Algorithms as Multilinear Tensor Equations

Edgar Solomonik

Department of Computer Science
ETH Zurich

Georgia Institute of Technology, Atlanta GA, USA

February 18, 2016

Pervasive paradigms in scientific computing

What commonalities exist among resource-intensive computations in simulation and data analysis?

- multidimensional datasets (observations, discretizations)
- higher-order relations between datasets, i.e. equations, maps, graphs, hypergraphs
- **sparsity** and **symmetry** in structure of relations
- relations lead to solution directly or by acting as an evolutionary (iterative) criterion
- **algebraic descriptions of datasets and relations**

Pervasive paradigms in scientific computing

What type of abstractions are desirable in high performance computing?

- data abstractions should reflect native dimensionality and structure
- global functional abstractions should efficiently orchestrate **communication** and **synchronization**
- **abstractions should enable development of provably efficient algorithms**

Outline

Introduction to tensor computations

Symmetry-preserving tensor algorithms

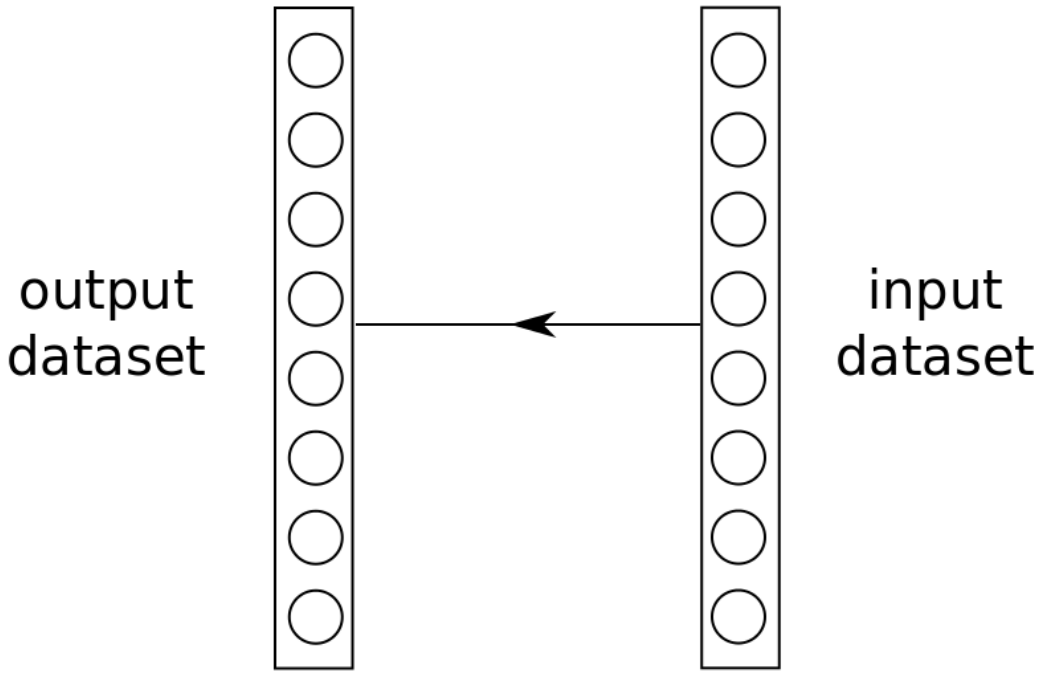
Communication-avoiding parallel algorithms

A massively-parallel tensor framework

Applications to electronic structure calculations

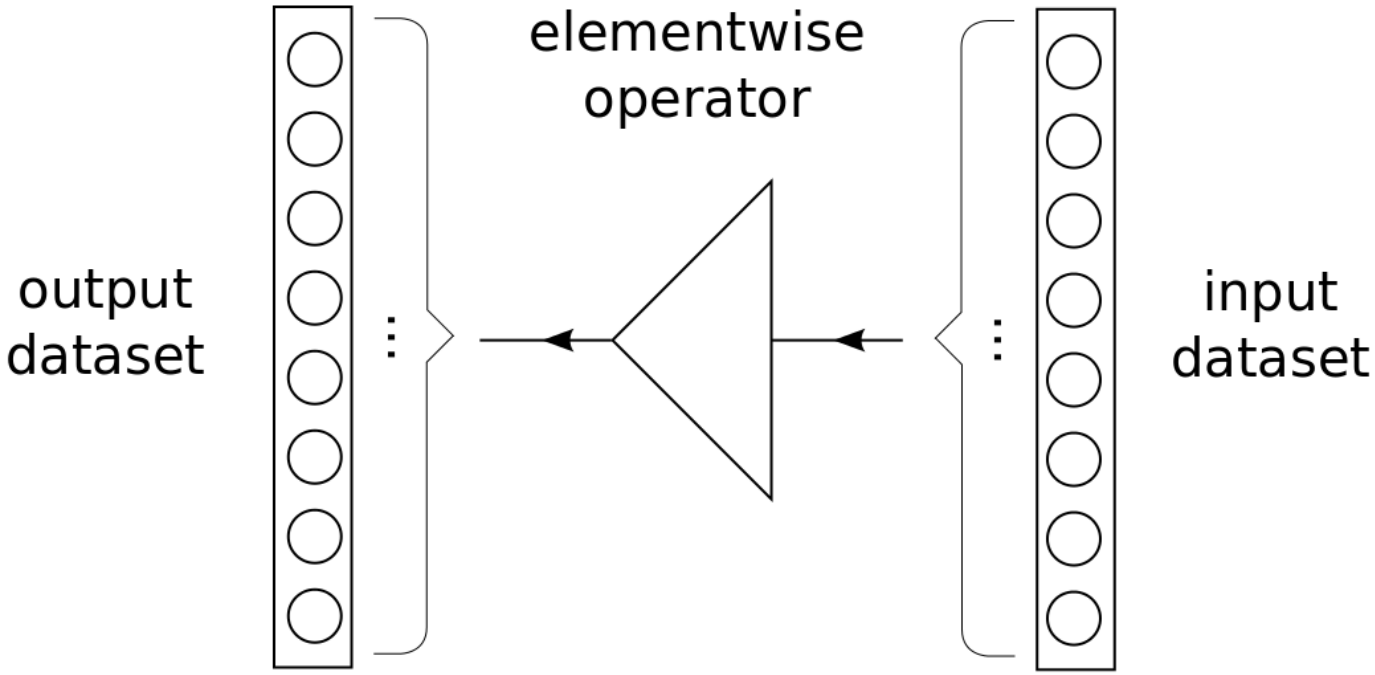
Conclusion

Basic data (vector) parallelism



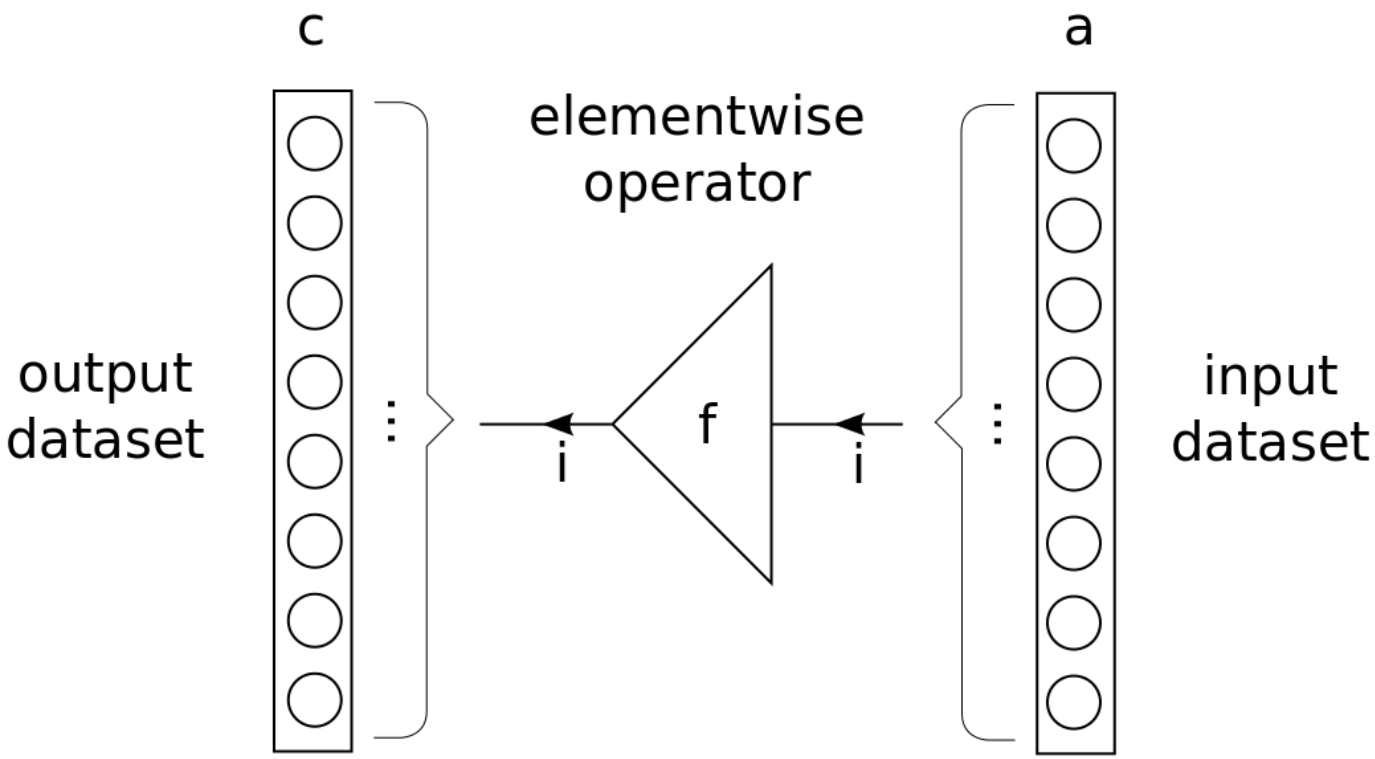
Basic data (vector) parallelism

$$c_i = f(a_i)$$



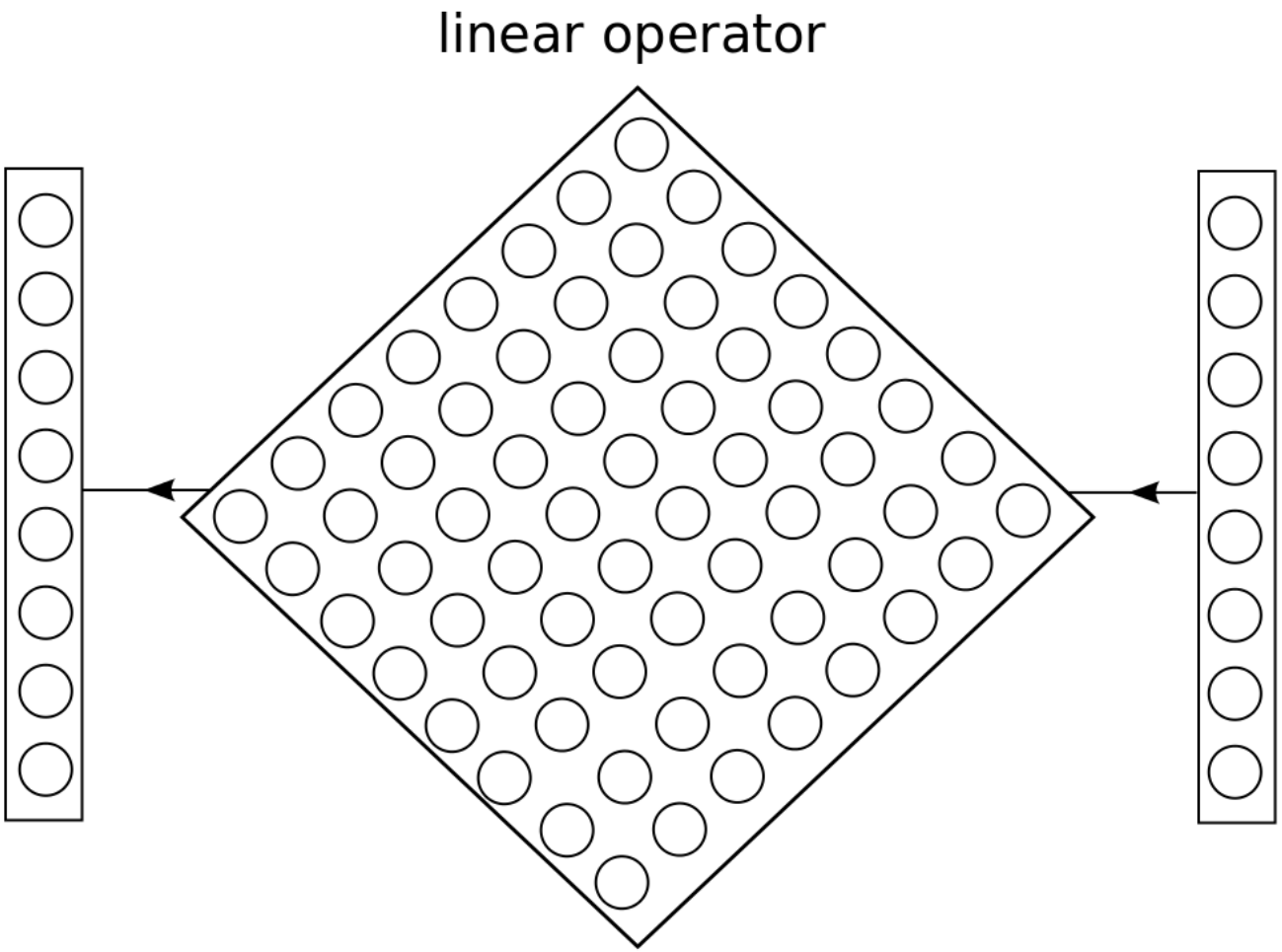
Basic data (vector) parallelism

$$c_i = f(a_i)$$



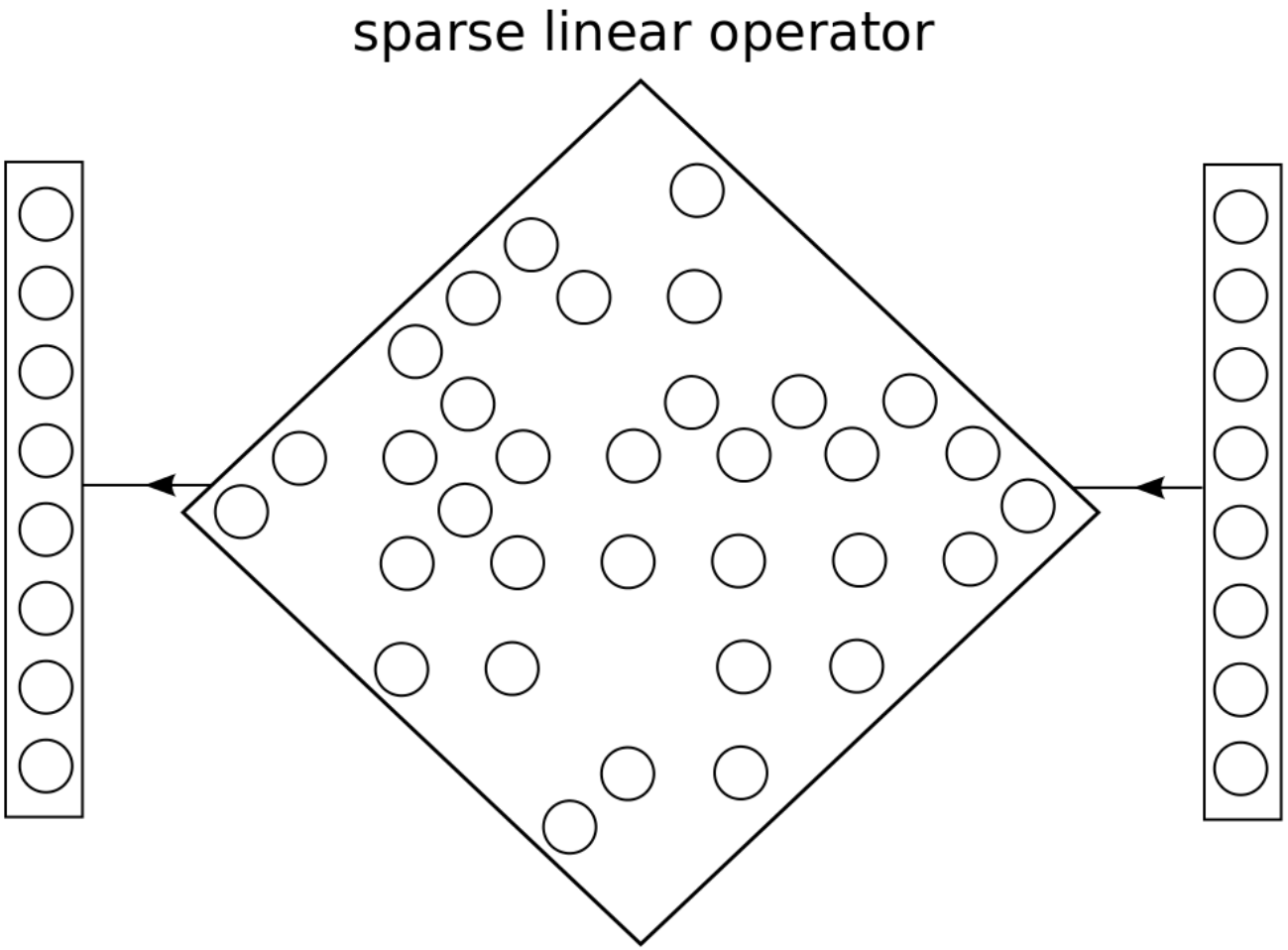
Dense matrix operators

$$\mathbf{c} = \mathbf{A}\mathbf{b}$$



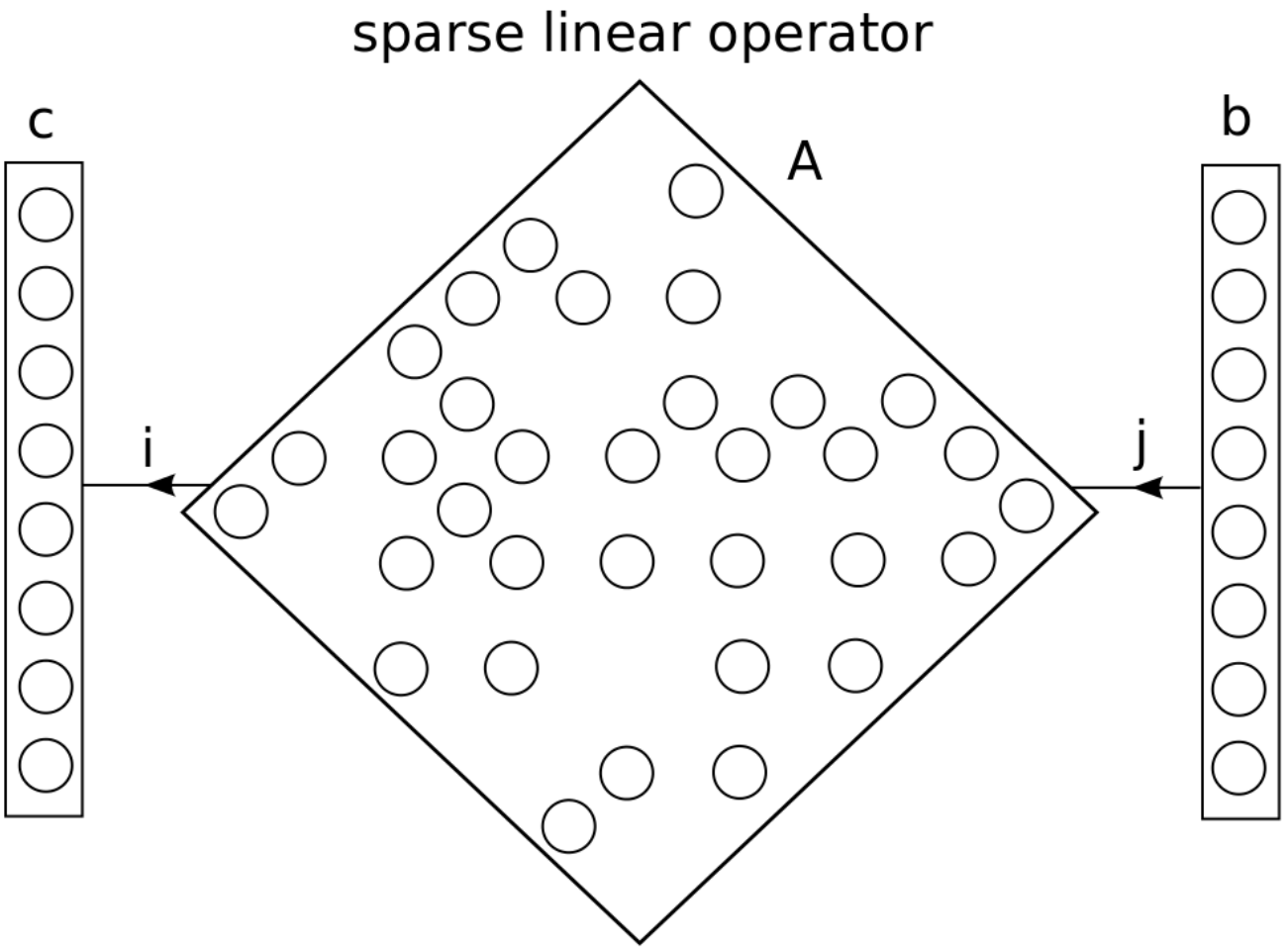
Sparse matrix operators

$$\mathbf{c} = \mathbf{A}\mathbf{b}$$



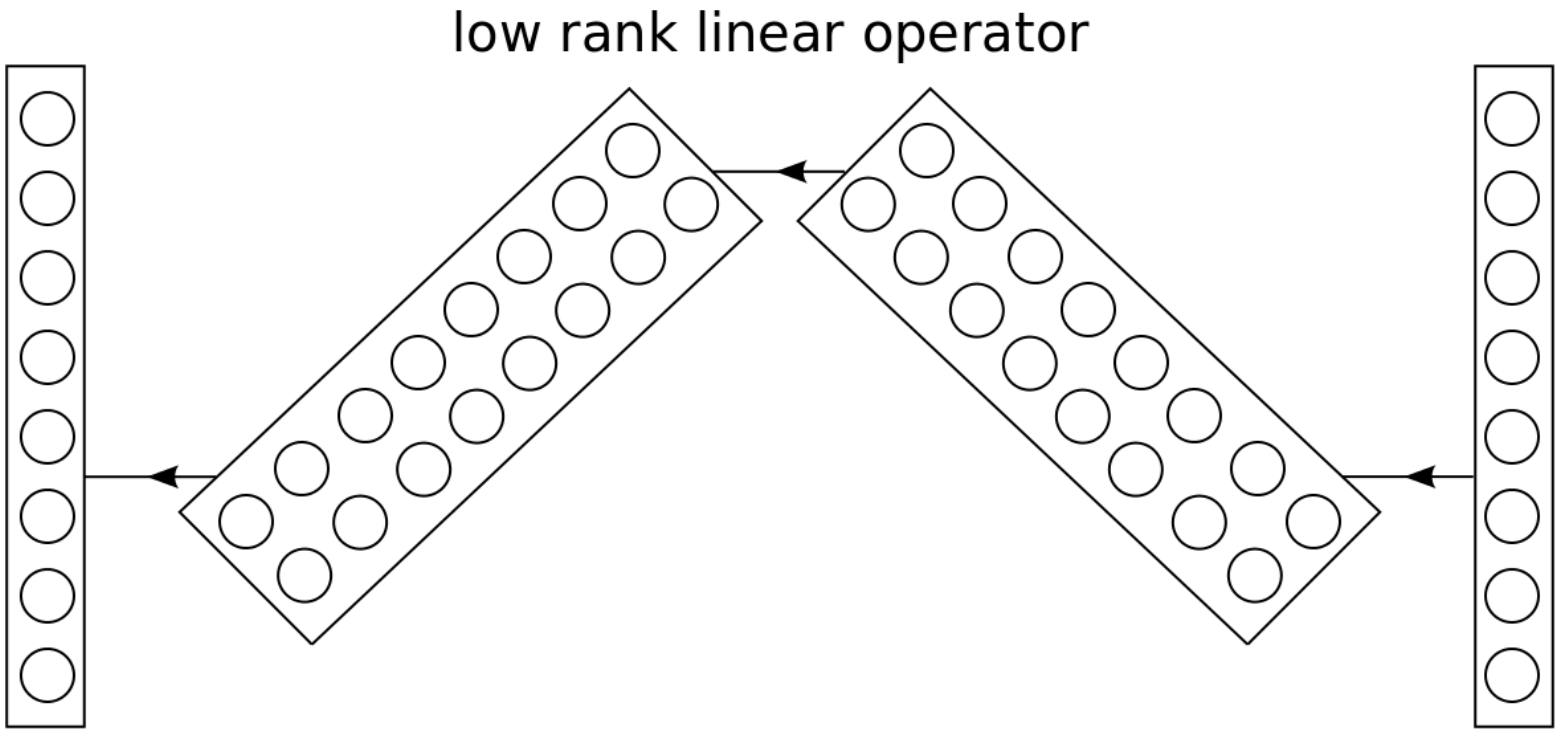
Sparse matrix operators

$$c_i = \sum_j A_{ij} b_j$$



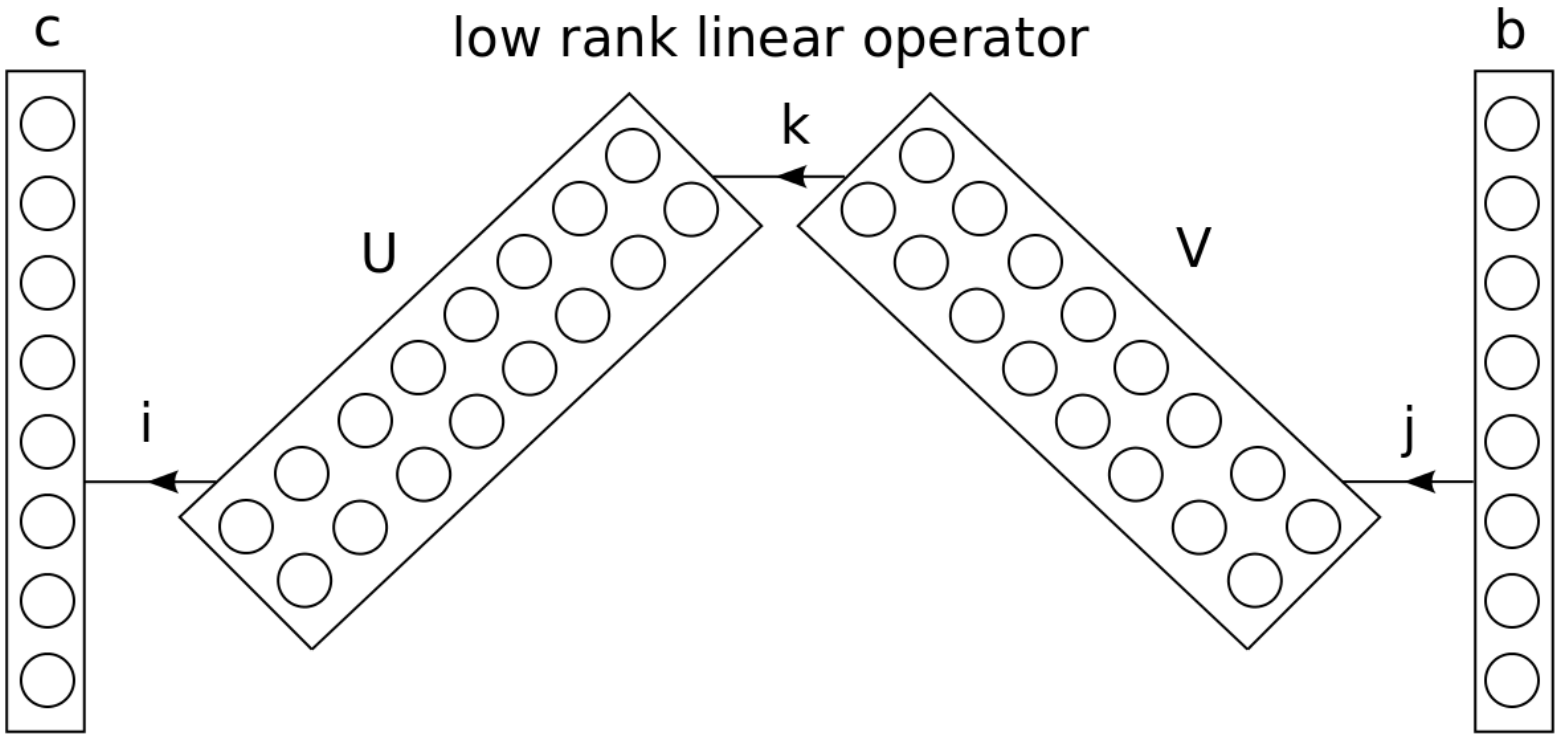
Low-rank matrix factorizations

$$\mathbf{A} = \mathbf{UV} \quad \rightarrow \quad \mathbf{c} = \mathbf{UVb}$$



Low-rank matrix factorizations

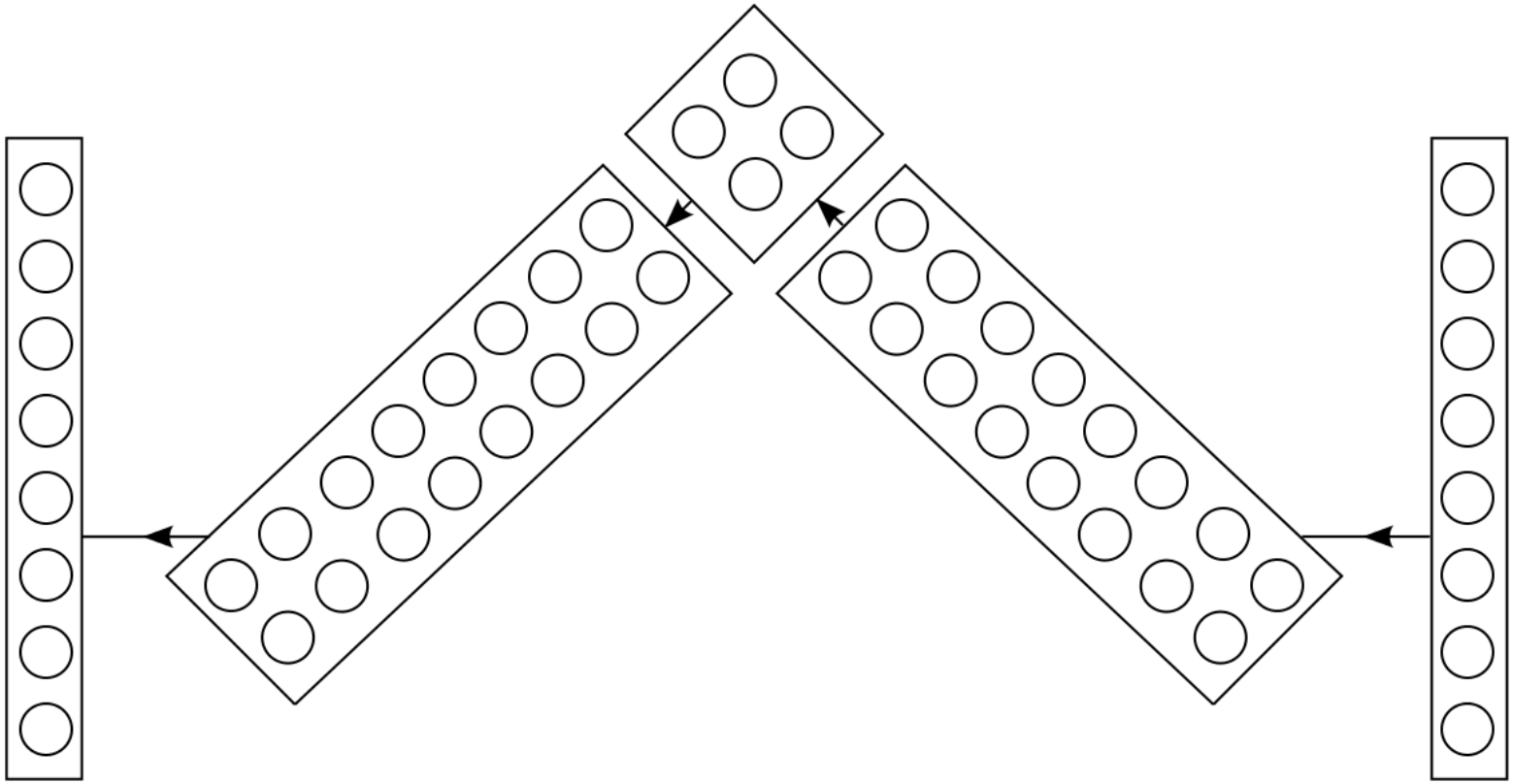
$$A_{ij} = \sum_k U_{ik} V_{kj} \quad \rightarrow \quad c_i = \sum_{j,k} U_{ik} V_{kj} b_j$$



Low-rank matrix factorizations

$$\mathbf{c} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}\mathbf{b}$$

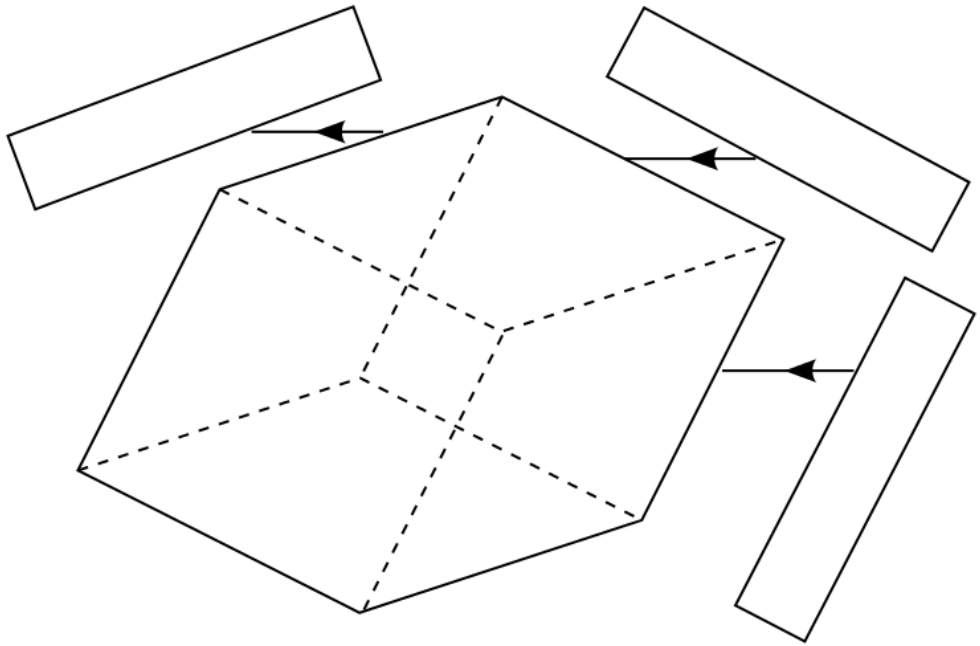
low rank linear operator



Tensor operator

$$d_i = \sum_{j,k} A_{ijk} b_j c_k$$

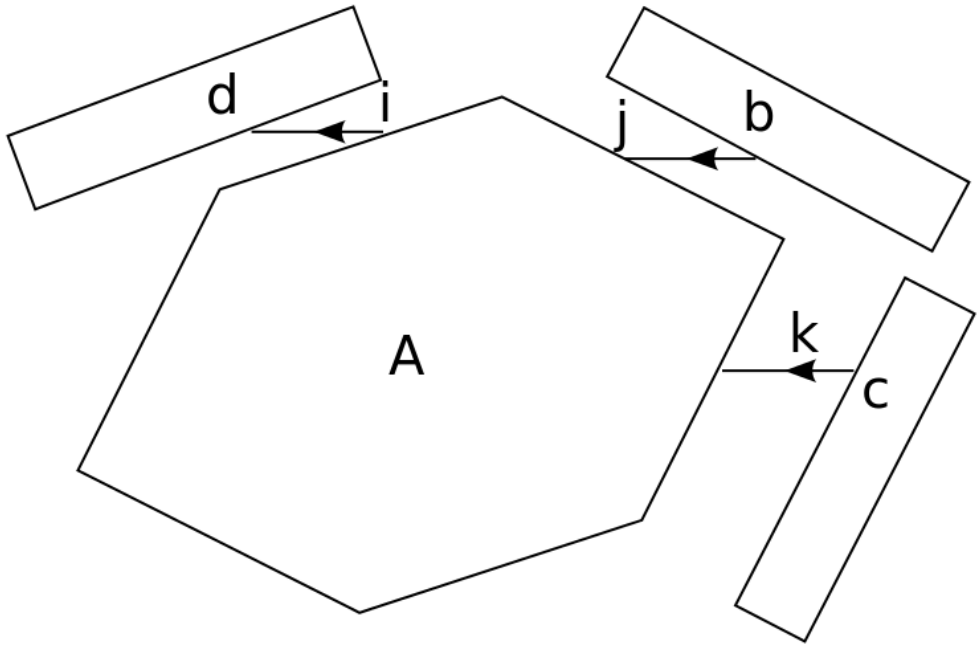
multilinear operator



Tensor operator

$$d_i = \sum_{j,k} A_{ijk} b_j c_k$$

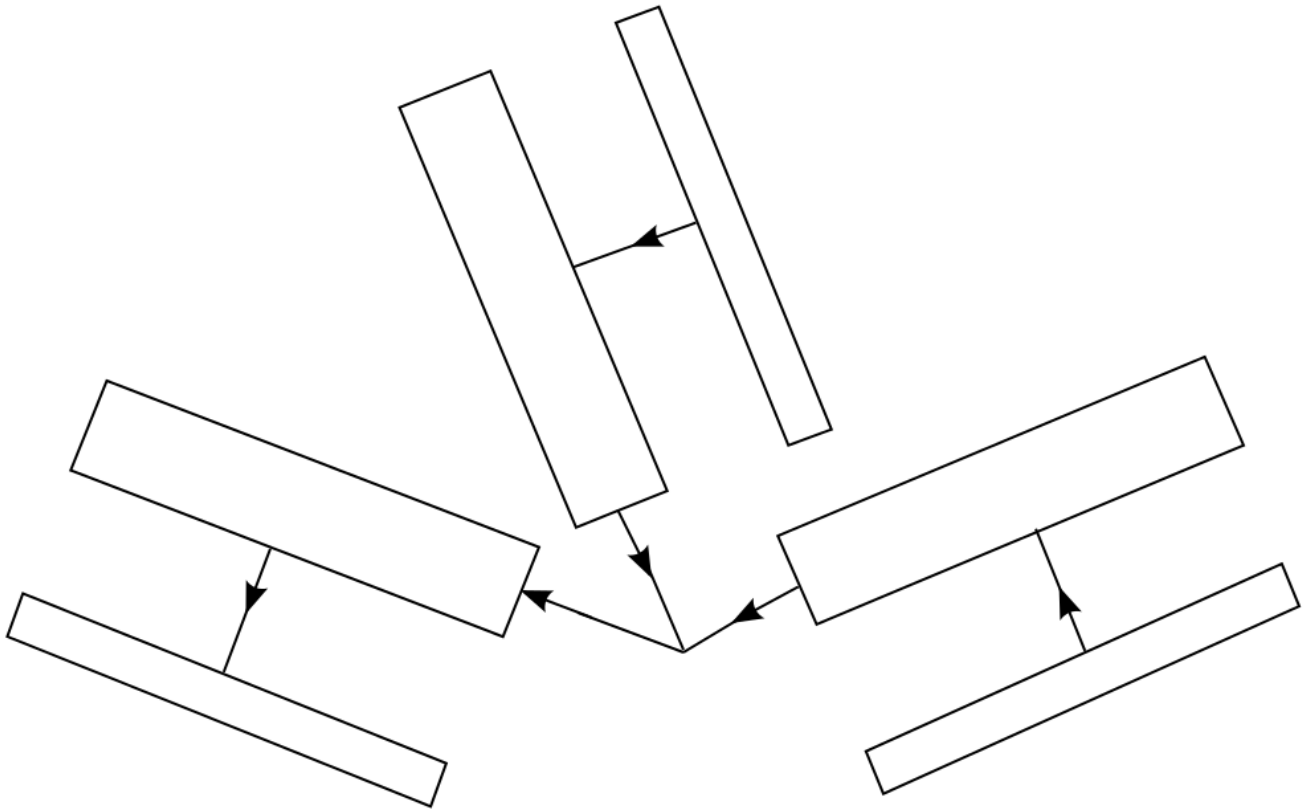
multilinear operator



Low-rank tensor factorizations (CP)

$$A_{ijk} = \sum_l U_{il} V_{jl} W_{kl} \quad \rightarrow \quad d_i = \sum_{j,k,l} U_{il} V_{jl} W_{kl} b_j c_k$$

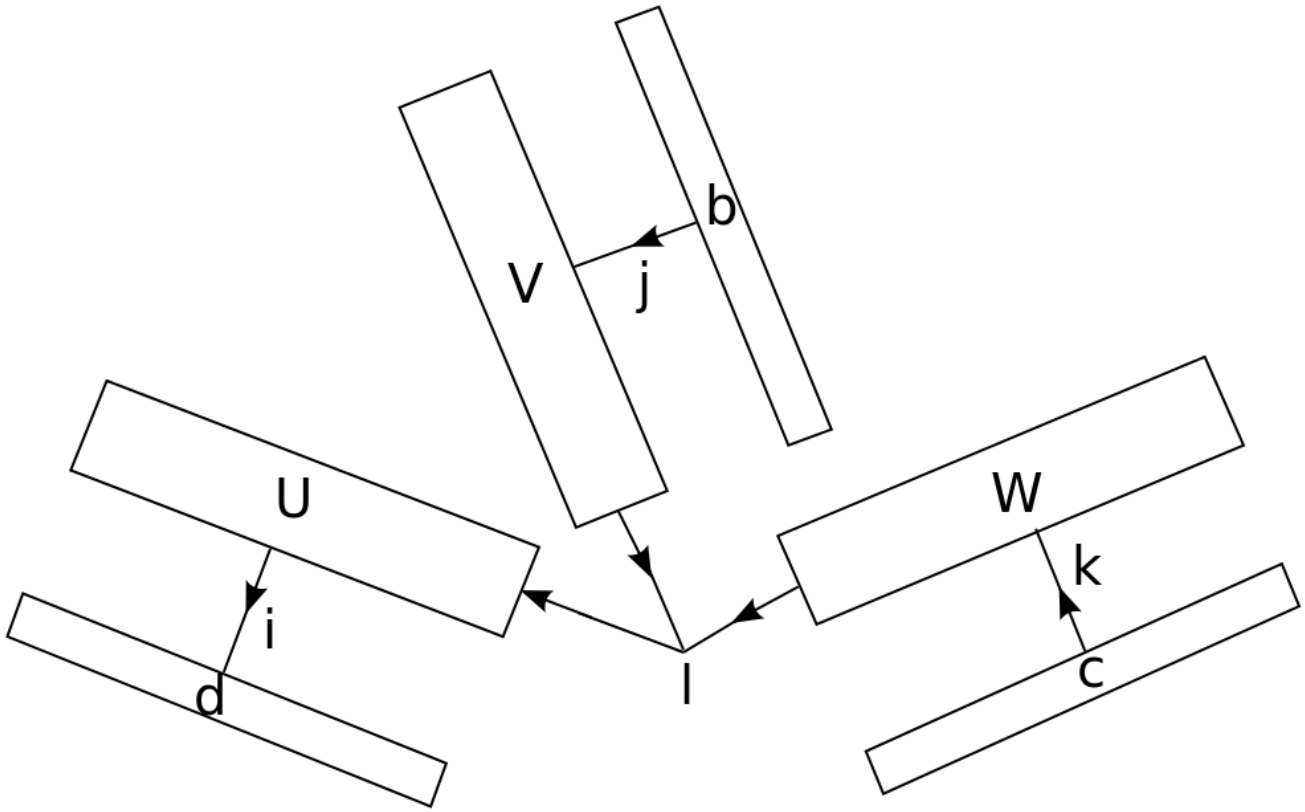
multilinear low rank operator



Low-rank tensor factorizations (CP)

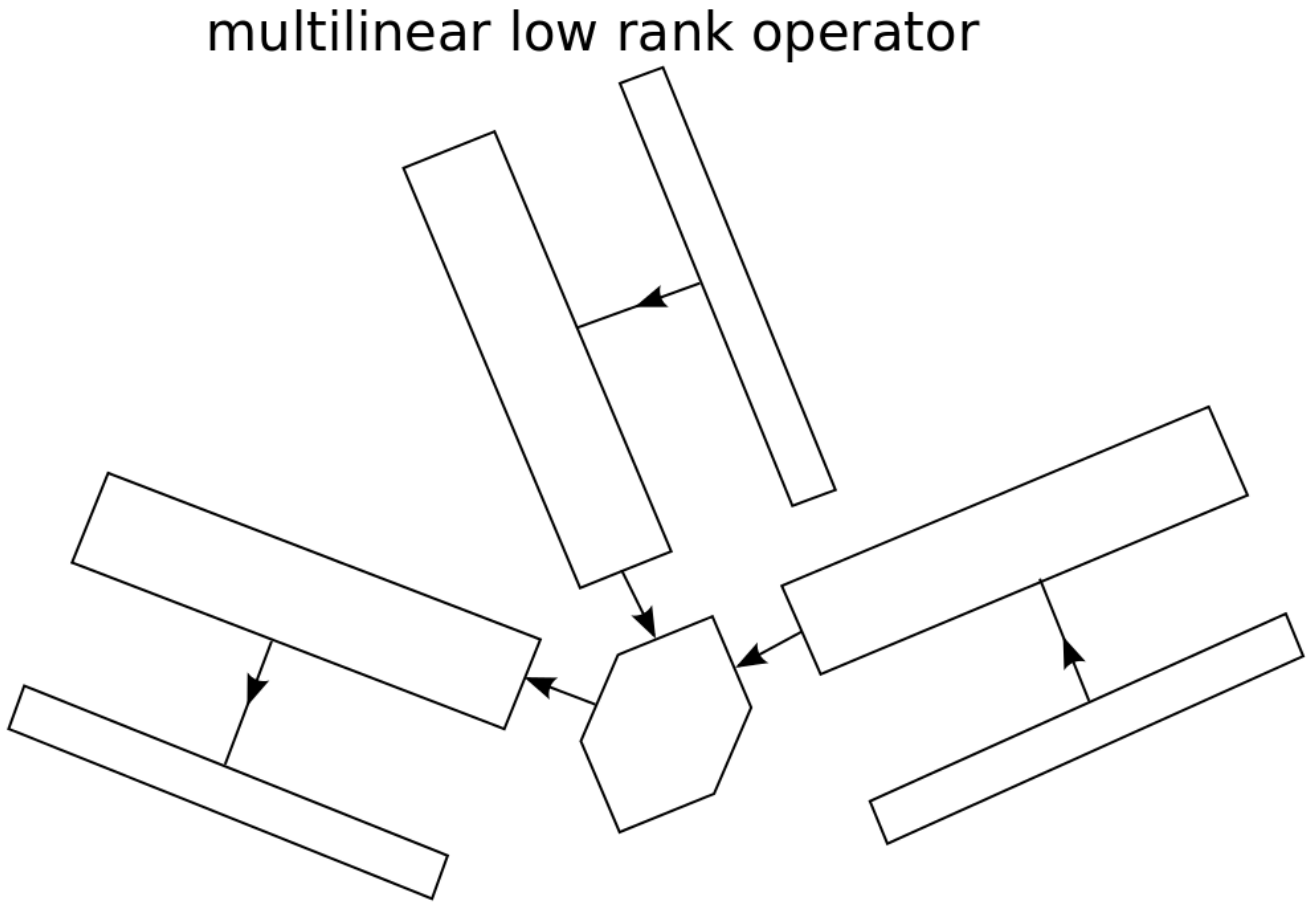
$$A_{ijk} = \sum_{l} U_{il} V_{jl} W_{kl} \quad \rightarrow \quad d_i = \sum_{j,k,l} U_{il} V_{jl} W_{kl} b_j c_k$$

multilinear low rank operator



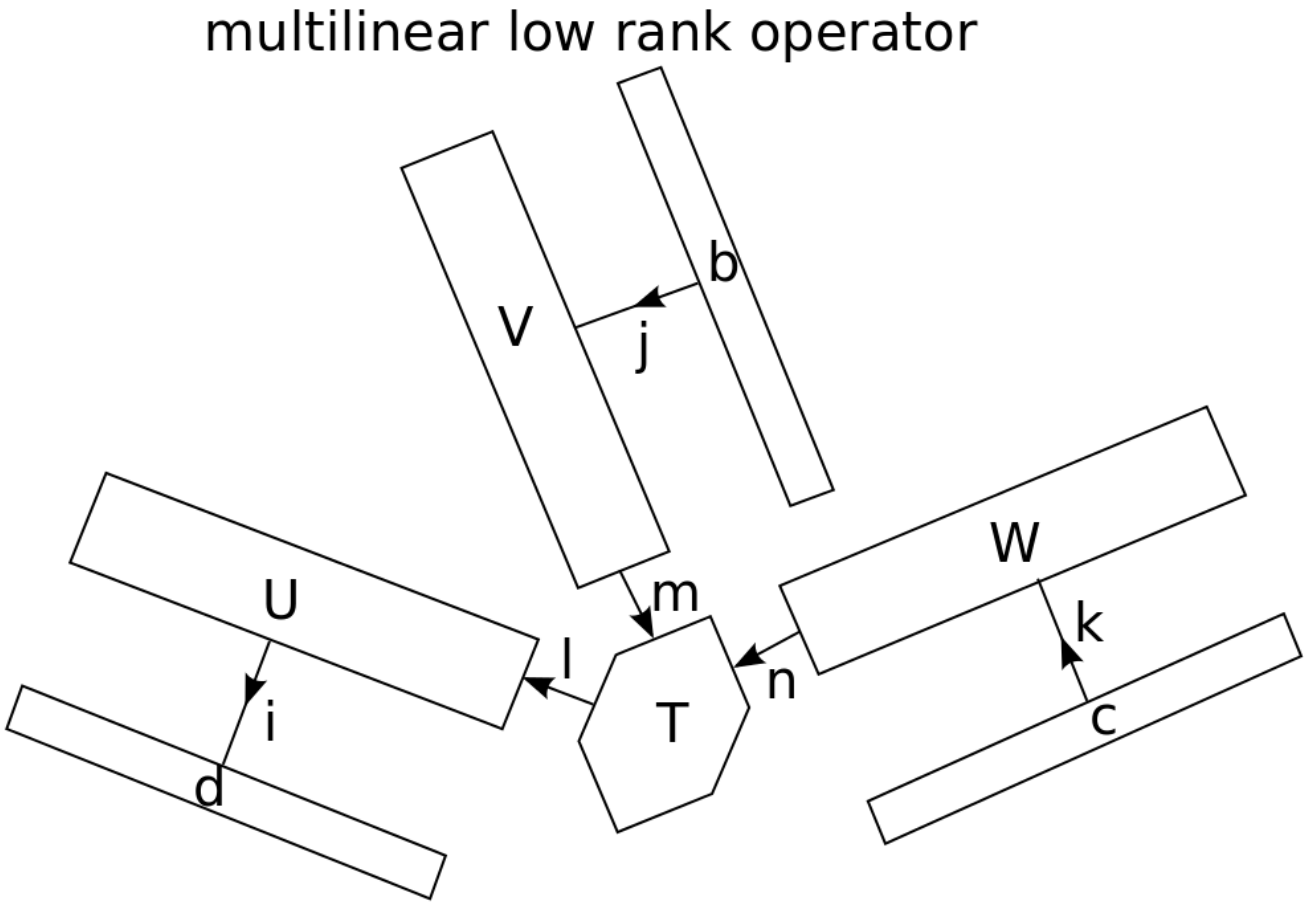
Low-rank tensor factorizations (Tucker)

$$A_{ijk} = \sum_{l,m,n} T_{lmn} U_{il} V^{jm} W^{kn} \quad \rightarrow \quad d_i = \sum_{j,k,l,m,n} T_{lmn} U_{il} V_{jm} W_{jn} b_j c_k$$



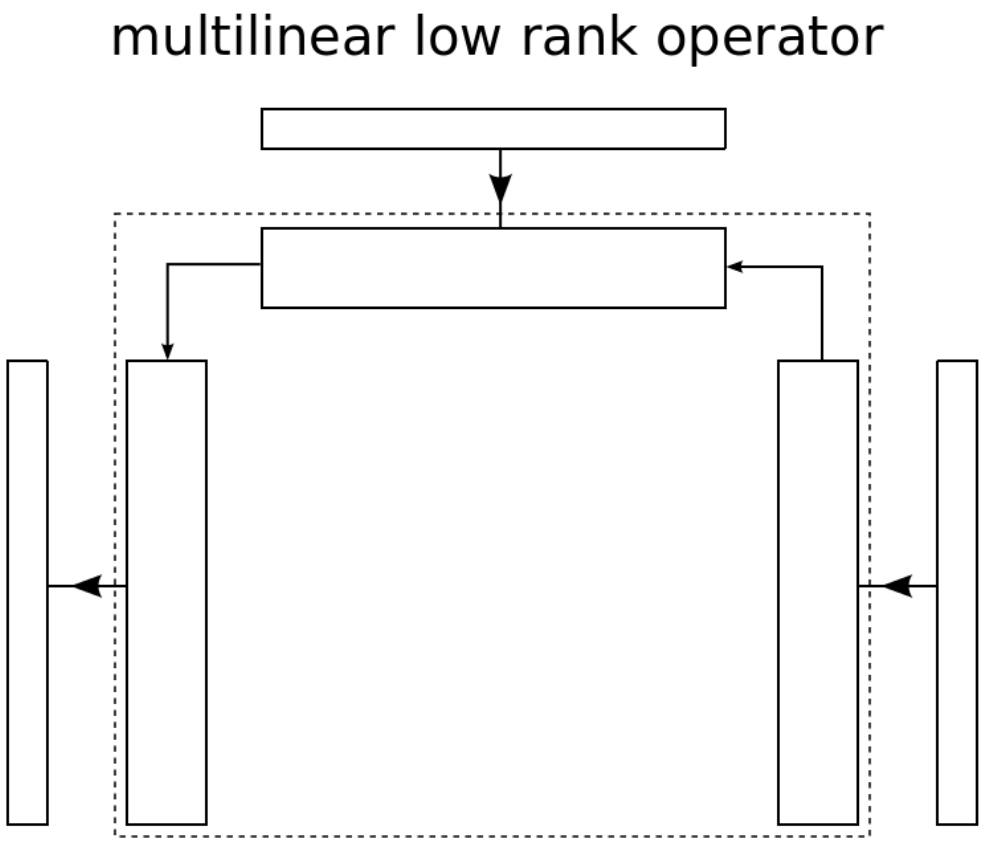
Low-rank tensor factorizations (Tucker)

$$A_i^{jk} = \sum_{l,m,n} T_{lmn} U_{il} V_{jm} W_{kn} \quad \rightarrow \quad d_i = \sum_{j,k,l,m,n} T_{lmn} U_{il} V_{jm} W_{kn} b_j c_k$$



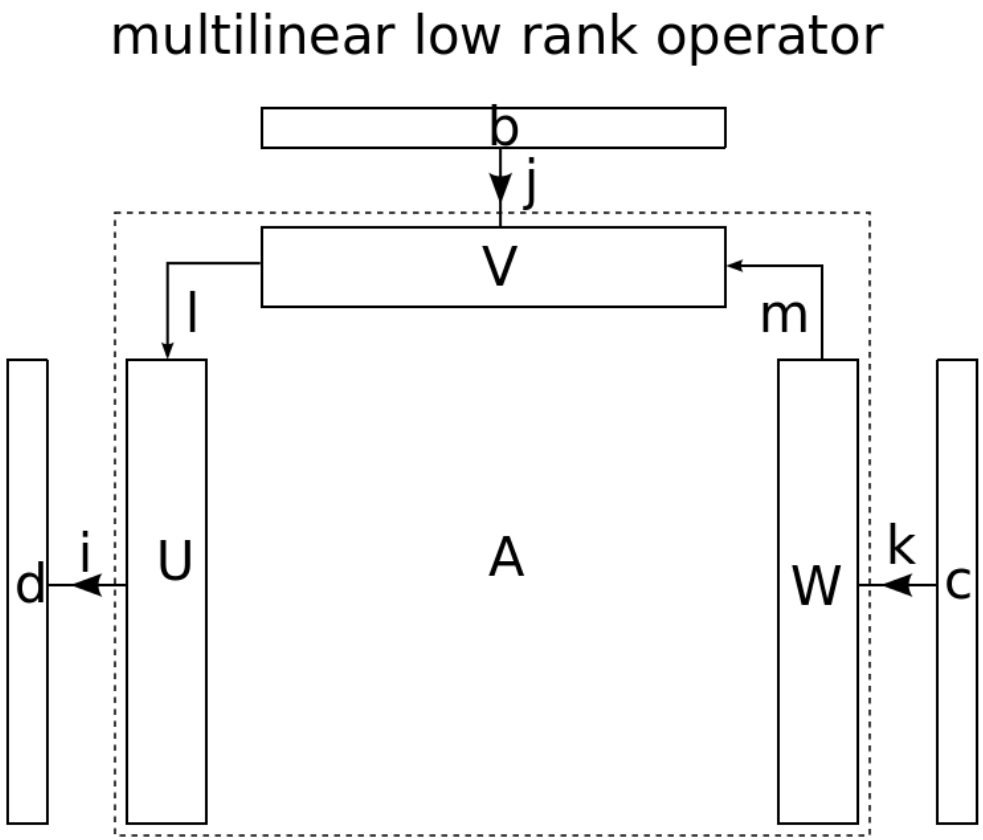
Low-rank tensor factorizations (TT)

$$A_{ijk} = \sum_{l,m} U_{il} V_{ljm} W_{mk} \quad \rightarrow \quad \sum_{j,k,l,m} U_{il} V_{ljm} W_{mk} b_j c_k$$



Low-rank tensor factorizations (TT)

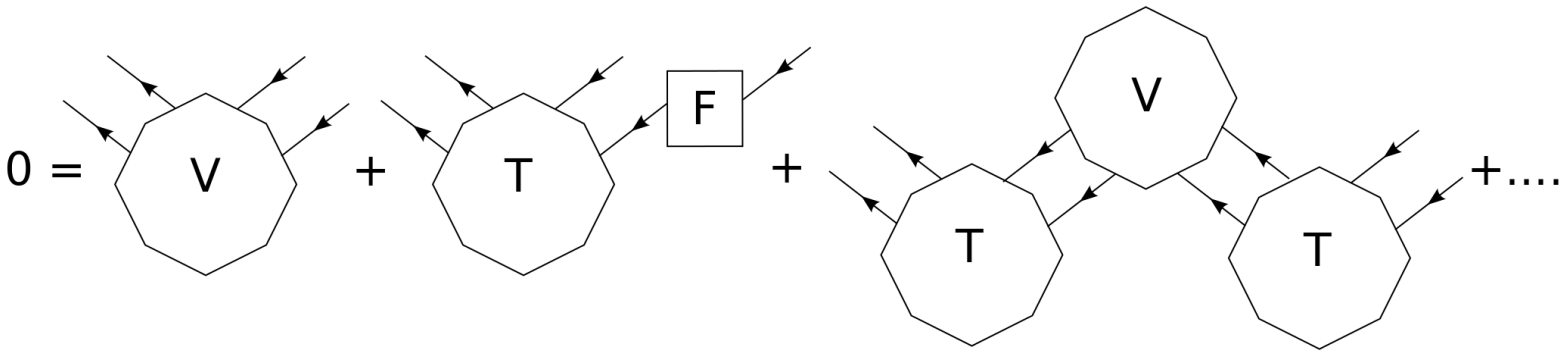
$$A_{ijk} = \sum_{l,m} U_{il} V_{ljm} W_{mk} \quad \rightarrow \quad \sum_{j,k,l,m} U_{il} V_{ljm} W_{mk} b_j c_k$$



Tensor contractions in electronic structure methods

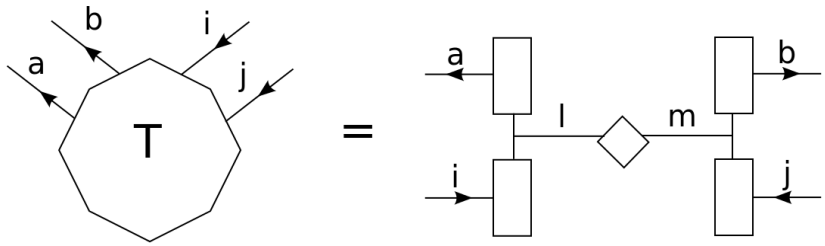
Amplitude equation snippet from coupled cluster doubles model

$$0 = V_{abij} + \sum_{k} T_{abik} F_{kj} + \sum_{k,l,m,n} T_{abkl} V_{klmn} T_{mnij} + \dots$$



Tensor hypercontraction representation[a]

$$T_{abij} = \sum_{l,m} \psi_{al} \phi_{jl} Z_{lm} \phi_{jm} \psi_{bm}$$



[a] Hohenstein, Parrish, Sherrill, Martinez, JCP, 2012

Structured tensor computations

Challenges for matrix/tensor algebraic abstractions

- data and relation sparsity $\rightarrow$ tensor sparsity
- low-order representations of data $\rightarrow$ tensor decompositions
- implicitly defined relations $\rightarrow$ implicit tensor representations
- data and relation equivalences $\rightarrow$ tensor symmetries

Exploiting symmetry in tensors

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

- 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[1]

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

- rank-2 vector outer product[1]

$$C_{ij} = a_i b_j + a_j b_i = (a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j$$

- squaring a symmetric matrix (or $AB + BA$)[1]

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

- for order ω contraction, $\omega!$ fewer multiplications [1]

[1] S., Demmel; Technical Report, ETH Zurich, 2015.

Symmetry preserving algorithms

By exploiting symmetry, we can reduce the number of multiplications at the cost of more additions[2]

- partially symmetric contractions
  - symmetry preserving algorithm can be nested over each index group
  - reduction in multiplications implies reduction in nested calls
  - cost reductions: 1.3 for CCSD, 2.1 for CCSDT
- algorithms generalize to most antisymmetric tensor contractions
- for Hermitian tensors, multiplication cost 3X more than addition
  - 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
- allows blocking of symmetric contractions into smaller (anti)symmetric contractions

[2] S., Demmel; Technical Report, ETH Zurich, 2015.

Beyond computation cost

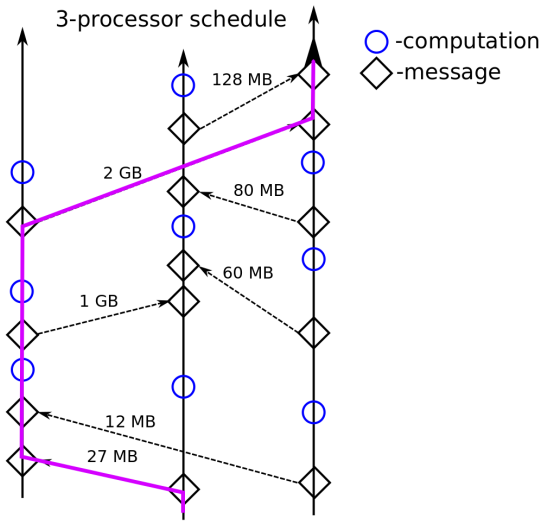
Algorithms should be not only work-efficient but communication-efficient

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

Cost model for parallel algorithms

Given a schedule that specifies all work and communication tasks on p processors, we 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, can hope to obtain $W = O(n^2/\sqrt{cp})$ communication
- standard parallel libraries (ScaLAPACK, Elemental) optimal only for $c = 1$

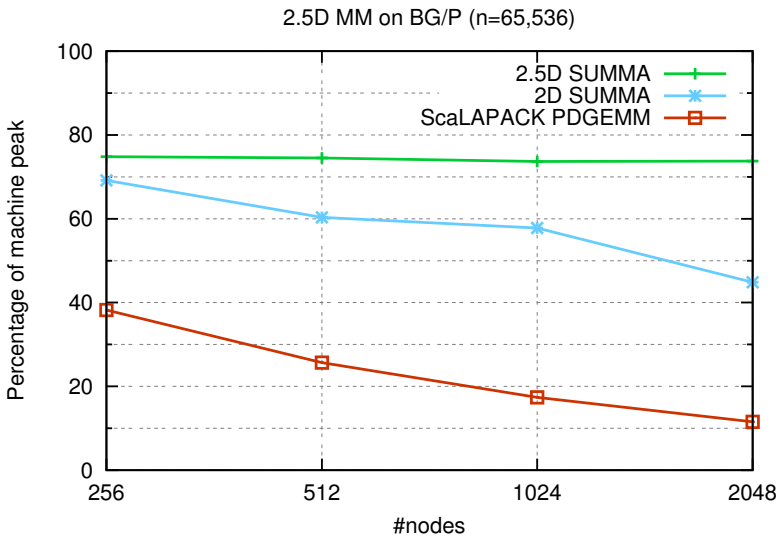
[3] Aggarwal, Chandra, Snir, TCS, 1990

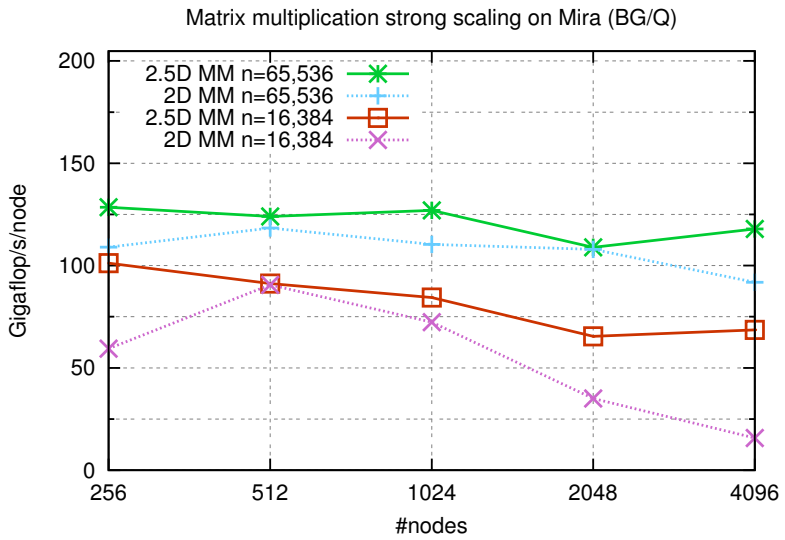
[4] Irony, Toledo, Tiskin, JPDC, 2004

Communication-efficient matrix multiplication

Communication-optimal algorithms for matrix multiplication have been studied extensively[5]

They continue to be attractive on modern architectures[6]





[5] Berntsen, Par. Comp., 1989; Agarwal, Chandra, Snir, TCS, 1990; Agarwal, Balle, Gustavson, Joshi, Palkar, IBM, 1995; McColl, Tiskin, Algorithmica, 1999; ...

[6] S., Bhatele, Demmel, SC, 2011

Synchronization cost lower bounds

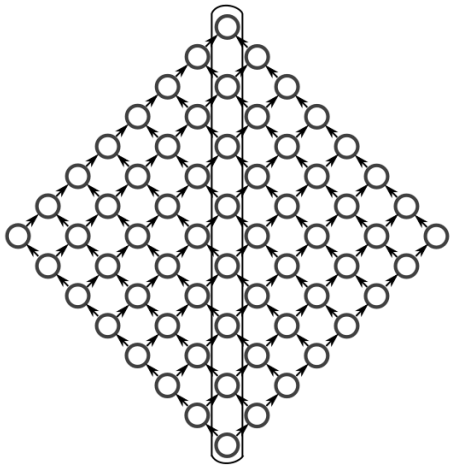
Unlike matrix multiplication, many algorithms in numerical linear algebra have polynomial depth

- synchronization cost bounds[7] $S_{\text{MM}} = \Theta\left(\frac{n^3}{pM^{3/2}}\right)$
- algorithms for Cholesky, LU, QR, SVD have additional dependencies
- lowering computation and communication costs, requires additional synchronization

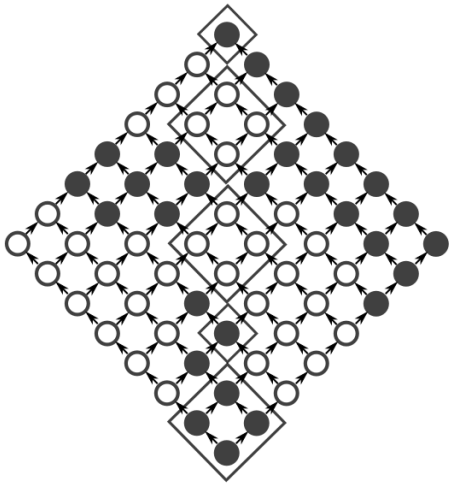
[7] Ballard, Demmel, Holtz, Schwartz, SIAM JMAA, 2011

Tradeoffs in the diamond DAG

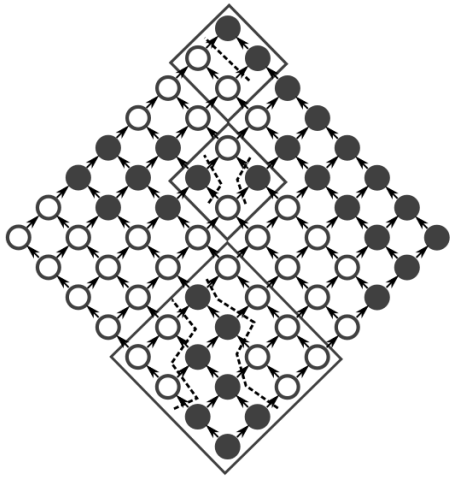
For the $n \times n$ diamond DAG, there is a tradeoff between computation and synchronization costs[8] $F \cdot S = \Omega(n^2)$



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

We generalize such tradeoffs to consider horizontal communication and arbitrary (polynomial or exponential) interval expansion[9]

[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

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

Proof employs classical Loomis-Whitney inequality.

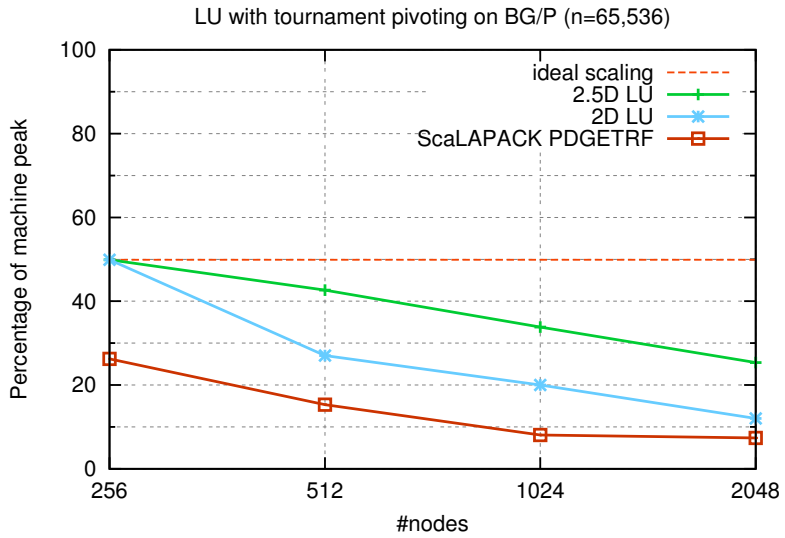
For any $R \subset \mathbb{N} \times \mathbb{N} \times \mathbb{N}$, three projections of R onto $\mathbb{N} \times \mathbb{N}$ have total size at least $|R|^{2/3}$

[a] S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)

Communication-efficient LU factorization

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

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



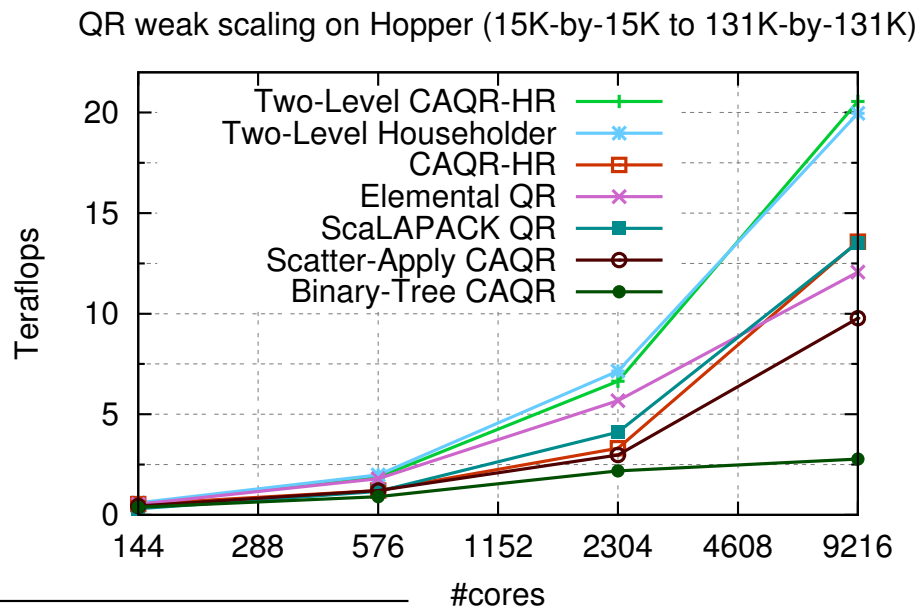
- LU with pairwise pivoting[10] extended to tournament pivoting [11]
- first implementation of a communication-optimal LU algorithm

[10] Tiskin, FGCS, 2007

[11] S., Demmel, Euro-Par, 2011

Communication-efficient QR factorization

- $W_{\mathrm{QR}} = O(n^2/\sqrt{cp}), S_{\mathrm{QR}} = O(\sqrt{cp})$ using Givens rotations[12]
- Householder form can be reconstructed quickly from TSQR[13]
- optimal QR communication and synchronization (modulo log factors) costs can be obtained with Householder representation[14]
- Householder aggregation yields performance improvements



[12] Tiskin, FGCS, 2007

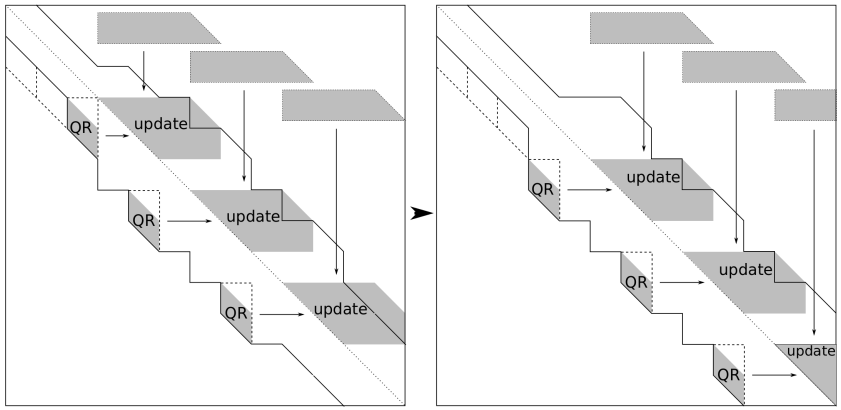
[13] Ballard, Demmel, Grigori, Jacquelin, Nguyen, Diep, S., IPDPS, 2014

[14] S., UCB, 2014

Communication-efficient eigenvalue computation

For the dense symmetric matrix eigenvalue problem

- $W_{\text{SE}} = O(n^2/\sqrt{cp}), S_{\text{QR}} = O(\sqrt{cp}\log^2 p)$ [15]
- optimal horizontal communication can be obtained with left-looking algorithm and aggregation, however, requires more vertical communication
- successive band reduction can be used to minimize both communication costs



[15] S., UCB, 2014. S., Hoefler, Demmel, in preparation

Synchronization tradeoffs in stencils

Our lower bound analysis extends also to sparse iterative methods.[16]
For computing s applications of a $(2m+1)^d$-point stencil

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

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

[16] S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)

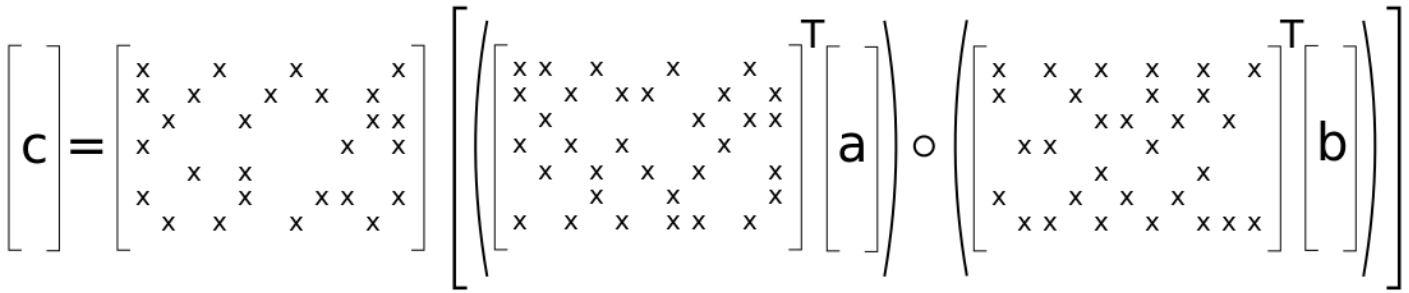
Beyond the Loomis-Whitney inequalities

Loomis-Whitney inequalities are not sufficient for all computations

- symmetry preserving tensor contraction algorithms have arbitrary order projections from order d set
- bilinear algorithms[17] provide a more general framework
- a bilinear algorithm is defined by matrices $F^{(A)}, F^{(B)}, F^{(C)}$,

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

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



- communication lower bounds can be formulated in terms of rank[18]

[17] Pan, Springer, 1984

[18] S., Hoefler, Demmel, in preparation

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[a] (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	$\frac{n^6}{8}$	$\frac{n^6}{720}$	$\frac{n^6}{H^{1/2}}$	$\frac{n^6}{H^{1/2}}$	$\frac{n^4}{p^{2/3}}$	$\frac{n^4}{p^{2/3}}$	$\frac{n^4}{p^{2/3}}$

[a] S., Hoefler, Demmel, ETHZ, 2014

Open theoretical problems

- lower bounds for multiplication of a sparse and a dense matrix
- lower bounds for nested bilinear algorithms
- broader parameterizations of algorithmic representations needed for QR and SVD lower bounds

Tensor algebra as a programming abstraction

Cyclops Tensor Framework[19]

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

[19] S., Hammond, Demmel, UCB, 2011. S., Matthews, Hammond, Demmel, IPDPS, 2013

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

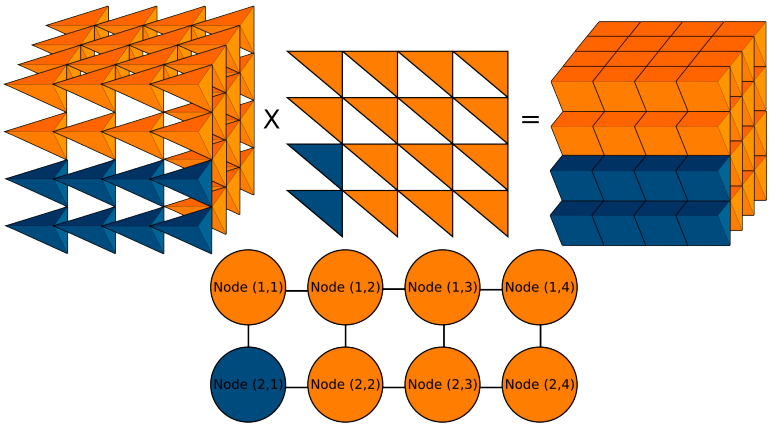
Betweenness centrality code snippet, for k of n nodes

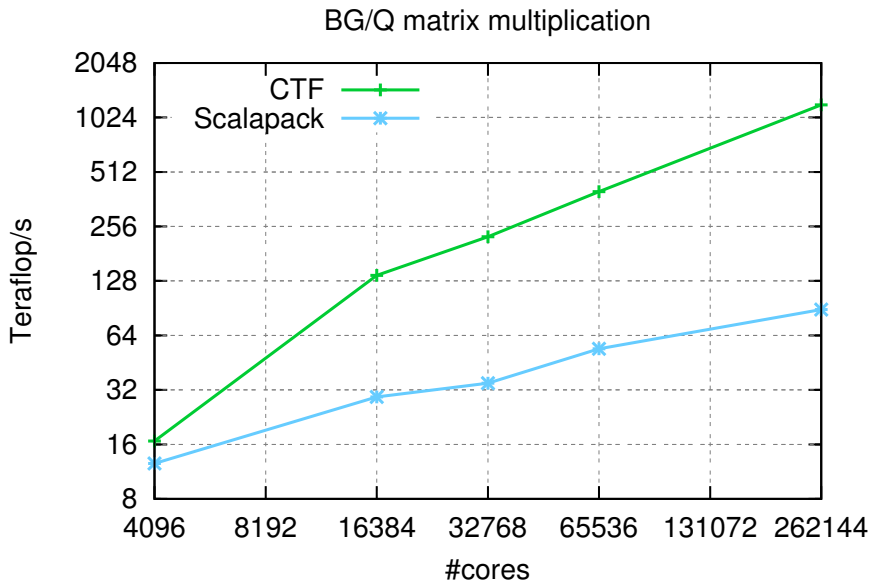
```
void btwn_central(Matrix<int> A, Matrix<path> P, int n, int k)
  Monoid<path> mon(...,
                   [](path a, path b){
                     if (a.w<b.w) return a;
                     else if (b.w<a.w) return b;
                     else return path(a.w, a.m+b.m);
                   }, ...);

  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

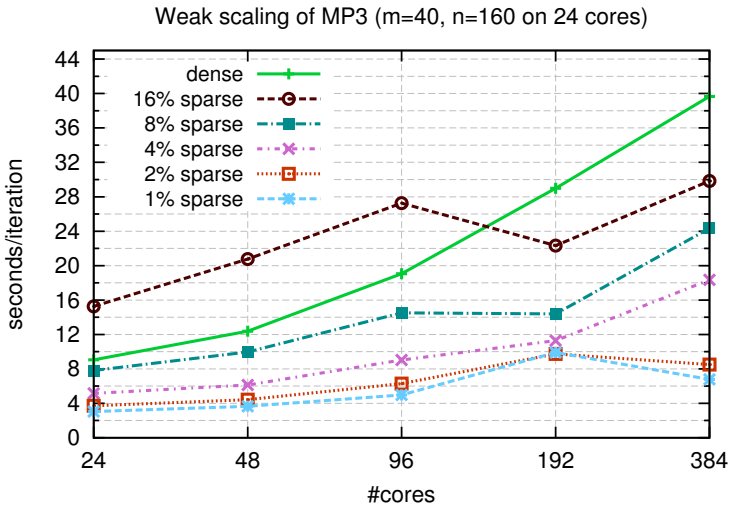
- virtualized multidimensional processor grids
- topology-aware mapping and collective communication
- performance-model-driven decomposition done at runtime
- optimized redistribution kernels for tensor transposition

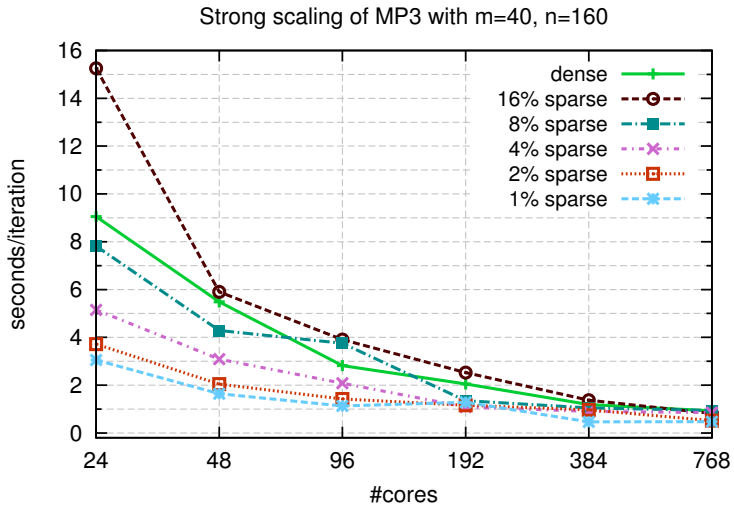




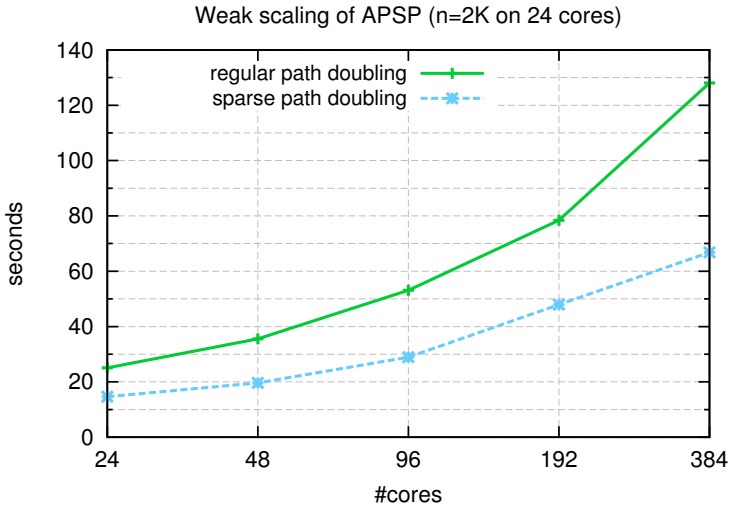
Performance of CTF for sparse computations

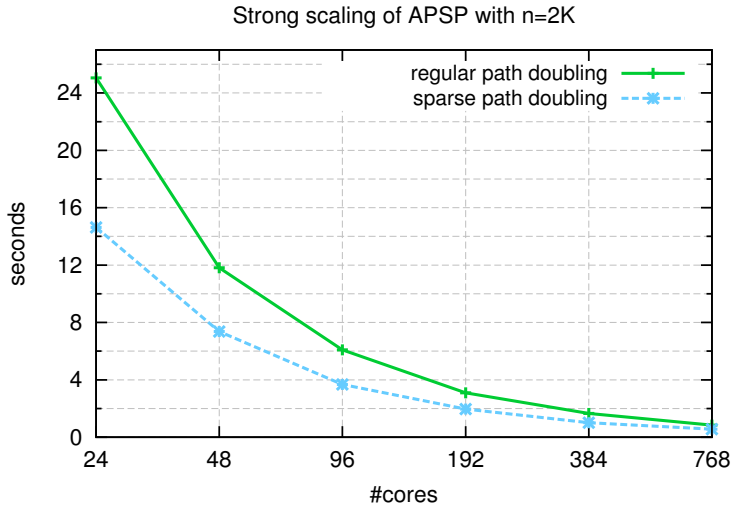
MP3 leveraging sparse-dense tensor contractions[a]





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

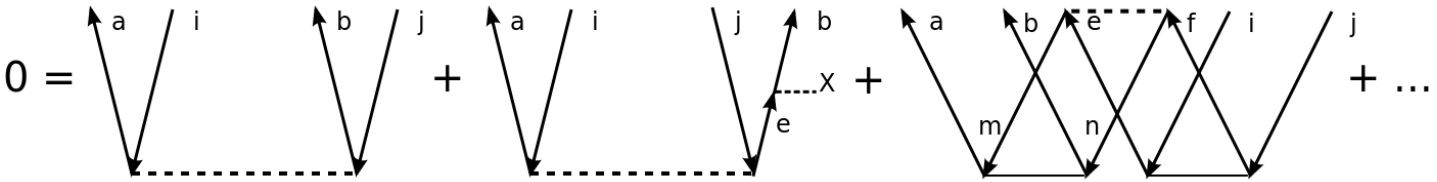




Post-Hartree-Fock (HF) methods

Accurate models of electronic correlation require approximation of contributions of excited-state transitions.

- Møller-Plesset methods provide perturbative corrections
- Coupled-cluster methods (CCSD, CCSDT, CCSDTQ) iteratively solve (2nd, 3rd, 4th) order equations in the state space



- tensor expressions naturally express high-order transitions
- tensor structure admits symmetries and sparsity
  - permutational index antisymmetry due to antisymmetry of wavefunction
  - sparsity due to strength of interactions diminishing with growing distance in the molecular orbital basis

CCSD using CTF

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

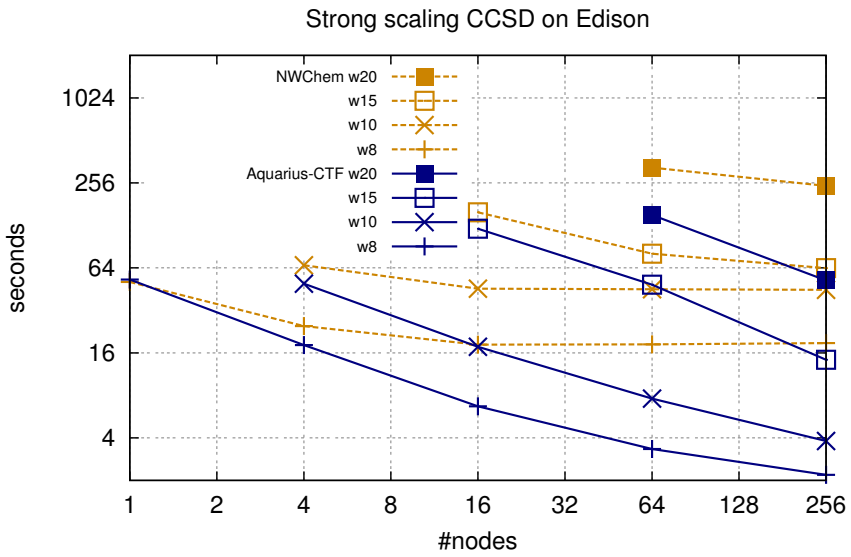
```
FMI["mi"]     += 0.5*WMNEF["mnef"]*T(2)["efin"];
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T(2)["efij"];
FAE["ae"]     -= 0.5*WMNEF["mnef"]*T(2)["afmn"];
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T(2)["afin"];

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"];
```

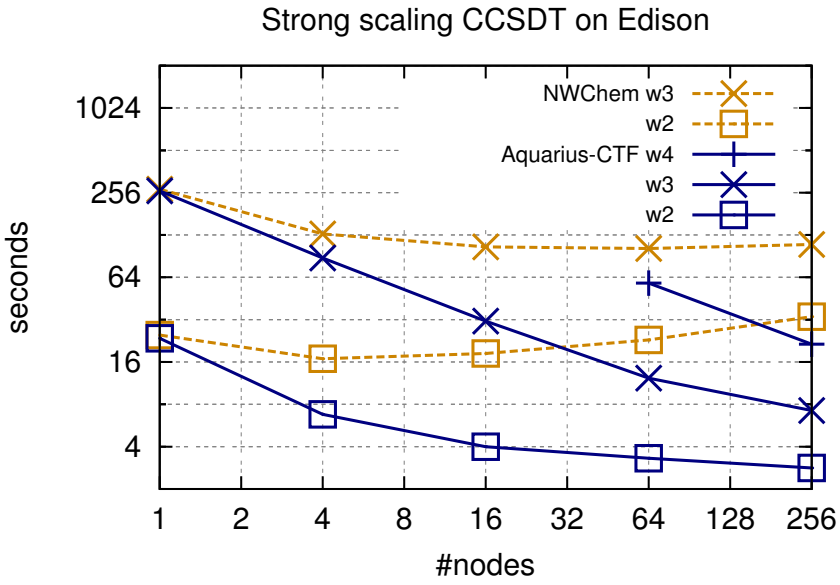
Other electronic structure codes using CTF include QChem (via Libtensor) and VASP

Comparison with NWChem

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

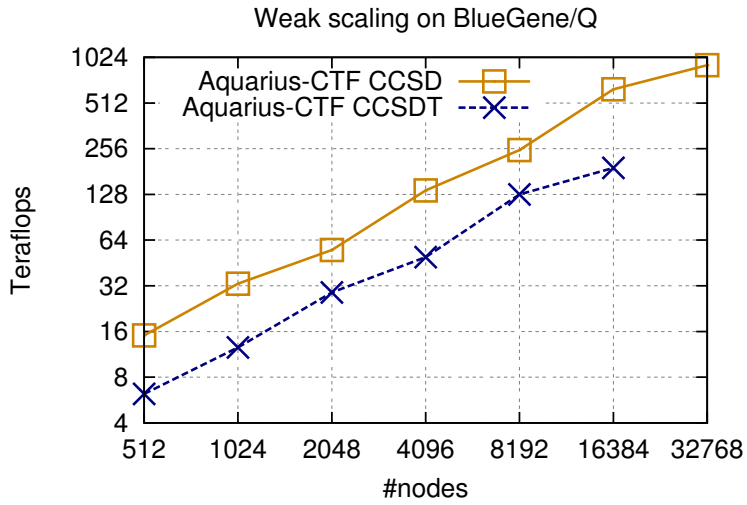
- provides CCSD and CCSDT
- uses Global Arrays a Partitioned Global Address Space (PGAS) backend for tensor contractions
- derives equations via Tensor Contraction Engine (TCE)

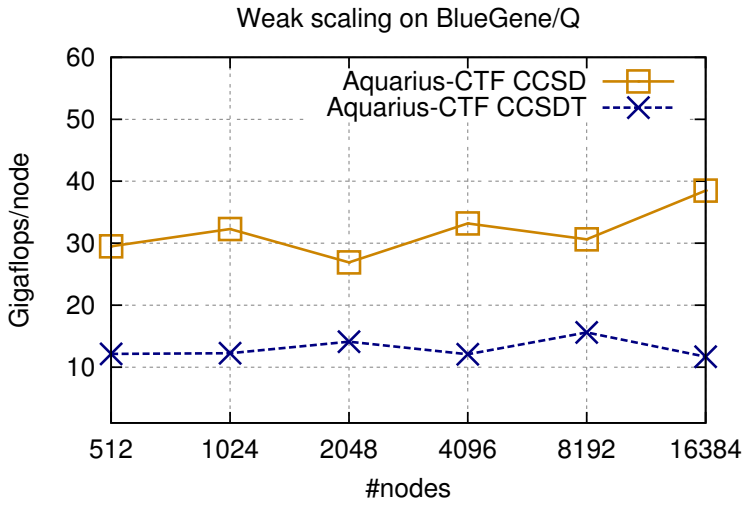


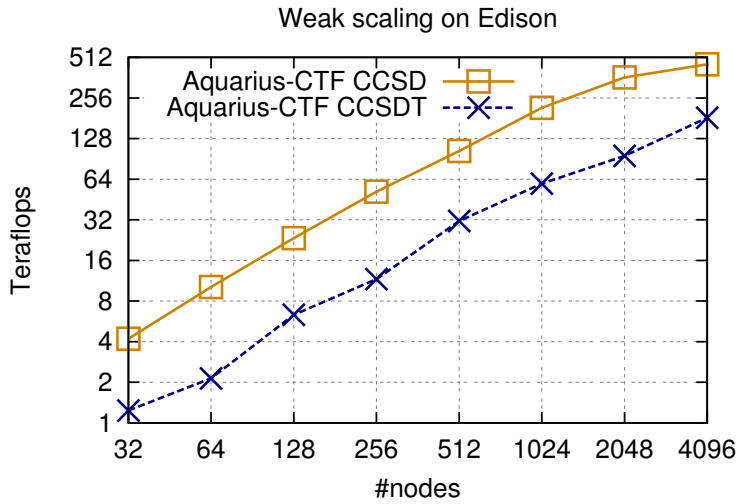


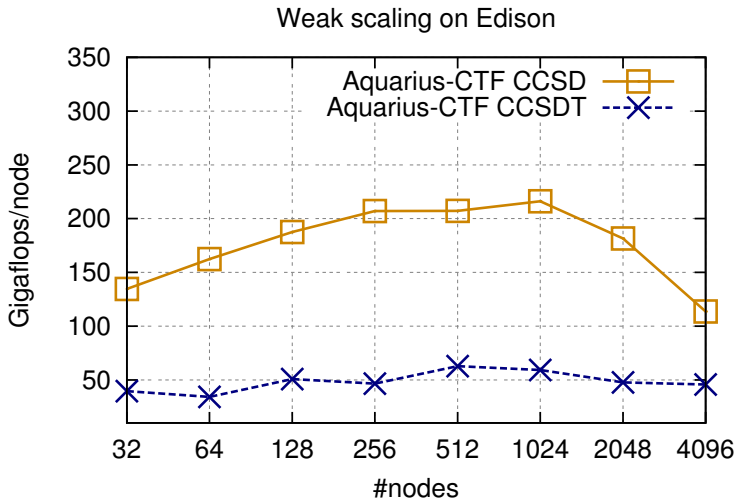
Coupled cluster on IBM BlueGene/Q and Cray XC30

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







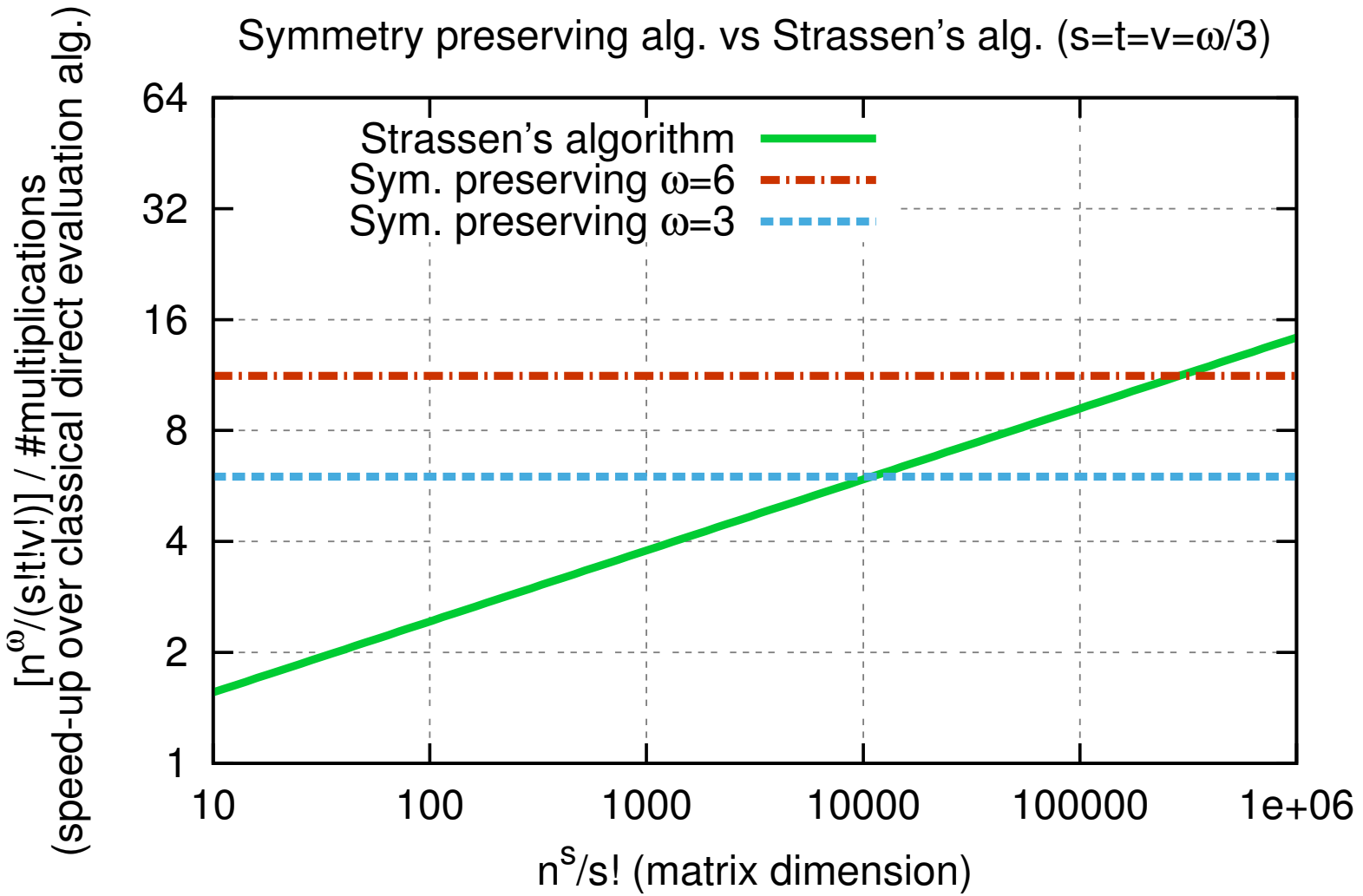


Future work

- symmetry preserving algorithms
  - high-performance implementations
  - bilinear algorithm complexity – fast matrix multiplication
- sparsity in tensor computations
  - handling multiple sparse operands and sparse output
  - worst-case lower bounds and efficient algorithms
- tensor algorithms
  - most algorithms correspond to multiple dependent tensors operations
  - scheduling, blocking, and decomposition of multiple tensor operations
  - programming abstractions for tensor factorizations
- application-driven development
  - tensor decompositions, sparsity, symmetry all motivated by electronic structure applications
  - many further applications in tensor networks (DMRG), machine learning, etc.

Backup slides

Symmetry preserving algorithm vs Strassen's algorithm



Nesting of bilinear algorithms

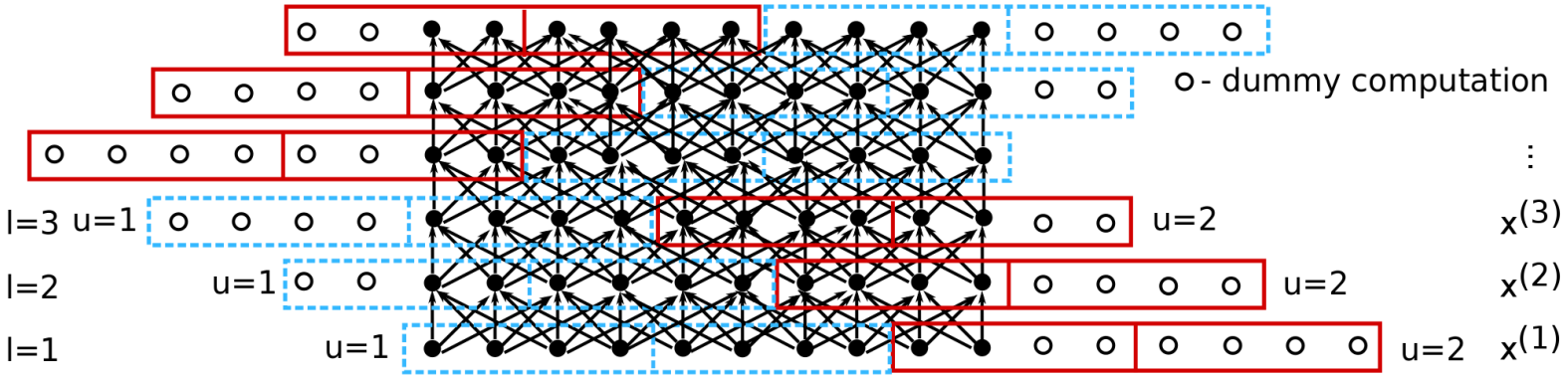
Given two bilinear algorithms:

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

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

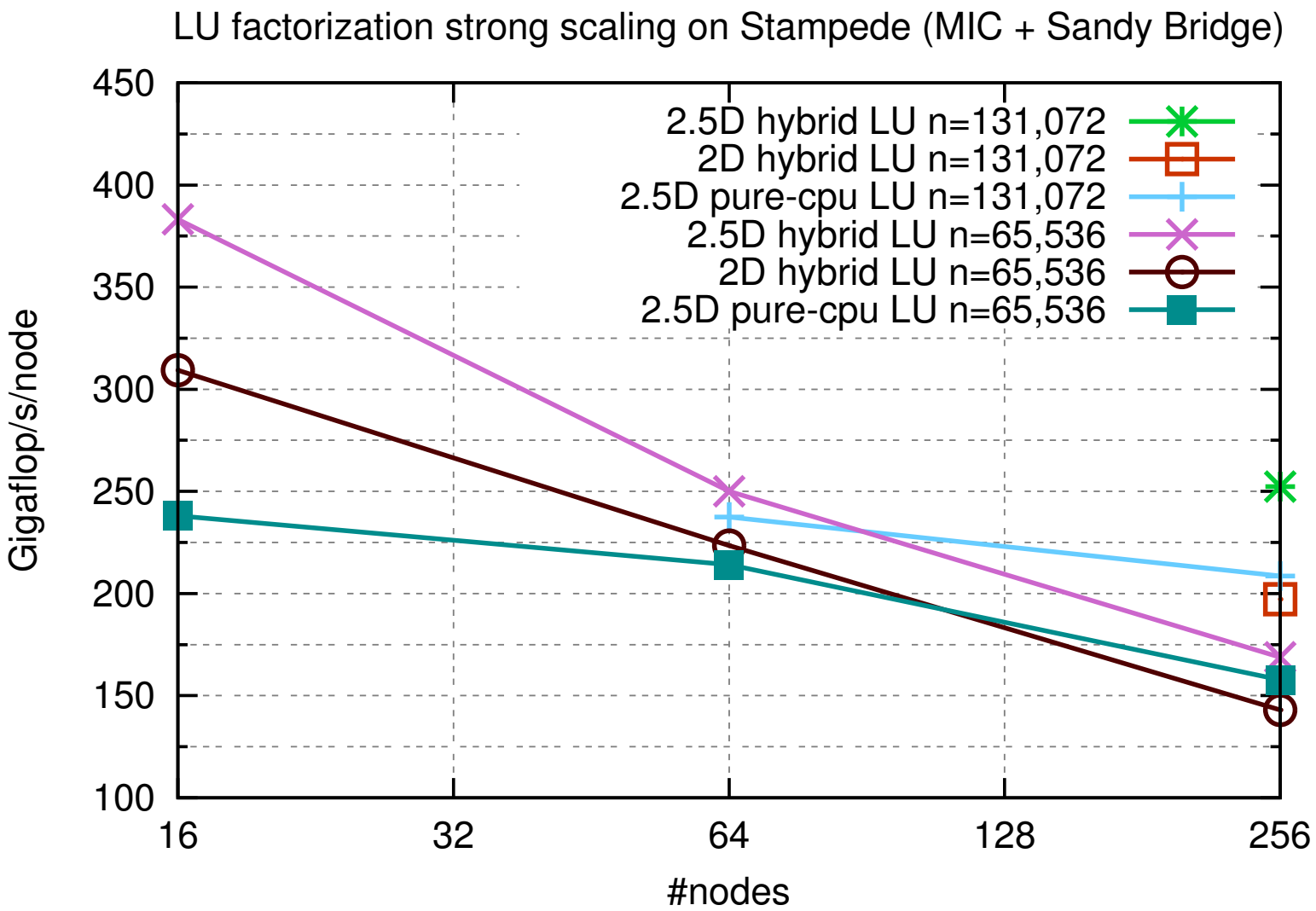


For s-steps of a $(2m+1)^d$-point stencil with block-size of $H^{1/d}/m$,

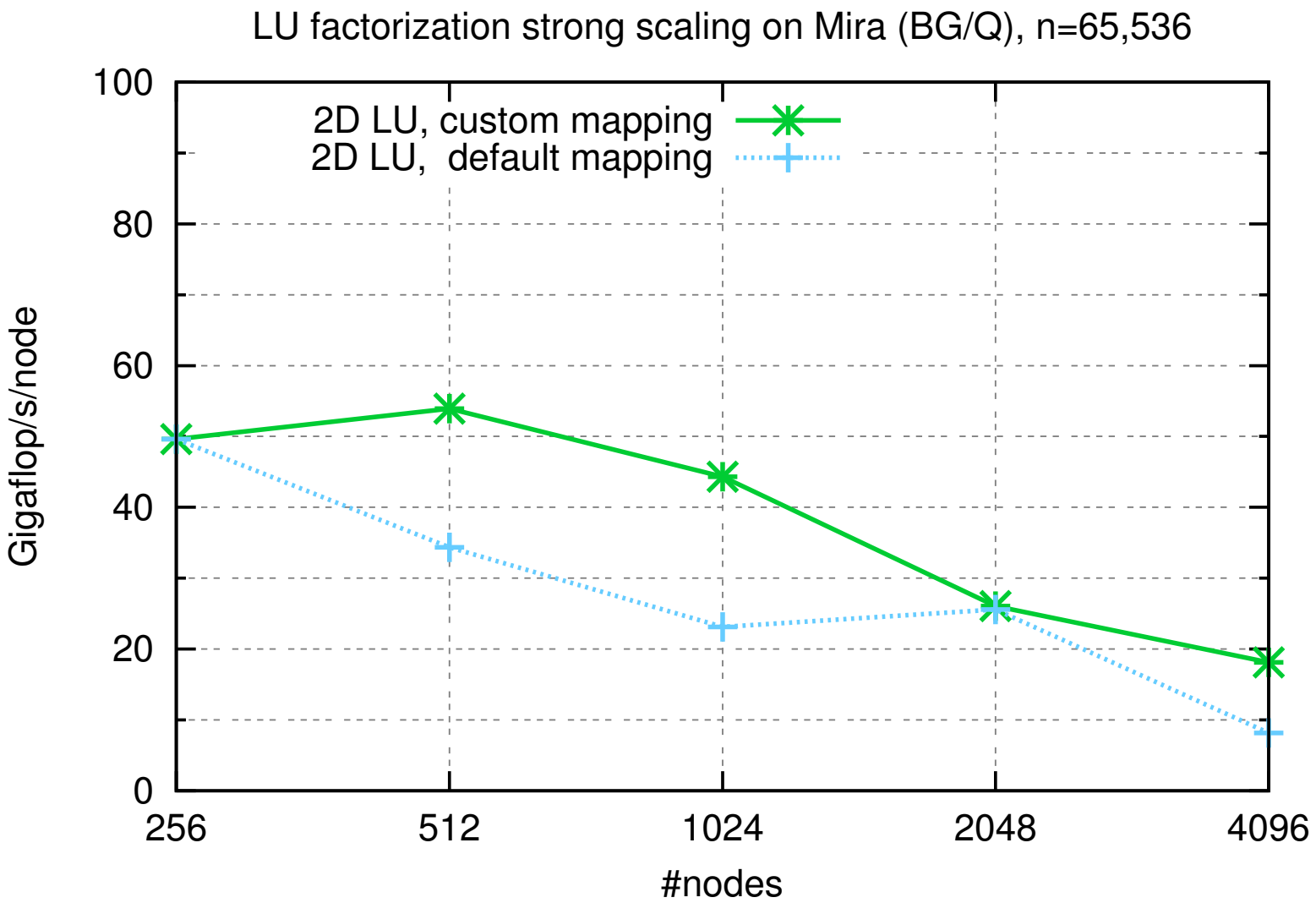
$$W_{\mathrm{Kr}} = O\left(\frac{msn^d}{H^{1/d}p}\right) \quad S_{\mathrm{Kr}} = O(sn^d/(pH)) \quad Q_{\mathrm{Kr}} = O\left(\frac{msn^d}{H^{1/d}p}\right)$$

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

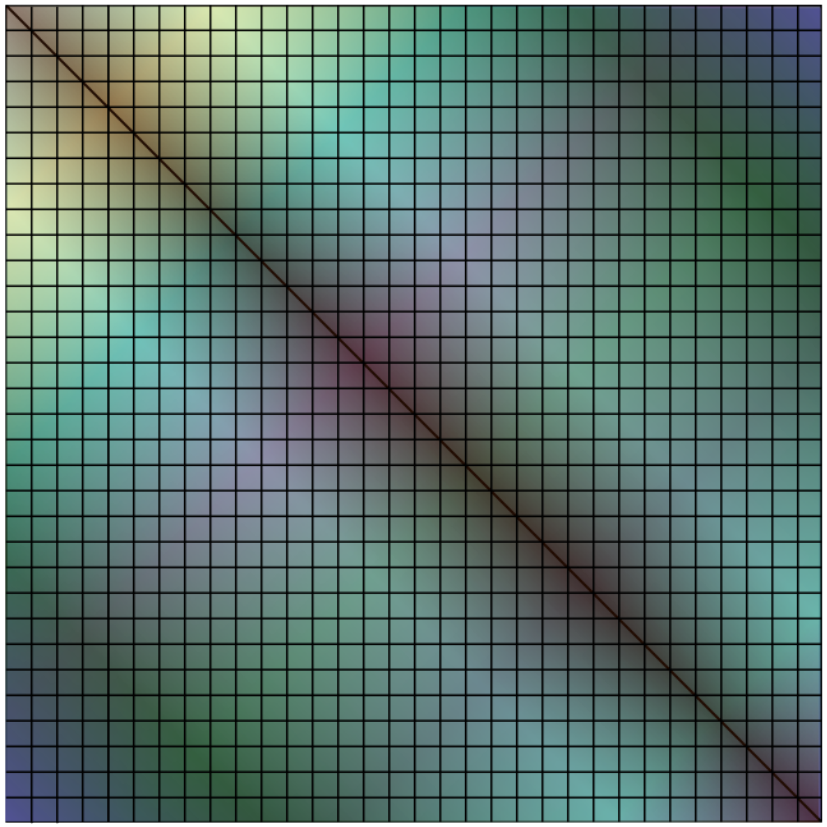

LU factorization strong scaling on Stampede (MIC + Sandy Bridge)
Gigaflop/s/node
#nodes
450
400
350
300
250
200
150
100
16
32
64
128
256
2.5D hybrid LU n=131,072
2D hybrid LU n=131,072
2.5D pure-cpu LU n=131,072
2.5D hybrid LU n=65,536
2D hybrid LU n=65,536
2.5D pure-cpu LU n=65,536

Topology-aware mapping on BG/Q

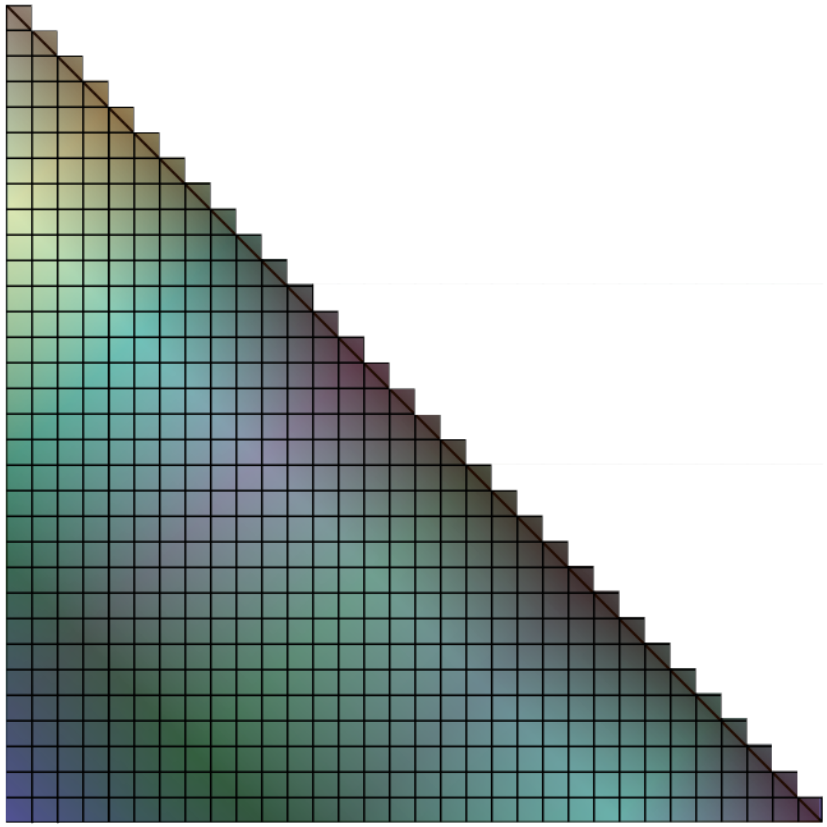


Symmetric matrix representation

Symmetric matrix

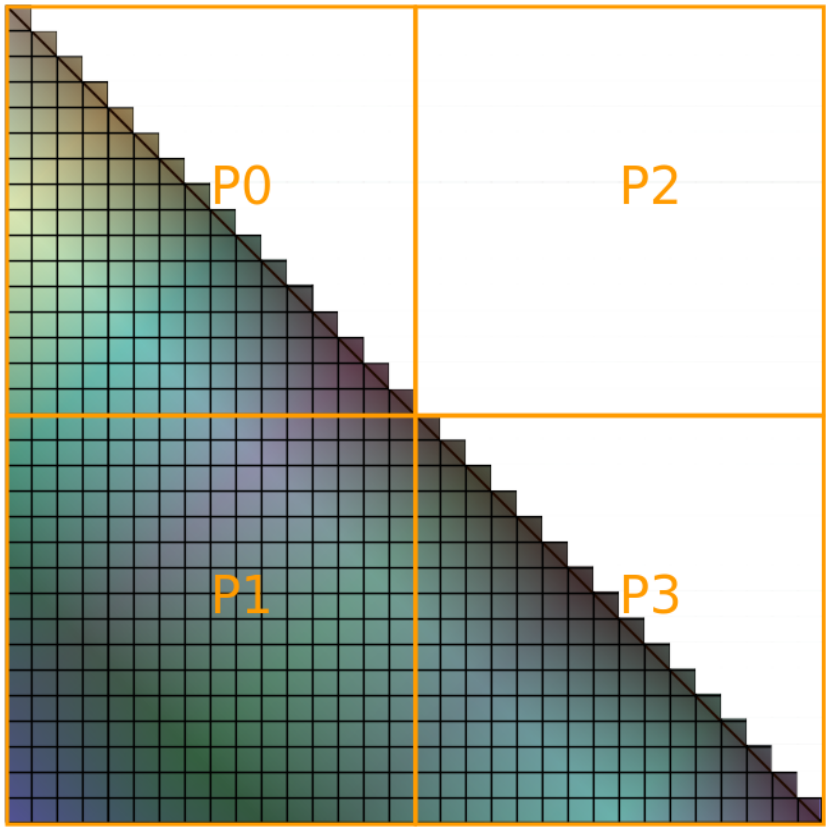


Unique part of symmetric matrix

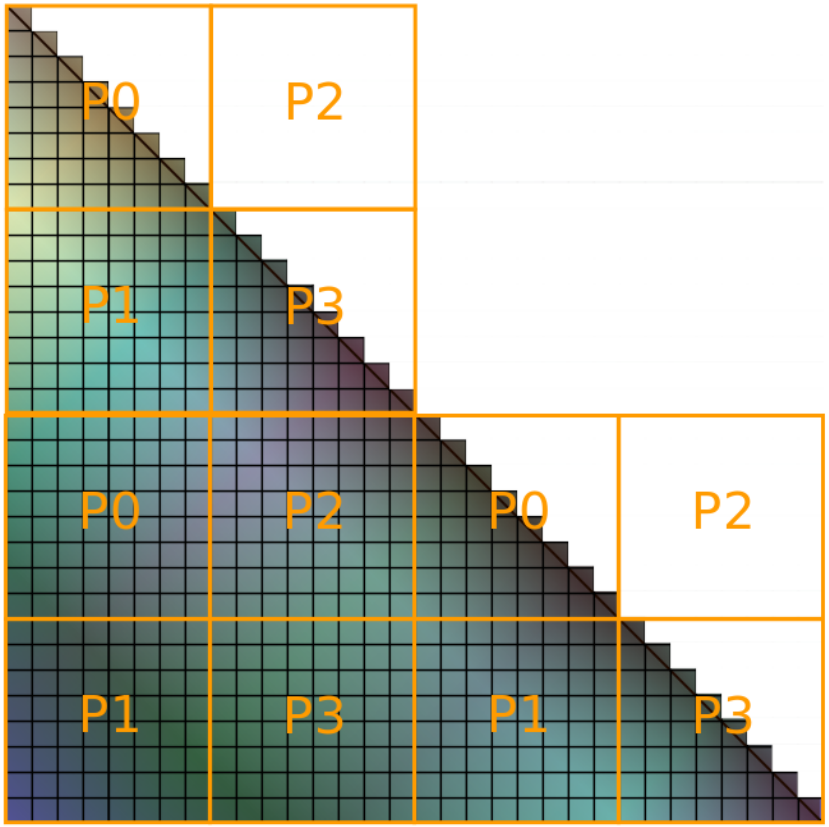


Blocked distributions of a symmetric matrix

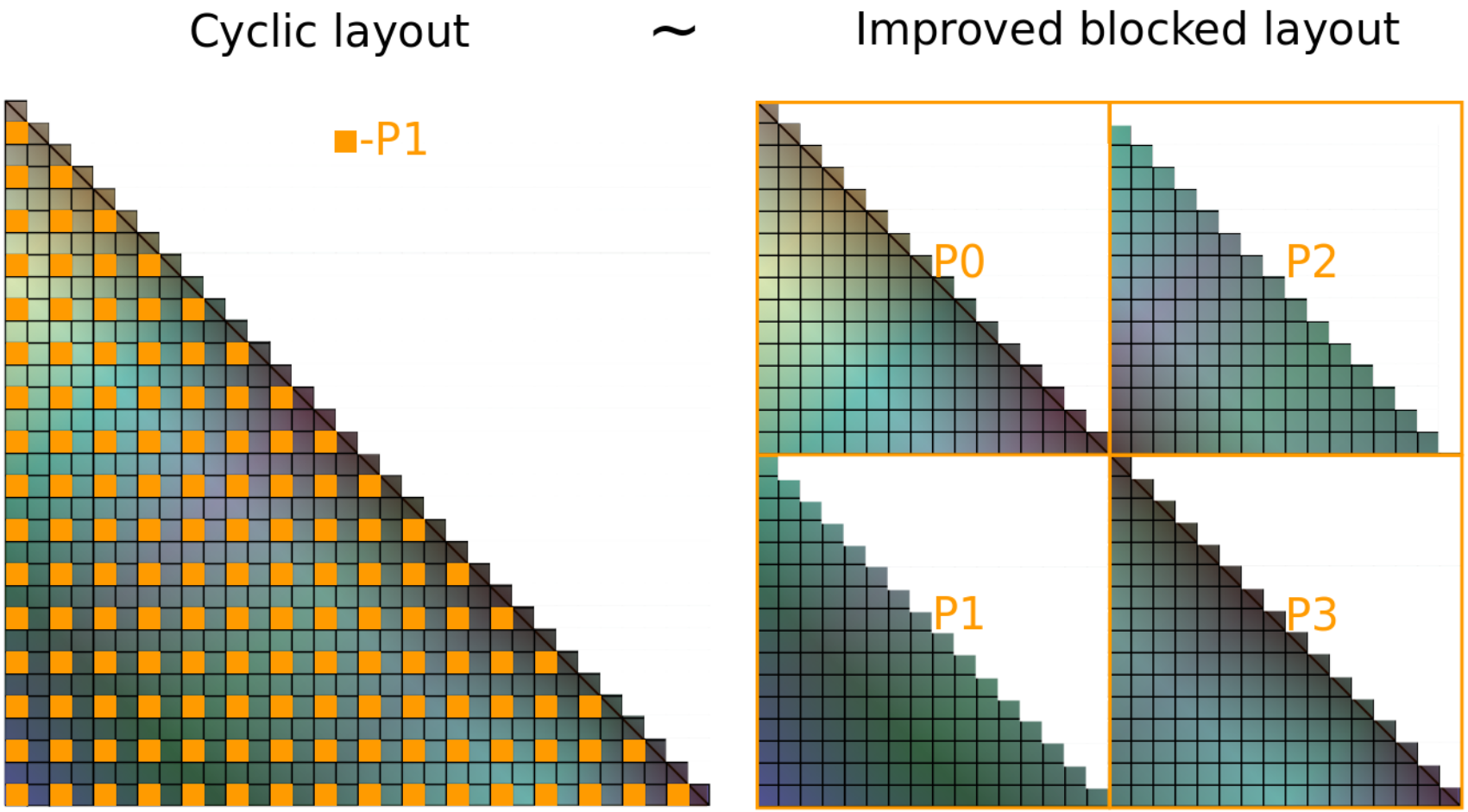
Naive blocked layout



Block-cyclic layout



Cyclic distribution of a symmetric matrix



Our CCSD factorization

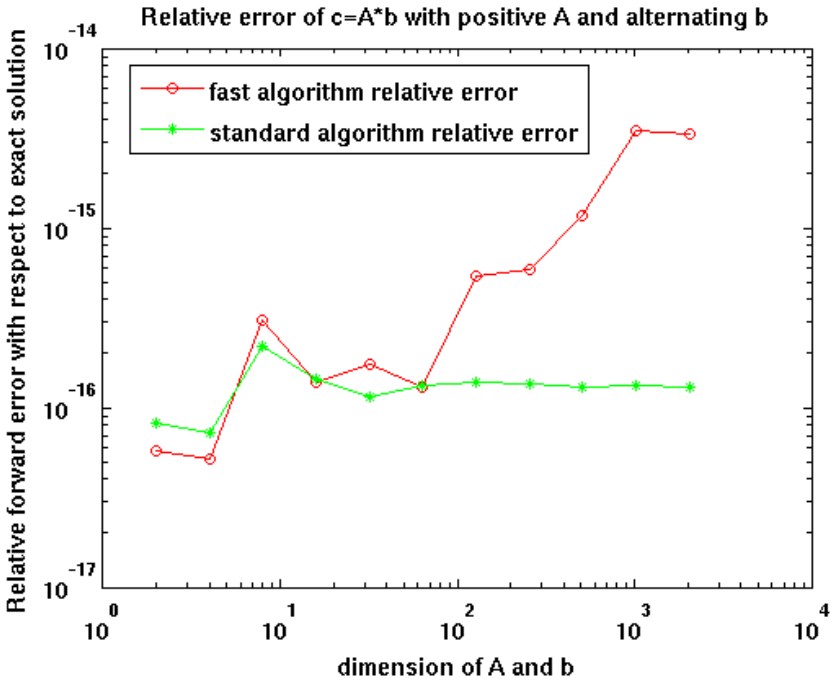
Credit to John F. Stanton and Jurgen Gauss

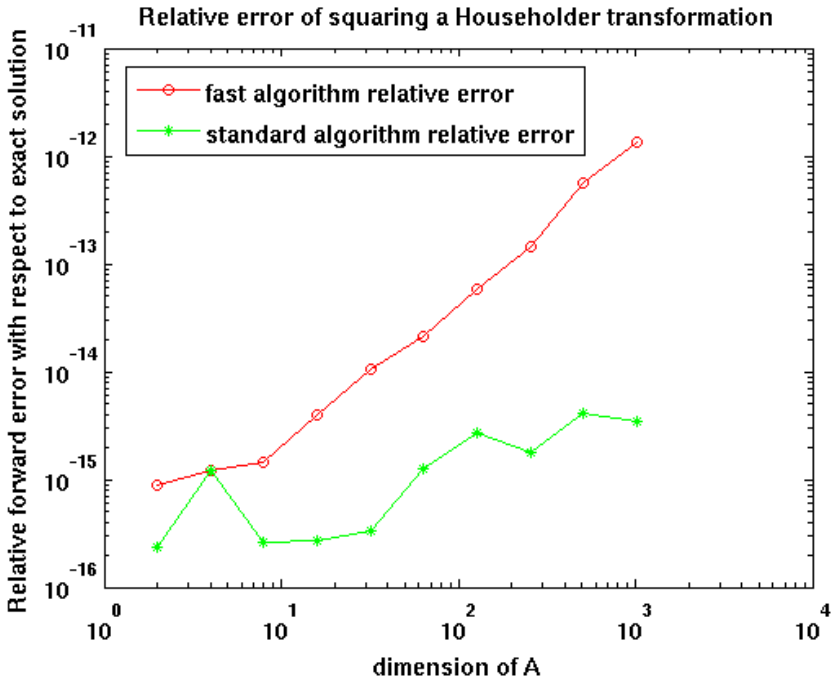
$$
\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\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

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.