v_{i+1}, v_{i+k}]\right| = \Theta(k^d)$ for all $k \in \mathbb{N}$, then

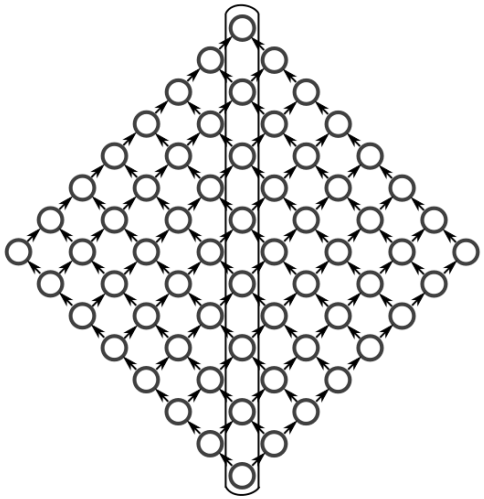
$$F \cdot S^{d-1} = \Omega(n^d)$$

where F is the computation cost and S is the synchronization cost

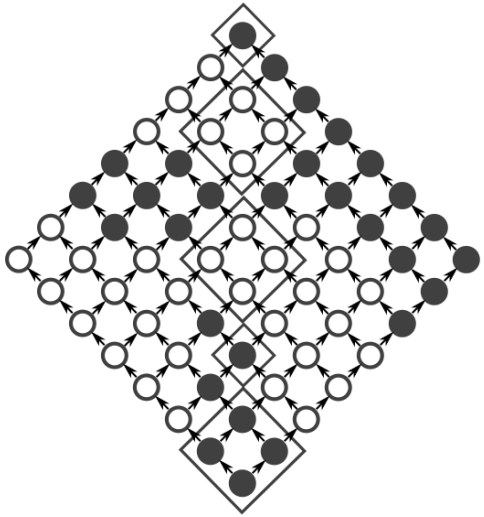
Further, if the algorithm has bilinear expansion $\mathcal{E}$, satisfying $\mathcal{E}^{\max}(H) = \Omega(H^{\frac{d}{d-1}})$, then

$$W \cdot S^{d-2} = \Omega(n^{d-1})$$

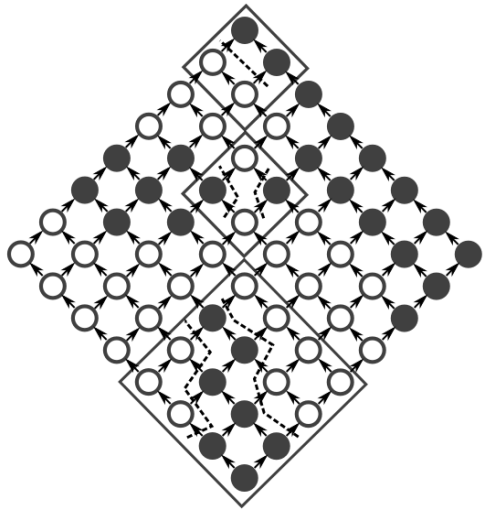
Example: diamond DAG



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

For the $n \times n$ diamond DAG ($d = 2$),

$$F \cdot S^{d-1} = F \cdot S = \Omega((n/b)b^2) \cdot \Omega(n/b) = \Omega(n^2)$$
$$W \cdot S^{d-2} = W \quad = \Omega((n/b)b) \qquad\qquad = \Omega(n)$$

idea of $F \cdot S$ tradeoff goes back to Papadimitriou and Ullman, 1987

Tradeoffs involving synchronization

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

$$F_{\mathrm{TRSV}} \cdot S_{\mathrm{TRSV}} = \Omega\left(n^2\right)$$

For Cholesky of an $n \times n$ matrix

$$F_{\mathrm{CHOL}} \cdot S_{\mathrm{CHOL}}^2 = \Omega\left(n^3\right) \qquad W_{\mathrm{CHOL}} \cdot S_{\mathrm{CHOL}} = \Omega\left(n^2\right)$$

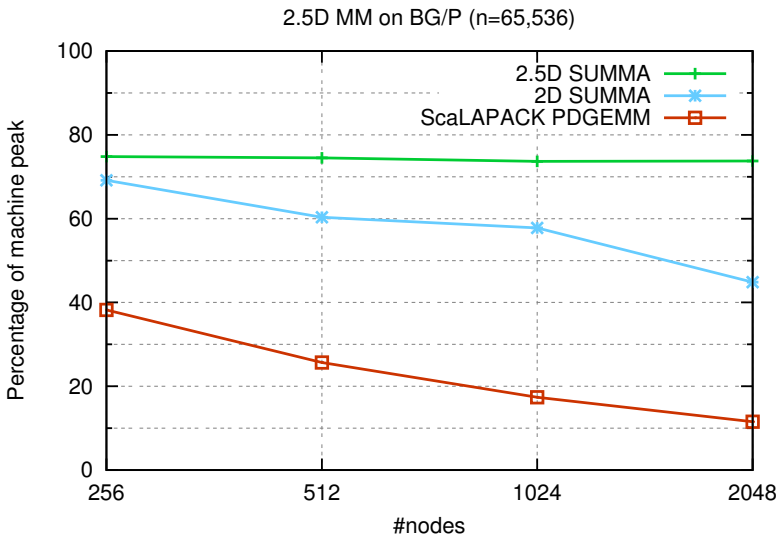
For computing s applications of a $(2m+1)^d$-point stencil

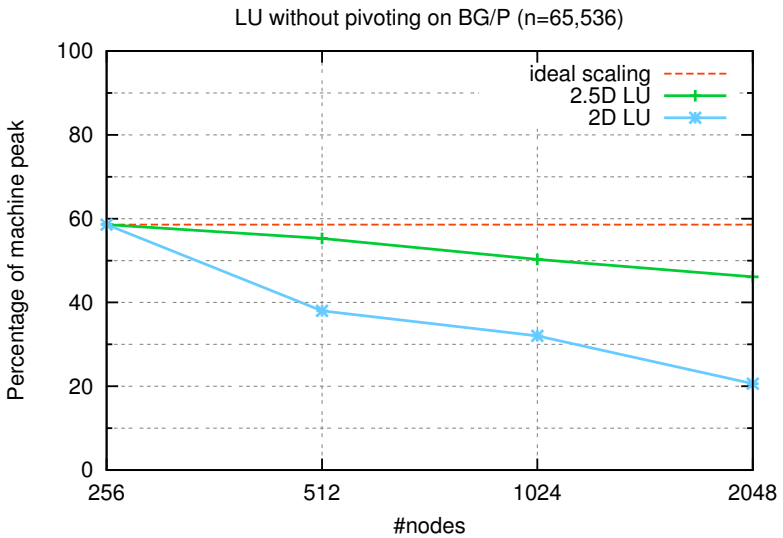
$$F_{\mathrm{St}} \cdot S_{\mathrm{St}}^d = \Omega\left(m^{2d} \cdot s^{d+1}\right) \qquad W_{\mathrm{St}} \cdot S_{\mathrm{St}}^{d-1} = \Omega\left(m^d \cdot s^d\right)$$

Communication-optimal dense matrix algorithms

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

$$W_{\mathrm{DMF}} = O(n^2/\sqrt{cp}), \qquad S_{\mathrm{DMF}} = O(\sqrt{cp})$$

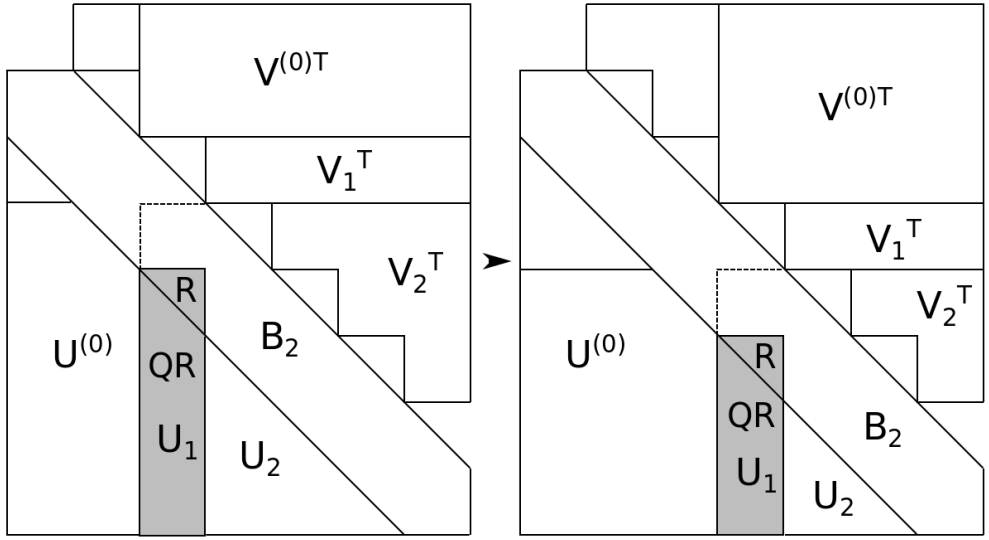




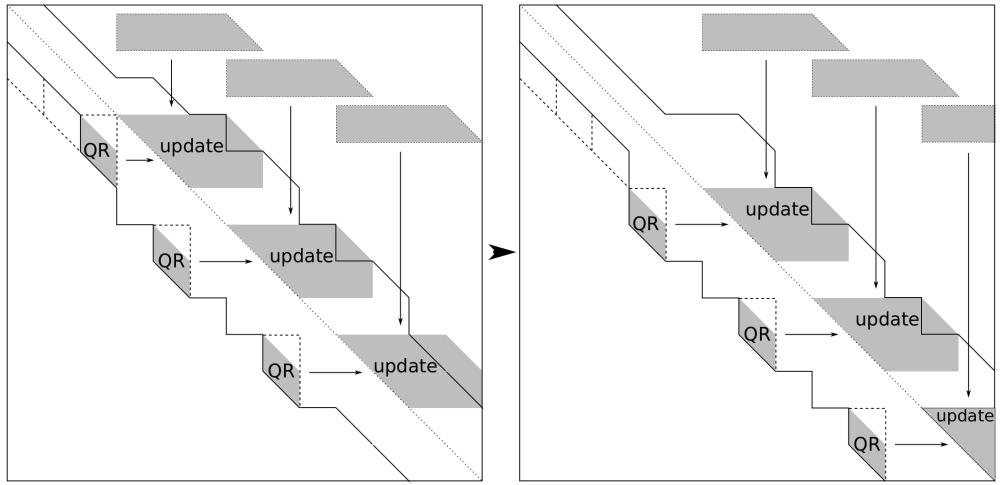
- LU with pairwise pivoting extended to tournament pivoting
- QR with Givens rotations extended to Householder transformations
- full-to-banded reduction for symmetric eigenvalue problem
- successive band reduction for symmetric eigenvalue problem

Parallel algorithms for the symmetric eigenvalue problem

Direct full-to-band reduction with $W = O(n^2/\sqrt{cp})$ communication



Successive band reduction with $W, Q = O(n^2 \log p/\sqrt{cp})$ communication

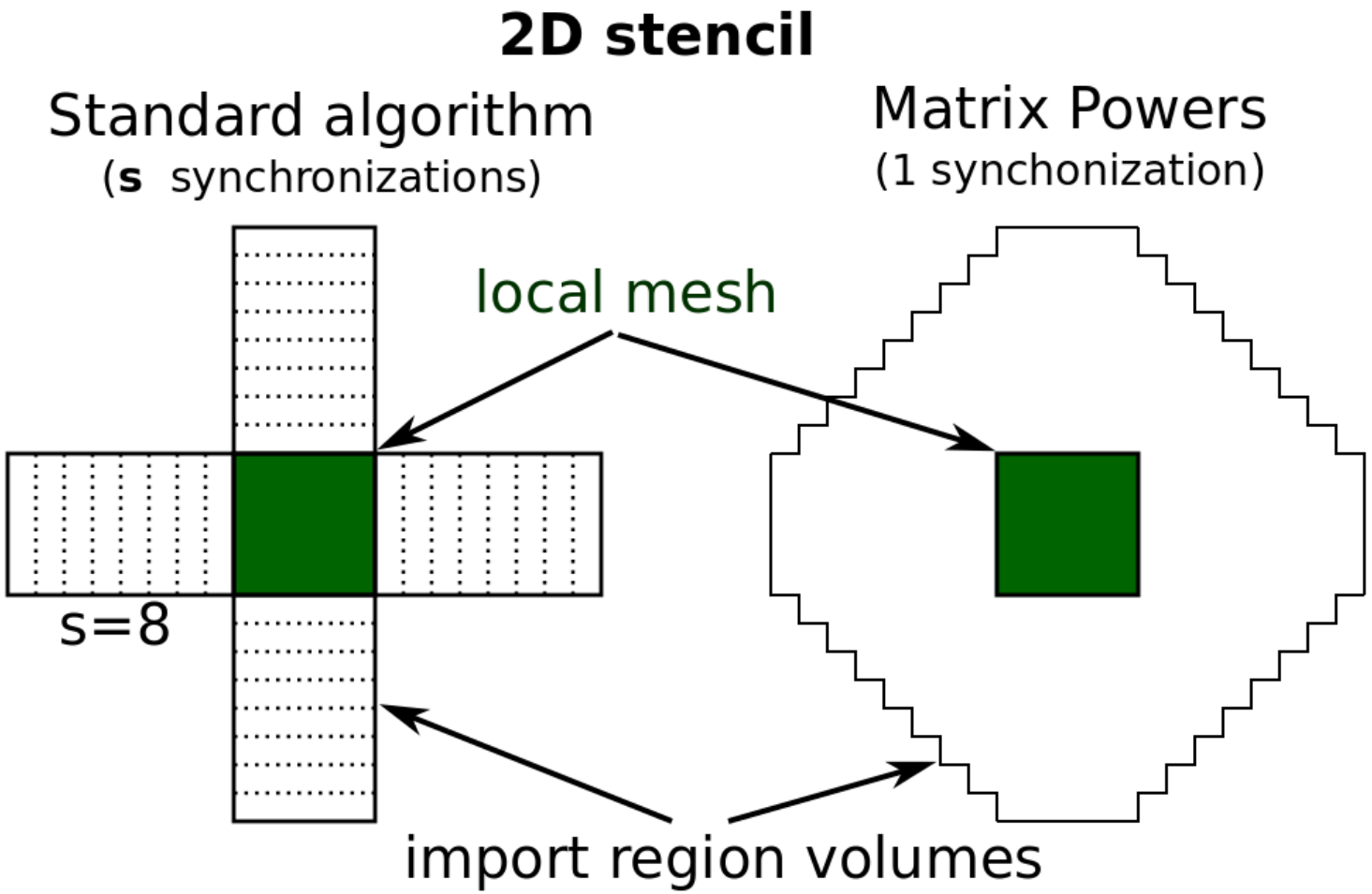


Communication-efficient stencil computations

Iterative s-step stencil computations

- previous work: in-time blocking (matrix powers kernel)
  - lowers synchronization cost by factor of b
  - lowers vertical communication cost by up to a factor of $b^{1/d}$
  - increases horizontal communication cost by an additive factor of $O(sb^{d-1})$, as dictated by the lower bound $W_{\mathrm{St}} \cdot S_{\mathrm{St}}^{d-1} = \Omega\left(s^d\right)$
- new 'block-cyclic' algorithm, in-time blocking executed bulk synchronously
  - lowers vertical communication cost by factor of $b^{1/d}$
  - maintains minimal horizontal communication cost
  - increases synchronization cost
- alternatives are both optimal in different lower bound regimes

Illustration of import region of the matrix-powers kernel



Scalable sparse matrix multiplication

Multiplication of a sparse matrix by a dense matrix

- key primitive with many applications
  - iterative solvers
  - tensor computations (MP3 or coupled cluster with localized orbitals)
  - graph algorithms (Bellman-Ford, APSP, betweenness centrality)
- new algorithms and analysis
  - communication-efficient 3D partitioning
  - lower bounds based on worst-case fill structure
  - extension to tensor contractions and implementation

Exploiting symmetry in tensors

Tensor symmetry (e.g. $A_{ij} = A_{ji}$) permits savings in memory and cost

- for order d tensor, $d!$ less memory
- matrix-vector multiplication

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

- 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

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

- for order ω contraction, $\omega!$ less multiplications (lower bilinear rank)

Symmetry preserving algorithms

By exploiting symmetry, we can reduce the number of multiplications at the cost of more additions

- algorithms generalize to most antisymmetric tensor contractions
- for Hermitian tensors, multiplications cost 3X more than additions
  - BLAS routines: `hemm` and `her2k` as well as LAPACK routines like `hetrd` (tridiagonal reduction) may be done with 25% fewer operations
- achieves $(2/3)n^3$ bilinear rank for squaring a nonsymmetric matrix, assuming elementwise commutativity
- can be nested for partially symmetric contractions
  - reduction in multiplications implies reduction in nested calls to contractions
  - yields significant reductions in overall cost

Horizontal communication cost of symmetry preserving algorithms

For contraction of order $s + v$ tensor with order $v + t$ tensor

- Υ is the nonsymmetric contraction algorithm
- Ψ is the best previously known algorithm
- Φ is the symmetry preserving algorithm

Asymptotic communication lower bounds based on bilinear expansion (H–cache size, p–#processors, n–dimension):

s	t	v	F_Υ	F_Ψ	F_Φ	$Q_{\Upsilon,\Psi}$	Q_Φ	W_Υ	W_Ψ	W_Φ
1	1	0	n^2	n^2	$\frac{n^2}{2}$	n^2	n^2	$\frac{n}{p^{1/2}}$	$\frac{n}{p^{1/2}}$	$\frac{n}{p^{1/2}}$
2	1	0	n^3	$\frac{n^3}{2}$	$\frac{n^3}{6}$	n^3	n^3	n	$\frac{n^2}{p^{2/3}}$	$\frac{n^2}{p^{2/3}}$
2	2	0	n^4	$\frac{n^4}{4}$	$\frac{n^4}{24}$	n^4	n^4	$\frac{n^2}{p^{1/2}}$	$\frac{n^2}{p^{1/2}}$	$\frac{n^2}{p^{1/2}}$
1	1	1	n^3	n^3	$\frac{n^3}{6}$	$\frac{n^3}{H^{1/2}}$	$\frac{n^3}{H^{1/2}}$	$\frac{n^2}{p^{2/3}}$	$\frac{n^2}{p^{2/3}}$	$\frac{n^2}{p^{2/3}}$
2	1	1	n^4	$\frac{n^4}{2}$	$\frac{n^4}{24}$	$\frac{n^4}{H^{1/2}}$	$\frac{n^4}{H^{1/3}}$	n^2	n^2	$\frac{n^3}{p^{3/4}}$
2	2	2	n^6	$\frac{n^6}{8}$	$\frac{n^6}{720}$	$\frac{n^6}{H^{1/2}}$	$\frac{n^6}{H^{1/2}}$	$\frac{n^4}{p^{2/3}}$	$\frac{n^4}{p^{2/3}}$	$\frac{n^4}{p^{2/3}}$

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)

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 example code snippet

```
void 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
}
```

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

```
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

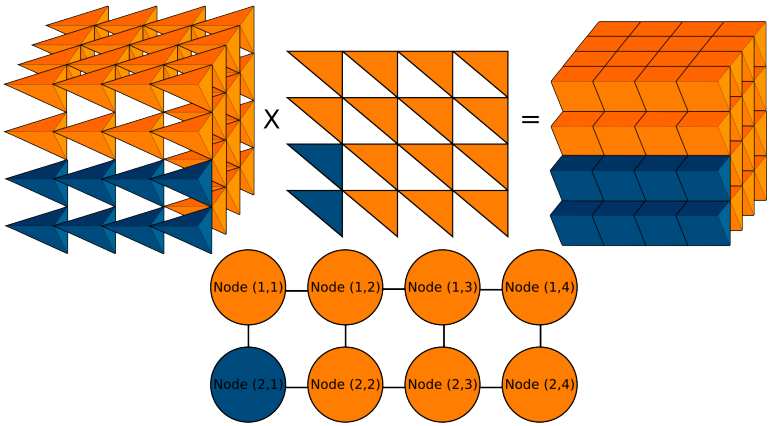
Betweenness centrality code snippet, for k of n nodes

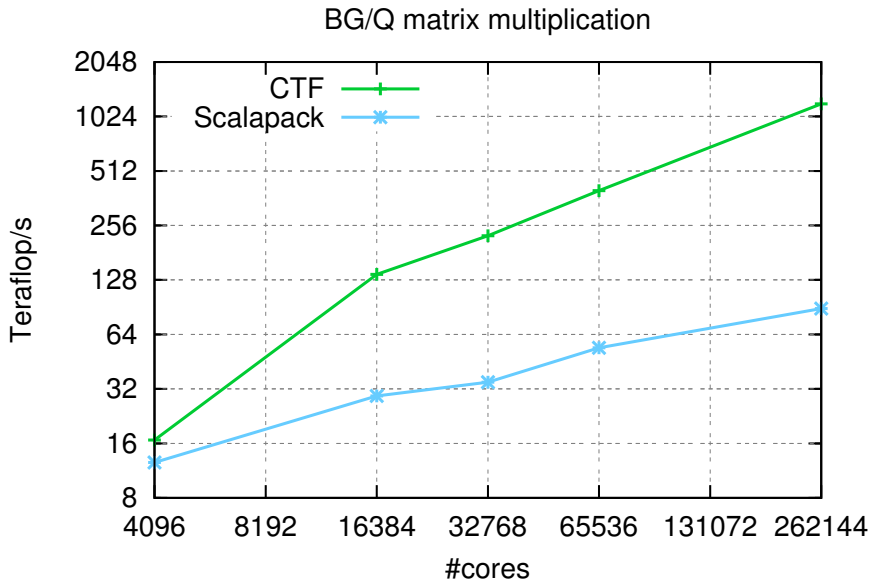
```
void btwn_central(Matrix<int> A, Matrix<path> P, int n, int k){
  Monoid<path> mon(...,
                   [](path a, path b){
                     if (a.w<b.w) return a;
                     else if (b.w<a.w) return b;
                     else return path(a.w, a.m+b.m);
                   }, ...);

  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 dense computations

CTF is highly tuned for massively-parallel machines

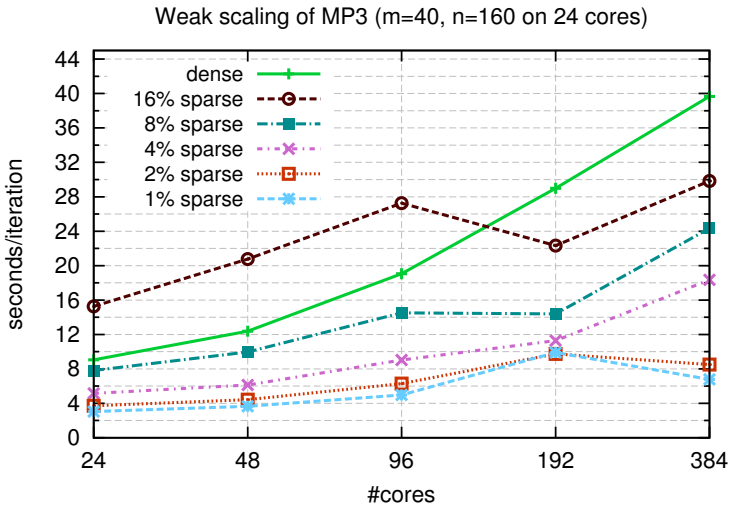
- virtualized multidimensional processor grids
- topology-aware mapping and collective communication
- performance-model-driven decomposition done at runtime
- optimized redistribution kernels for tensor transposition

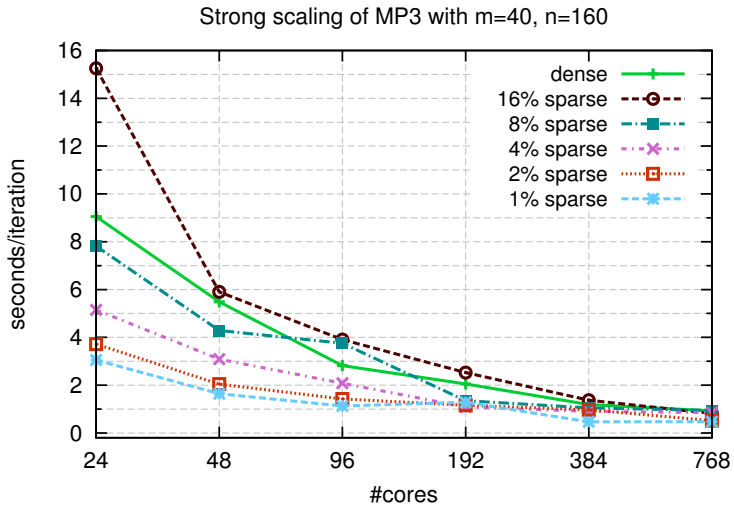




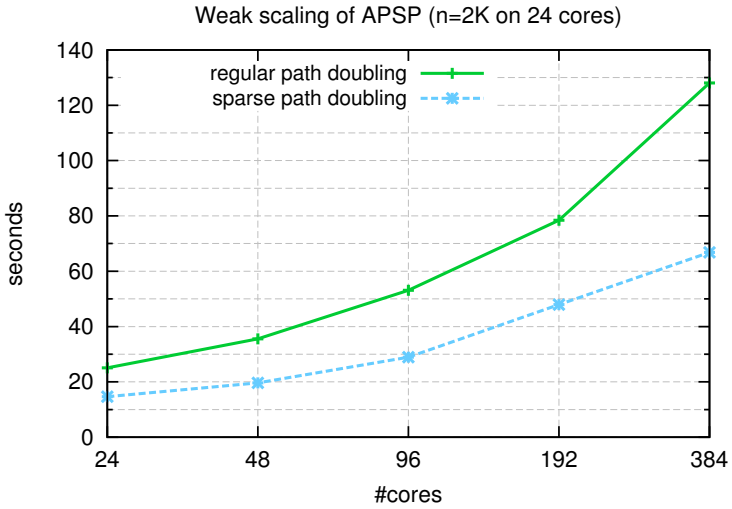
Performance of CTF for sparse computations

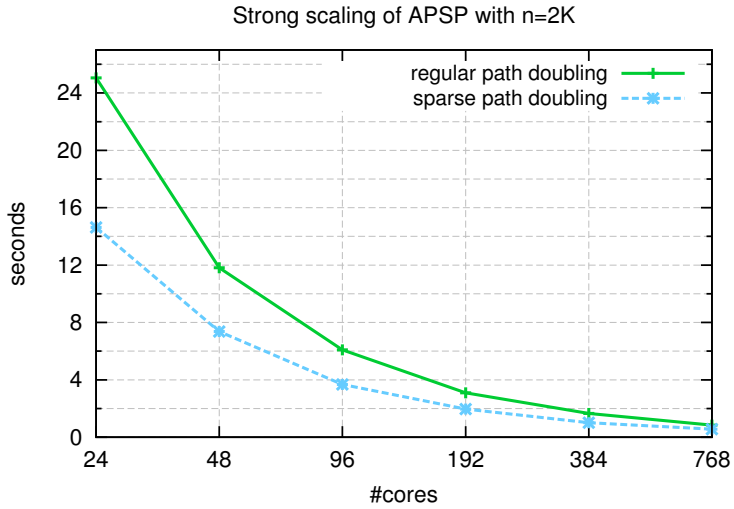
MP3 leveraging sparse-dense tensor contractions





All-pairs shortest-paths based on path doubling with sparsification





Application to Coupled Cluster

Coupled cluster methods

- approximate solution to manybody time-independent Schrödinger equation via set of nonlinear equations
- use $2r$-order tensor to represent r-electron correlation
  - measure excitation energy of r electrons to r virtual orbitals
  - each possible set of transitions (intermediate states) becomes a product of tensors
  - factorization of this set of multilinear tensor products transforms equations to bilinear tensor contractions
- systematically improvable, CCSD, CCSDT, CCSDTQ ($r = 1, 2, 3$)
- partial antisymmetries in tensors as a consequence of electron interchangeability
- symmetry preserving algorithms can reduce cost of CCSD by about 1.3, CCSDT by about 2.1

CCSD using CTF

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

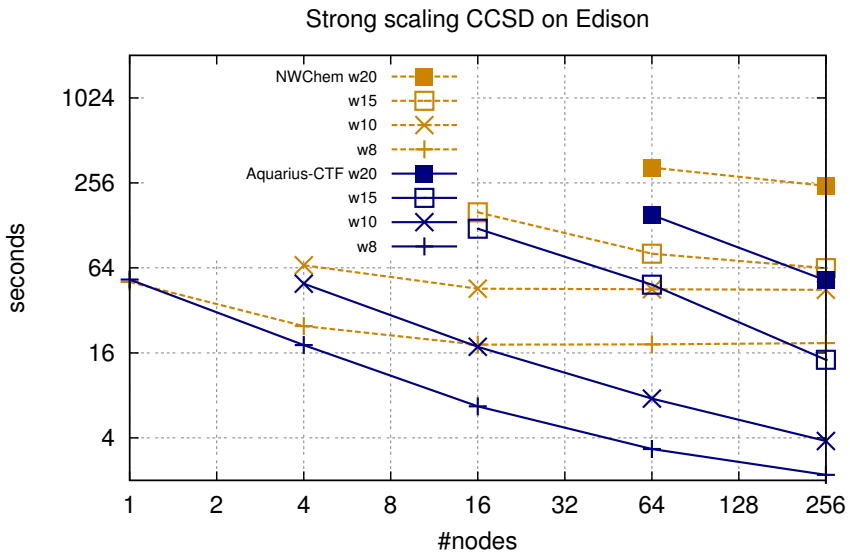
```
FMI["mi"]     += 0.5*WMNEF["mnef"]*T(2)["efin"];
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T(2)["efij"];
FAE["ae"]     -= 0.5*WMNEF["mnef"]*T(2)["afmn"];
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T(2)["afin"];

Z(2)["abij"]  = WMNEF["ijab"];
Z(2)["abij"] += FAE["af"]*T(2)["fbij"];
Z(2)["abij"] -= FMI["ni"]*T(2)["abnj"];
Z(2)["abij"] += 0.5*WABEF["abef"]*T(2)["efij"];
Z(2)["abij"] += 0.5*WMNIJ["mnij"]*T(2)["abmn"];
Z(2)["abij"] -= WAMEI["amei"]*T(2)["ebmj"];
```

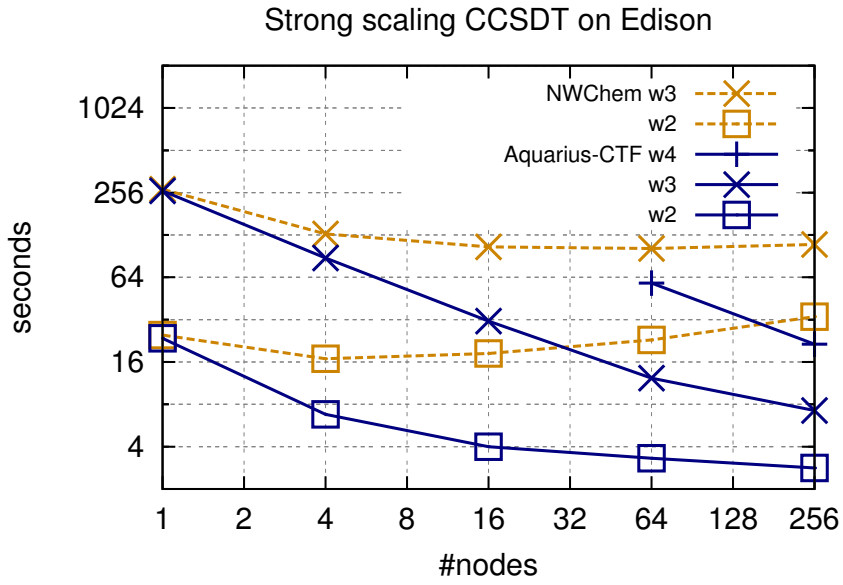
Other electronic structure codes using CTF include QChem (via Libtensor)

Comparison with NWChem

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

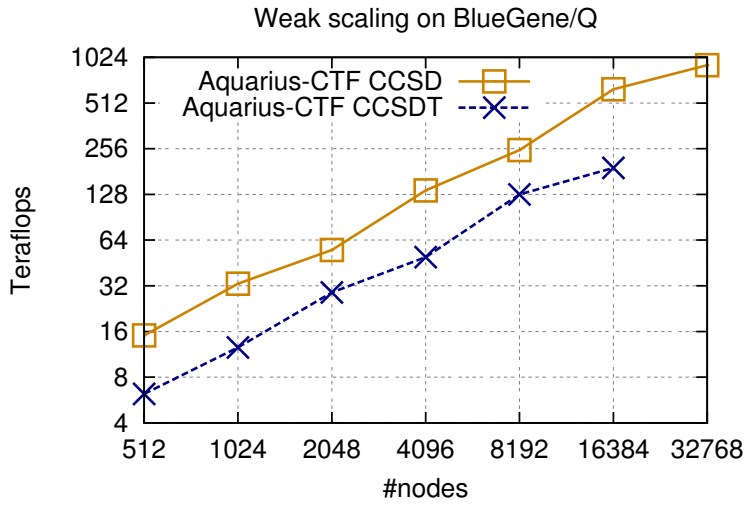
- provides CCSD and CCSDT
- uses Global Arrays a Partitioned Global Address Space (PGAS) backend for tensor contractions
- derives equations via Tensor Contraction Engine (TCE)

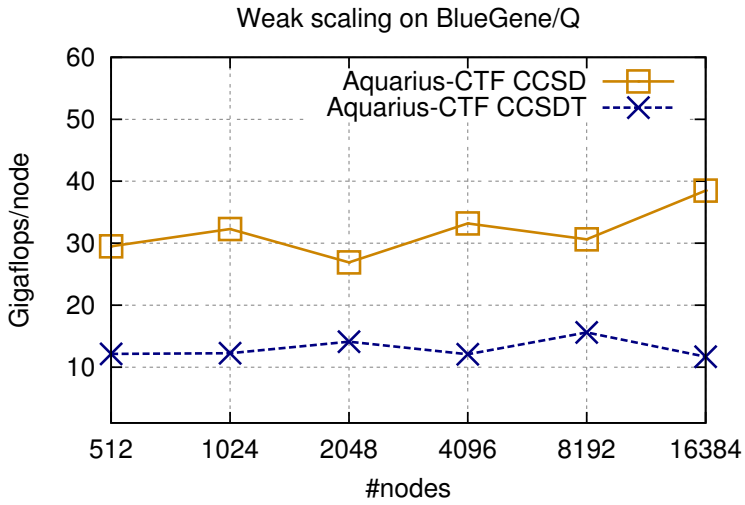


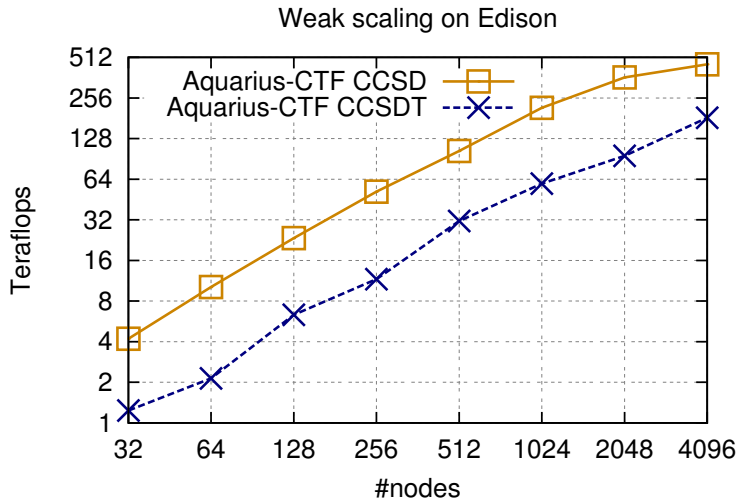


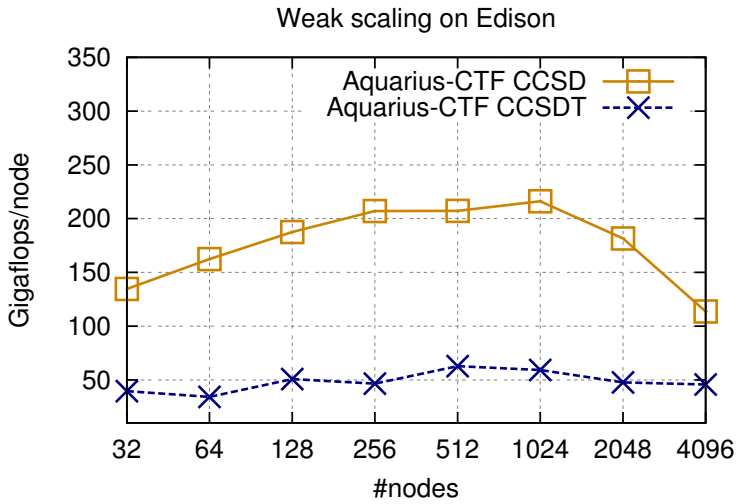
Coupled Cluster on IBM BlueGene/Q

CCSD up to 55 (50) water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ







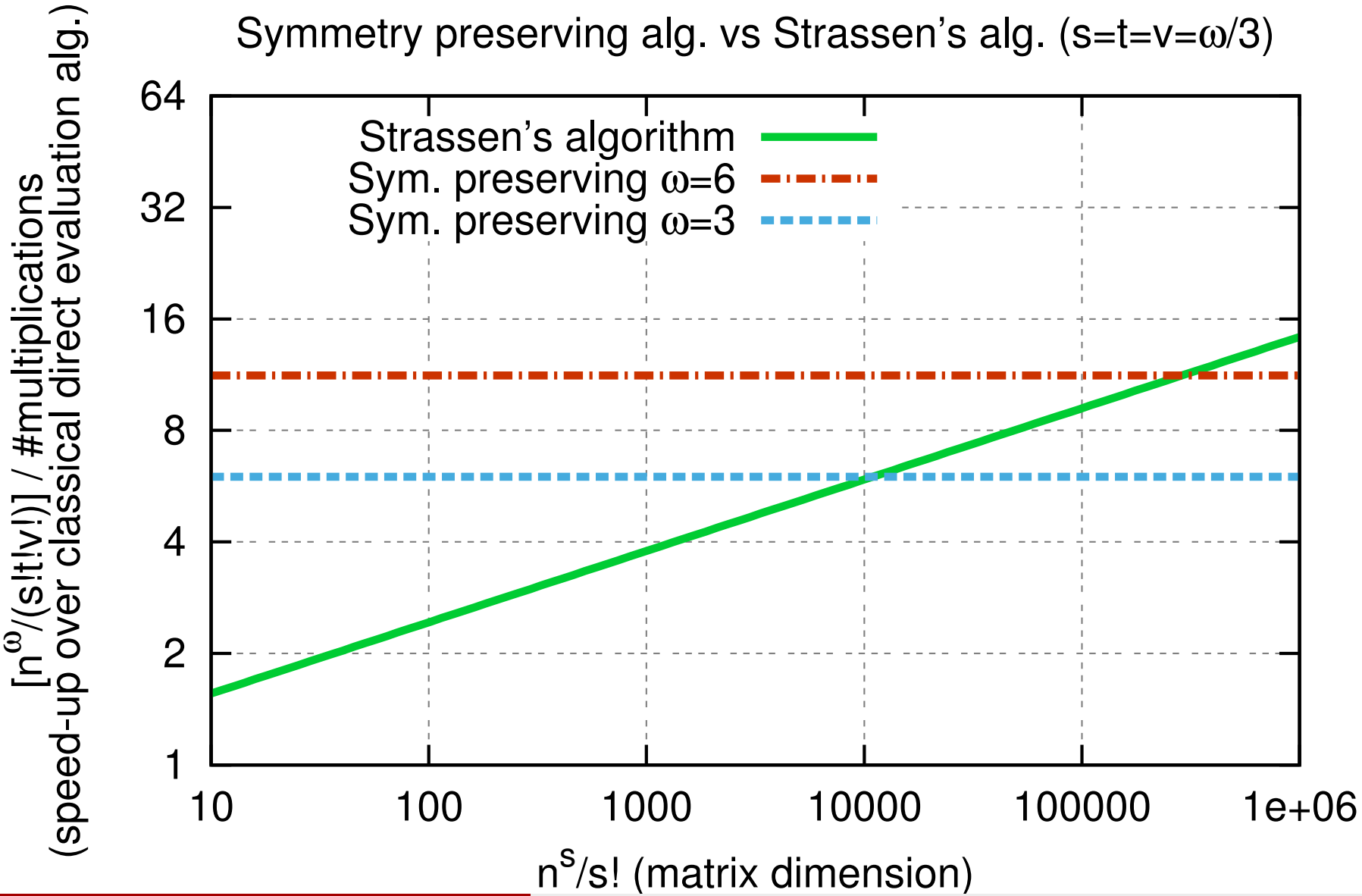


Future work

- further work sparse and symmetric tensor computations
  - bridging the gap between abstractions and application performance
  - bilinear algorithm complexity – fast matrix multiplication
- tensor decompositions
  - communication-efficient parallel algorithms and lower bounds
  - symmetry-preserving tensor decomposition and DMRG algorithms
  - programming abstractions for dense and sparse tensors
- sets of tensor operations
  - most algorithms correspond to multiple dependent tensors operations
  - communication cost analysis for sets of contractions
  - scheduling, blocking, and decomposition of multiple tensor operations
  - higher-level programming abstractions
- application-driven development
  - tensor decompositions, sparsity, symmetry all motivated by electronic structure applications
  - optimization of primitives serves as feedback loop for development of new electronic structure methods

Backup slides

Symmetry preserving algorithm vs Strassen's algorithm



Nesting of bilinear algorithms

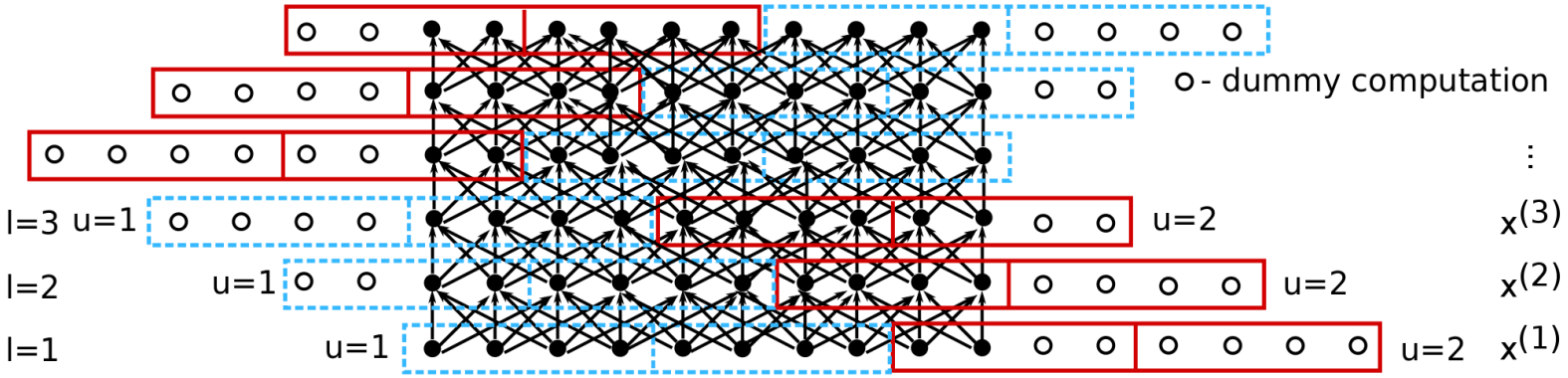
Given two bilinear algorithms:

$$\Lambda_1 = (\mathbf{F_1^{(A)}}, \mathbf{F_1^{(B)}}, \mathbf{F_1^{(C)}})$$
$$\Lambda_2 = (\mathbf{F_2^{(A)}}, \mathbf{F_2^{(B)}}, \mathbf{F_2^{(C)}})$$

We can nest them by computing their tensor product

$$\Lambda_1 \otimes \Lambda_2 := (\mathbf{F_1^{(A)}} \otimes \mathbf{F_2^{(A)}}, \mathbf{F_1^{(B)}} \otimes \mathbf{F_2^{(B)}}, \mathbf{F_1^{(C)}} \otimes \mathbf{F_2^{(C)}})$$
$$\mathsf{rank}(\Lambda_1 \otimes \Lambda_2) = \mathsf{rank}(\Lambda_1) \cdot \mathsf{rank}(\Lambda_2)$$

Block-cyclic algorithm for s-step methods

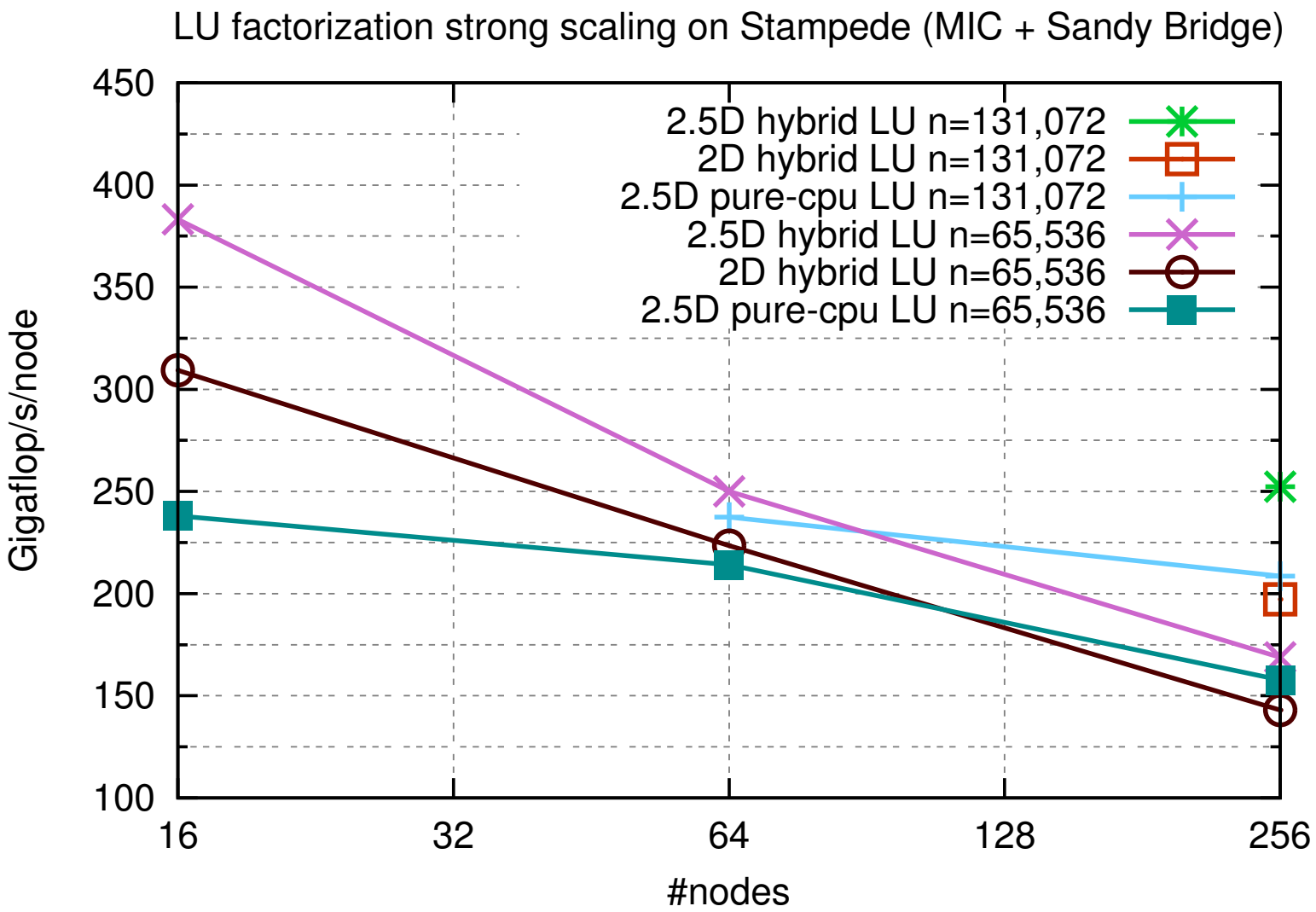


For s-steps of a $(2m+1)^d$-point stencil with block-size of $H^{1/d}/m$,

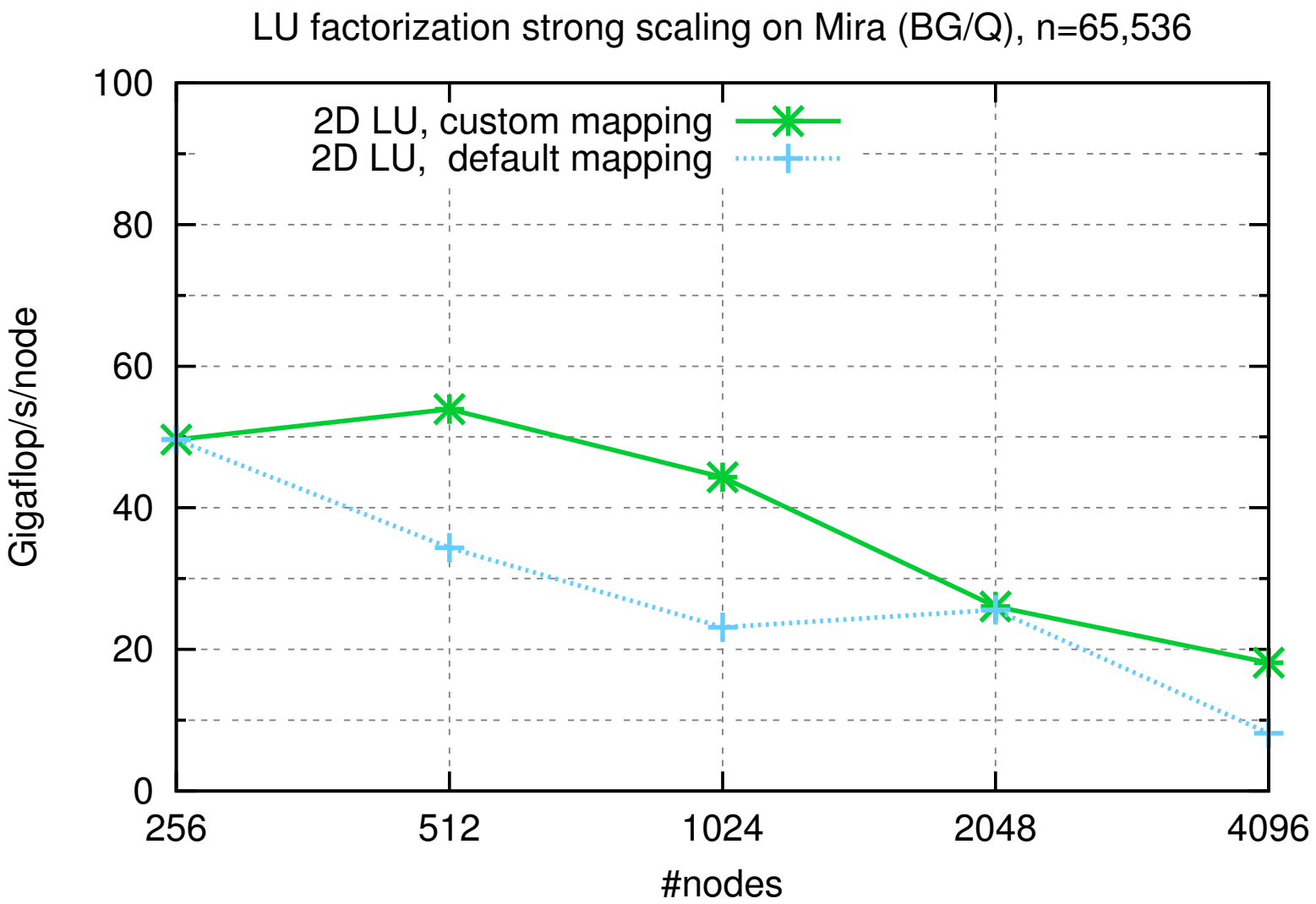
$$W_{\mathrm{Kr}} = O\left(\frac{msn^d}{H^{1/d}p}\right) \quad S_{\mathrm{Kr}} = O(sn^d/(pH)) \quad Q_{\mathrm{Kr}} = O\left(\frac{msn^d}{H^{1/d}p}\right)$$

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

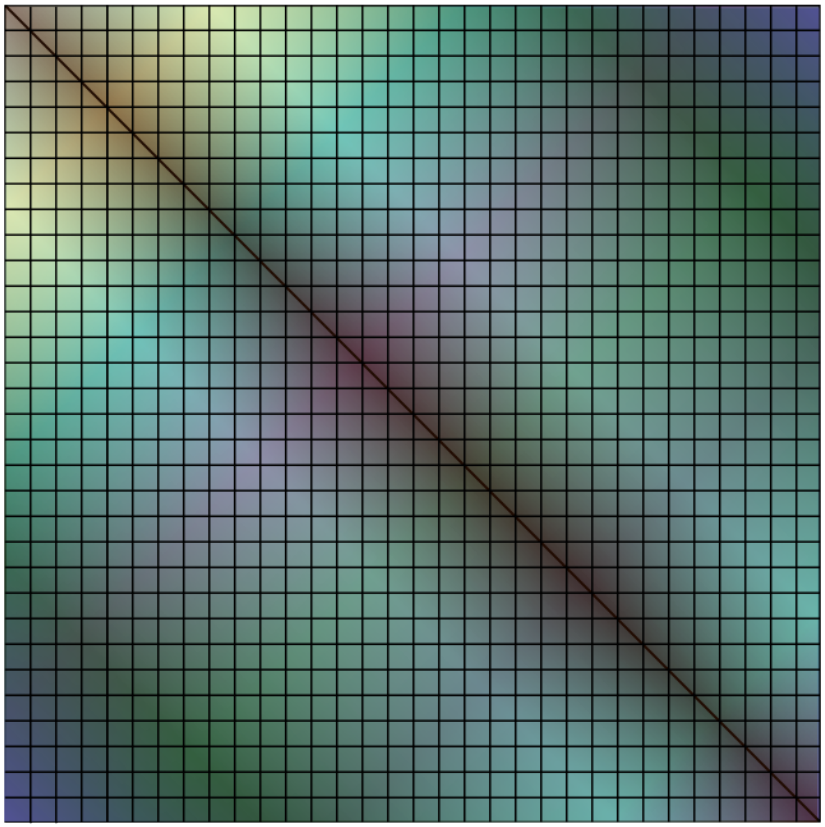

LU factorization strong scaling on Stampede (MIC + Sandy Bridge)
Gigaflop/s/node
#nodes
450
400
350
300
250
200
150
100
16
32
64
128
256
2.5D hybrid LU n=131,072
2D hybrid LU n=131,072
2.5D pure-cpu LU n=131,072
2.5D hybrid LU n=65,536
2D hybrid LU n=65,536
2.5D pure-cpu LU n=65,536

Topology-aware mapping on BG/Q

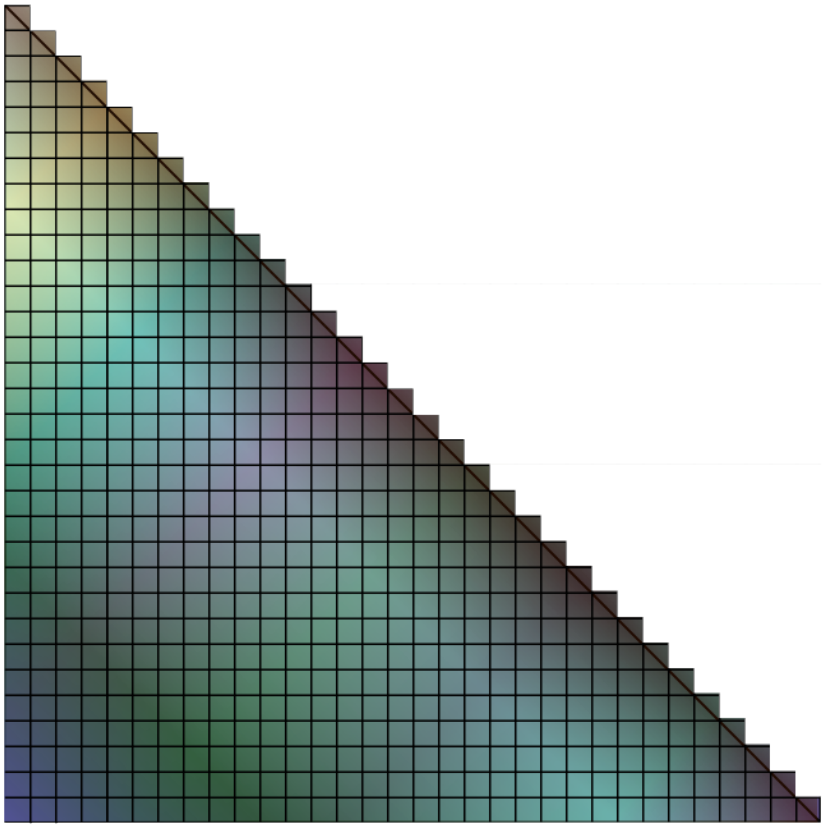


Symmetric matrix representation

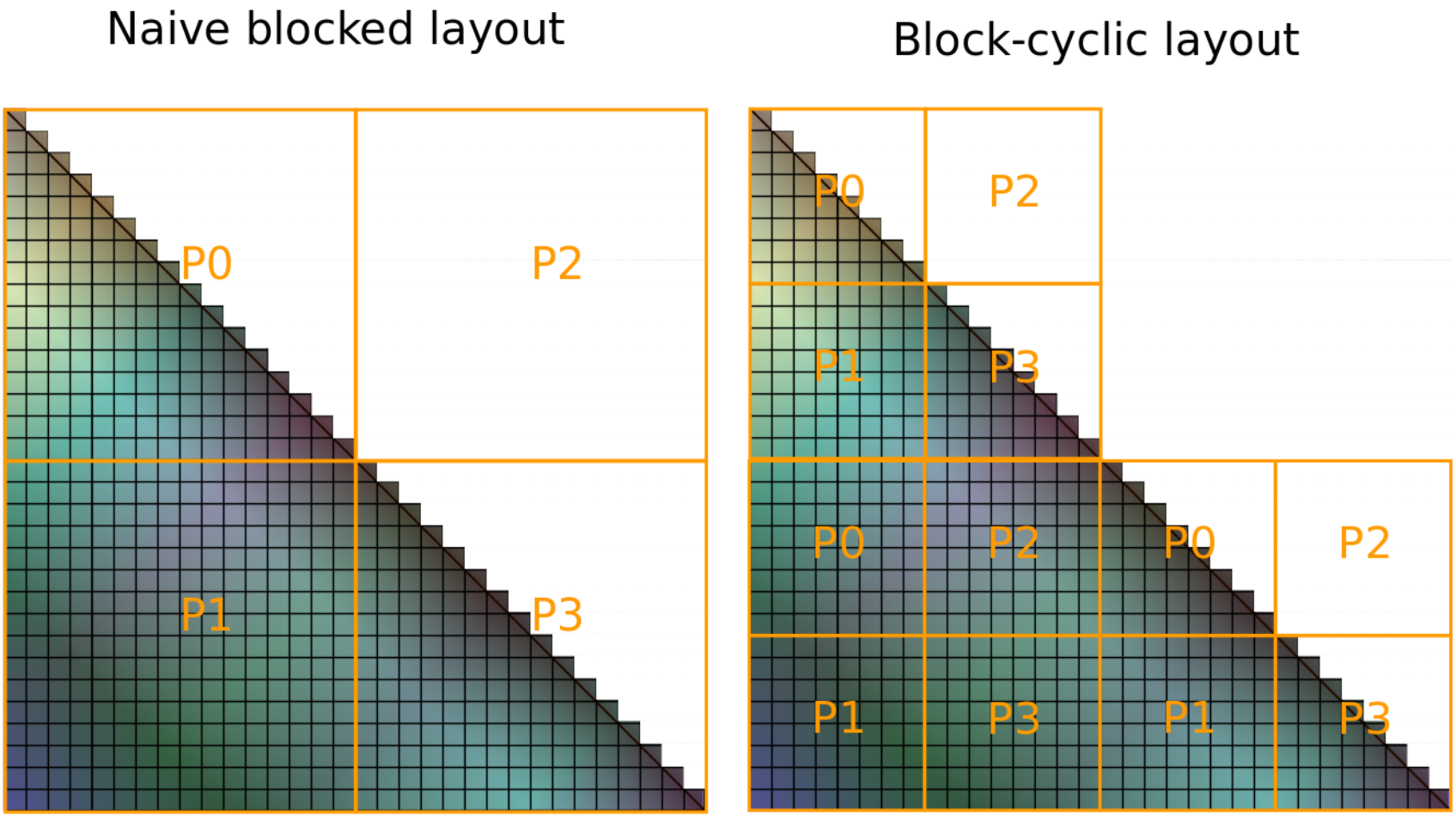
Symmetric matrix



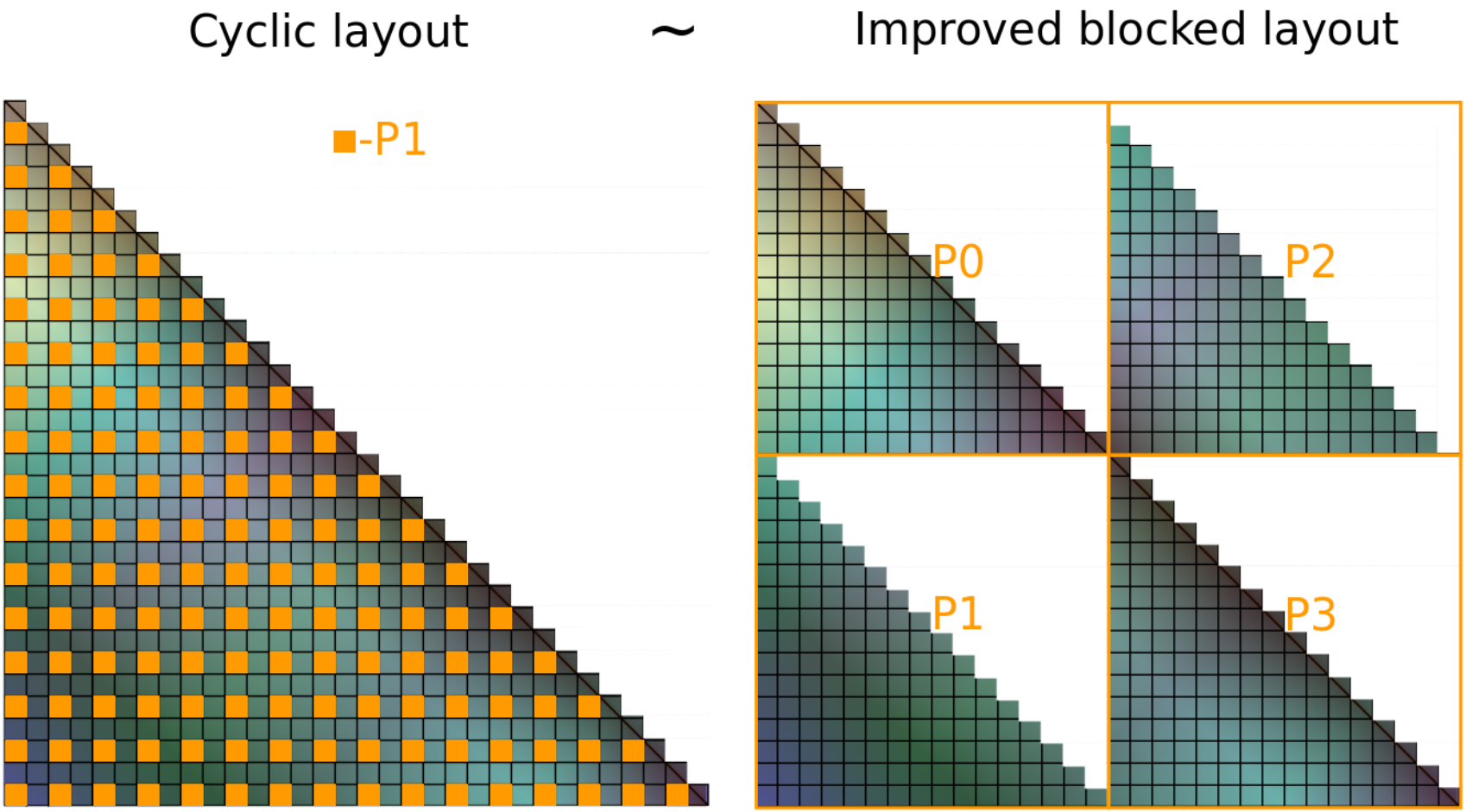
Unique part of symmetric matrix



Blocked distributions of a symmetric matrix



Cyclic distribution of a symmetric matrix



Our CCSD factorization

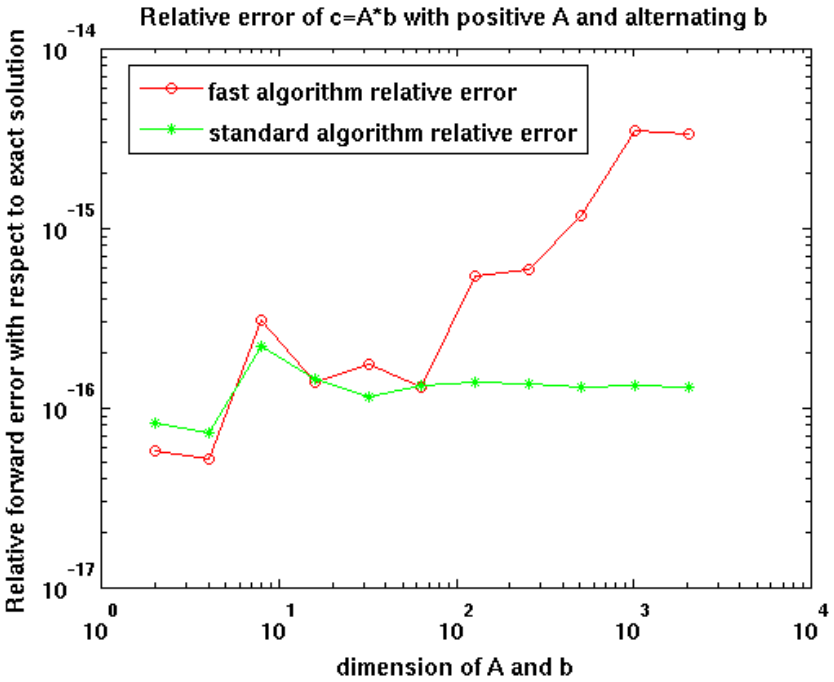
Credit to John F. Stanton and Jurgen Gauss

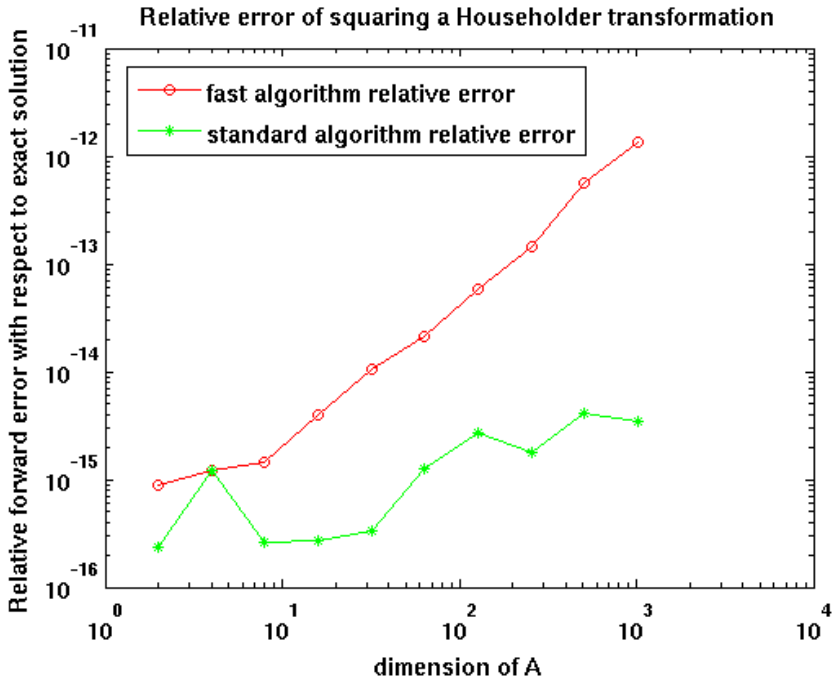
$$
\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1-\delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\
\tilde{F}_i^m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

Our CCSD factorization

$$
\begin{aligned}
\tilde{W}_{ei}^{mn} &= v_{ei}^{mn} + \sum_f v_{ef}^{mn} t_i^f, \\
\tilde{W}_{ij}^{mn} &= v_{ij}^{mn} + P_j^i \sum_e v_{ie}^{mn} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{mn} \tau_{ij}^{ef}, \\
\tilde{W}_{ie}^{am} &= v_{ie}^{am} - \sum_n \tilde{W}_{ei}^{mn} t_n^a + \sum_f v_{ef}^{ma} t_i^f + \frac{1}{2} \sum_{nf} v_{ef}^{mn} t_{in}^{af}, \\
\tilde{W}_{ij}^{am} &= v_{ij}^{am} + P_j^i \sum_e v_{ie}^{am} t_j^e + \frac{1}{2} \sum_{ef} v_{ef}^{am} \tau_{ij}^{ef}, \\
z_i^a &= f_i^a - \sum_m \tilde{F}_i^m t_m^a + \sum_e f_e^a t_i^e + \sum_{em} v_{ei}^{ma} t_m^e + \sum_{em} v_{im}^{ae} \tilde{F}_e^m + \frac{1}{2} \sum_{efm} v_{ef}^{am} \tau_{im}^{ef} \\
&- \frac{1}{2} \sum_{emn} \tilde{W}_{ei}^{mn} t_{mn}^{ea}, \\
z_{ij}^{ab} &= v_{ij}^{ab} + P_j^i \sum_e v_{ie}^{ab} t_j^e + P_b^a P_j^i \sum_{me} \tilde{W}_{ie}^{am} t_{mj}^{eb} - P_b^a \sum_m \tilde{W}_{ij}^{am} t_m^b \\
&+ P_b^a \sum_e \tilde{F}_e^a t_{ij}^{eb} - P_j^i \sum_m \tilde{F}_i^m t_{mj}^{ab} + \frac{1}{2} \sum_{ef} v_{ef}^{ab} \tau_{ij}^{ef} + \frac{1}{2} \sum_{mn} \tilde{W}_{ij}^{mn} \tau_{mn}^{ab},
\end{aligned}
$$

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%	$O(v^4o^2/p)$	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

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)

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

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

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

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.