Provably efficient algorithms for tensor computations

Edgar Solomonik

ETH Zurich

EPFL MATHICSE Seminar

Nov 11, 2015

Outline

Representation of algorithms

How do we represent numerical algorithms?

- as dependency graphs families
  - direct acyclic graphs, in-degree up to two
  - vertices are inputs/intermediates/outputs
  - dependency hypergraphs provide more generality
- as 'bilinear algorithms':

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

  where
  - g is pointwise bilinear function
  - vectors $\mathbf{a}$ and $\mathbf{b}$ contain inputs, $\mathbf{c}$ contains outputs
  - matrices $\mathbf{F}^{(\mathbf{A})}$, $\mathbf{F}^{(\mathbf{B})}$, and $\mathbf{F}^{(\mathbf{C})}$ encode the dependencies

Algorithmic cost model

Given an algorithm, find a bound (ψ, ϕ) on

$$T\text{–minimum time to execute algorithm on } p \text{ processors}$$

in terms of the quantities

- γ – time per CPU cycle (unit computation)
- β – time per transfer of byte between two processors
- α – time per synchronization of two processors
- ν – time per transfer of byte between cache and memory

via the expression

$$T = \Theta\left(F \cdot \gamma + W \cdot \beta + S \cdot \alpha + Q \cdot \nu \right)$$

with respect to p, problem parameters $\mathbf{n}$, and cache size H

$$\psi(F, W, S, Q) = \Theta(\phi(p, \mathbf{n}, H))$$

Matrix multiplication

For $\mathbf{A} \in \mathbb{R}^{m \times k}$, $\mathbf{B} \in \mathbb{R}^{k \times n}$, compute $\mathbf{C} = \mathbf{AB}$.

- vertical communication bound

$$Q_{\mathrm{MM}} = \Theta\left(\frac{mnk}{p\sqrt{H}} + mk + kn + mn\right)$$

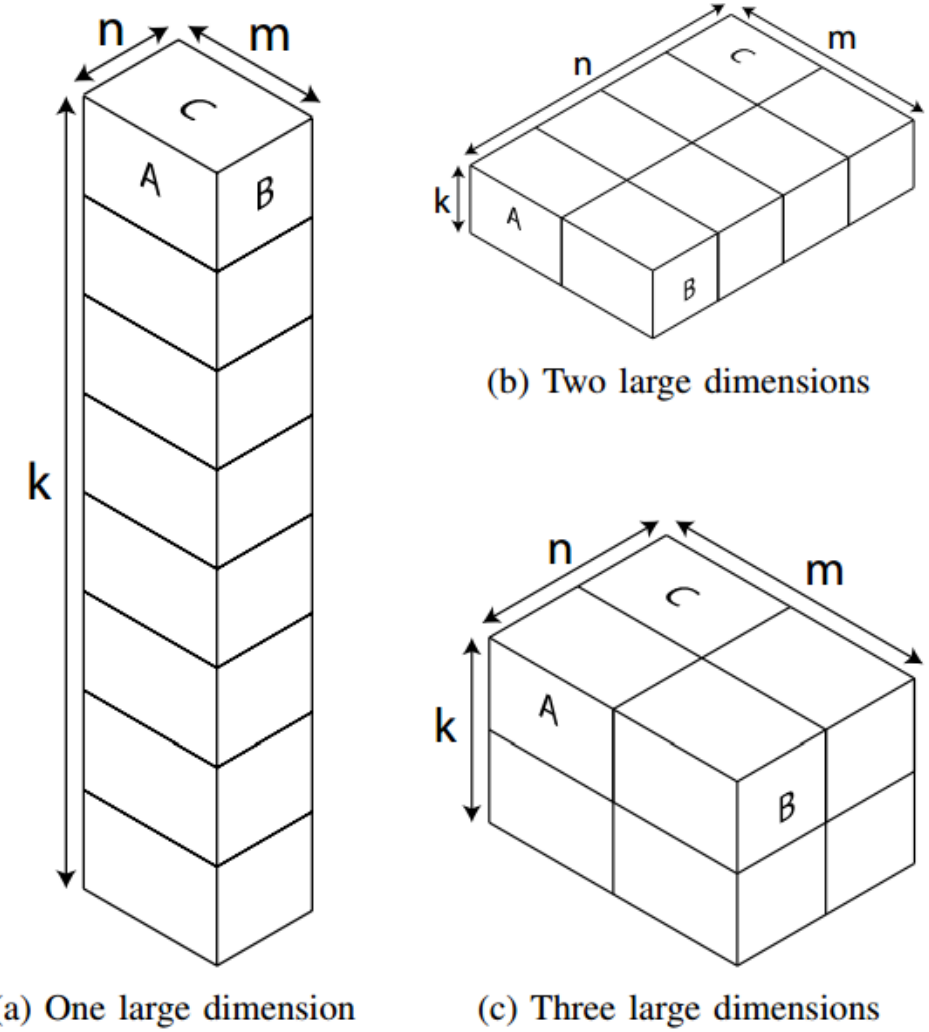
[Jia-Wei and Kung '81],[Irony et al '04], ...

- horizontal communication bound

$$W_{\mathrm{MM}} = \Theta\left(\min_{\substack{p_1,p_2,p_3 \geq 1 \\ p_1 p_2 p_3 = p}} \left(\frac{mk}{p_1 p_2} + \frac{kn}{p_2 p_3} + \frac{mn}{p_1 p_3}\right) - \frac{mk + kn + mn}{p}\right)$$

[Berntsen '89], [Aggarwal et al '89], [Agarwal et al '95], [Demmel et al '13]

Blocking matrix multiplication



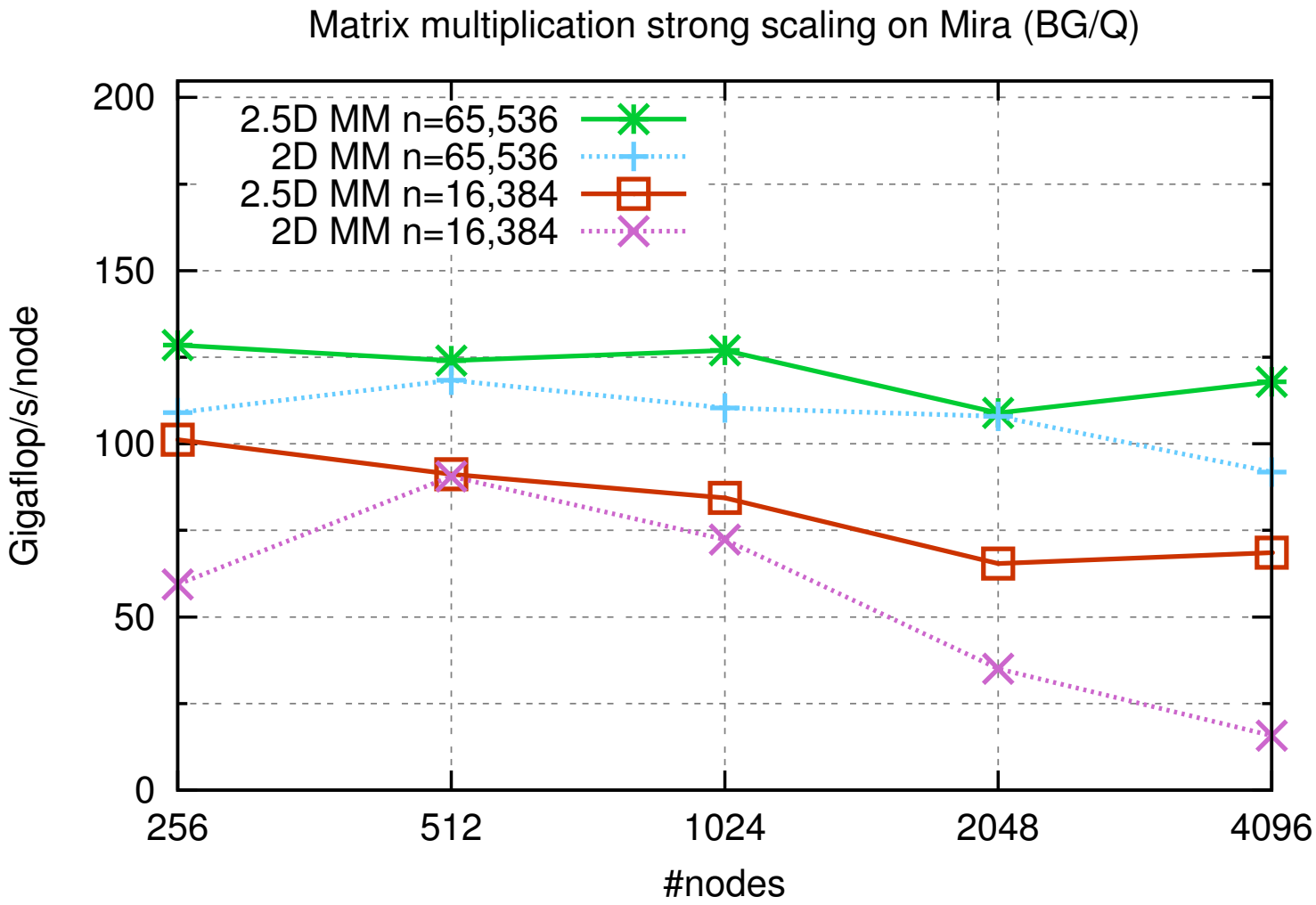
(a) One large dimension

(b) Two large dimensions

(c) Three large dimensions

[Demmel et al 13]

Benefit of three-dimensional matrix multiplication



Dense matrix factorizations

For Cholesky, LU, QR, SVD, symmetric eigensolve of $n \times n$ matrix, we generally have

- the same cost lower bounds as multiplication of $n \times n$ matrices
- tradeoffs between work and synchronization

$$F_{\mathrm{DMF}} \cdot S_{\mathrm{DMF}}^2 = \Omega(n^3)$$

- tradeoffs between horizontal communication and synchronization

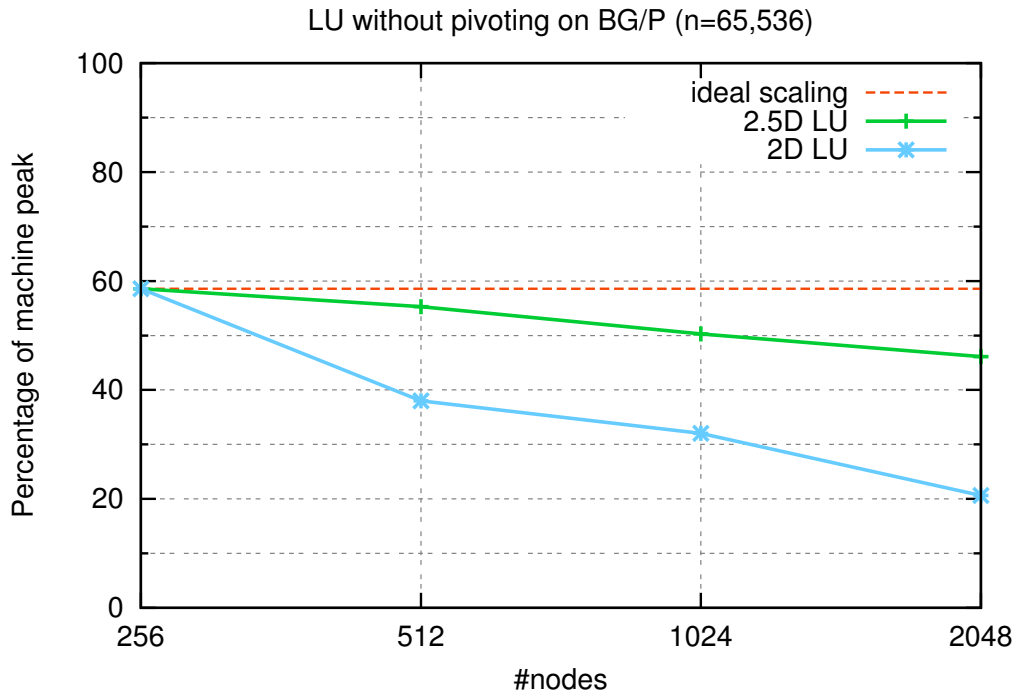
$$W_{\mathrm{DMF}} \cdot S_{\mathrm{DMF}} = \Theta(n^2)$$

[Tiskin '02], [Tiskin '07], [S. and Demmel '11], [S. et al '14]

2.5D algorithms

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

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



Communication-efficient (3D/2.5D) algorithms for LU and QR

LU factorization

- non-pivoted recursive communication-optimal LU [Aggarwal et al '89], [Tiskin '02]
- 2.5D LU with pairwise pivoting [Tiskin '07]
- 2.5D LU with tournament pivoting [S. and Demmel '11]

QR factorization

- 2.5D QR with Givens rotations [Tiskin '07]
- 2.5D QR in Householder form [S. '14]

General scheme for 2.5D algorithms

- Partition columns $A = [A_1, A_2]$ and perform both recursive calls with all processors
- Once matrix is sufficiently tall-and-skinny partition rows $A^T = [A_1^T, A_2^T]$ and recurse on both with half the processors

Communication-efficient algorithms for the symmetric eigensolve and SVD

2D algorithm with low vertical communication cost

- dense $\rightarrow$ banded $\rightarrow$ tridiagonal [Bischof et al '00]
- distributed-memory implementation in ELPA [Auckenthaller et al '11]

2.5D algorithms [S. '14]

- dense $\rightarrow$ banded $\rightarrow$ tridiagonal with optimal horizontal communication cost, requires extra work in computing

$$\mathbf{Z} = (\mathbf{A} + \mathbf{U}\mathbf{V}^T + \mathbf{V}\mathbf{U}^T)\mathbf{Y}$$

  with aggregated tall and skinny $\mathbf{U}$ and $\mathbf{V}$
- successive band reduction with optimal horizontal and vertical communication costs (modulo $\log(p)$ factors)

Krylov subspace methods

We consider the s-step Krylov subspace basis computation

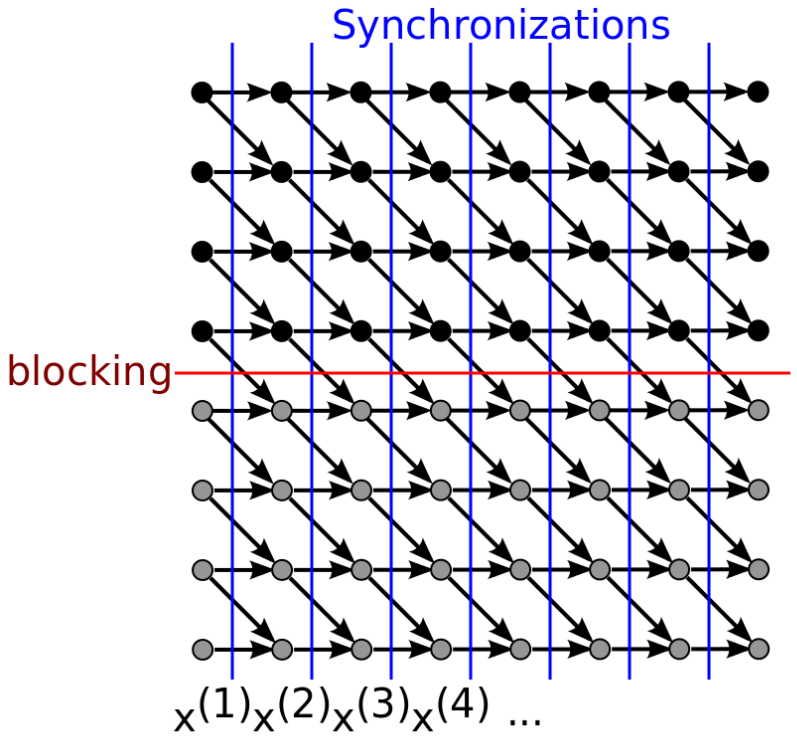
$$\mathbf{x}^{(l)} = \mathbf{A} \cdot \mathbf{x}^{(l-1)},$$

for $l \in \{1, \ldots, s\}$ where the graph of the sparse matrix $\mathbf{A}$ is a $(2m+1)^d$-point stencil.

The Krylov subspace dependency graph has $d+1$ dimensions, we say it has d mesh dimensions and 1 time dimension.

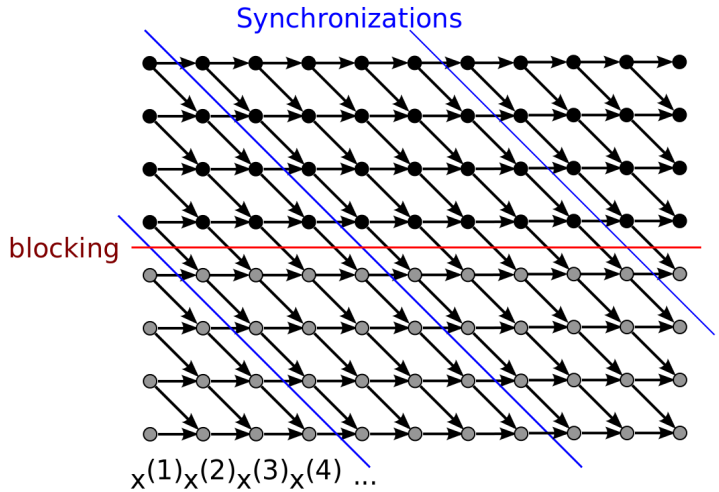
The standard algorithm (1D 2-pt stencil diagram)

Block the d mesh dimensions and perform one matrix vector multiplication at a time, synchronizing each time



The matrix-powers kernel

Avoid synchronization by blocking across matrix-vector multiplies (in the time dimension)

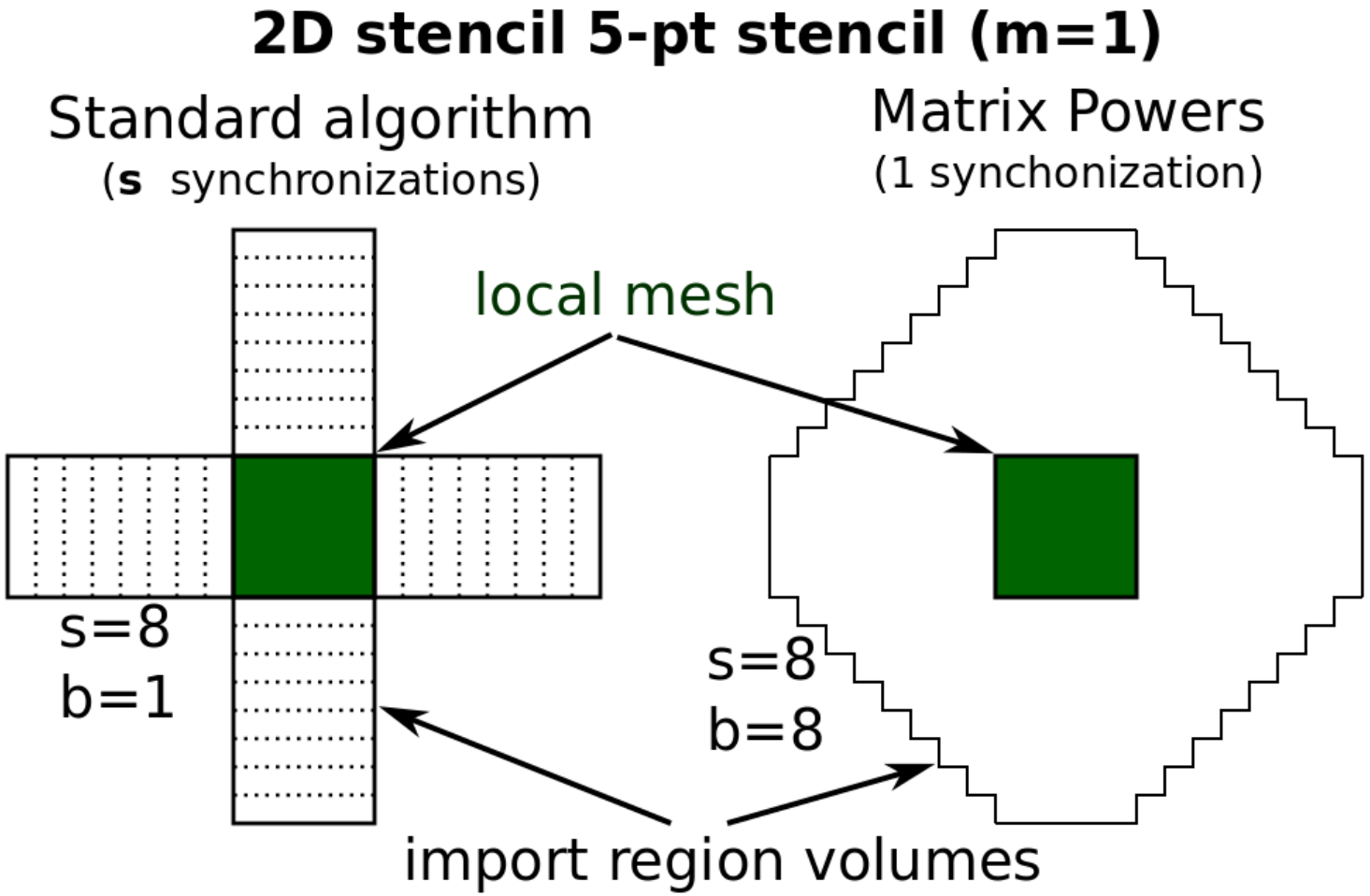


For a s-steps of a $(2m+1)^d$-point stencil with block-size b this algorithm has costs

$$W_{\mathrm{PA1}} = O\left((s/b)\left[(n/p^{1/d} + bm)^d - n^d/p\right]\right)$$

$$S_{\mathrm{PA1}} = O(s/b) \qquad Q_{\mathrm{PA1}} = O\left(\frac{sn^d}{\min(b, H^{1/d})p}\right)$$

Illustration of import region of the matrix-powers kernel



Communication lower bounds for Krylov subspace methods

Dependency interval analysis can be used to show the following result

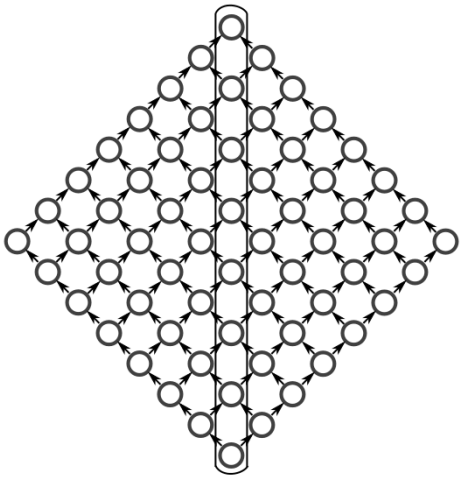
Theorem

Any parallel execution of an s-step Krylov subspace basis computation for a $(2m+1)^d$-point stencil, requires the following computational, bandwidth, and latency costs

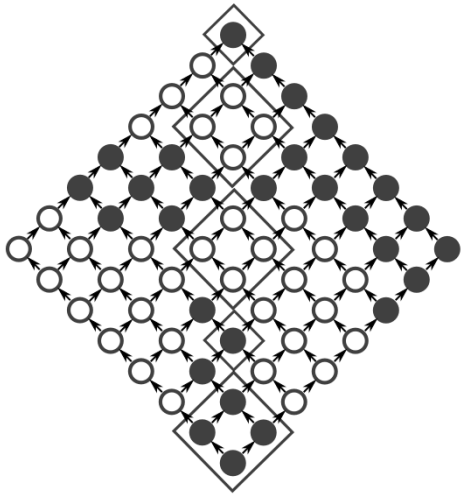
$$F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d} = \Omega\left(m^{2d} s^{d+1}\right), \quad W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^{d} s^{d}\right).$$

Derivation of communication lower bounds

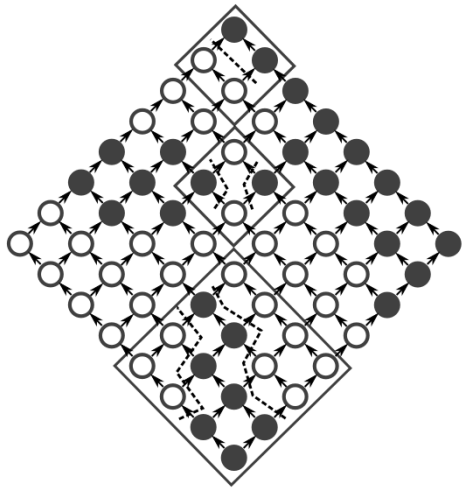
Done via dependency interval expansion combined with volumetric inequalities ([S. et al '14])



Dependency chain P



Monochrome dependency intervals

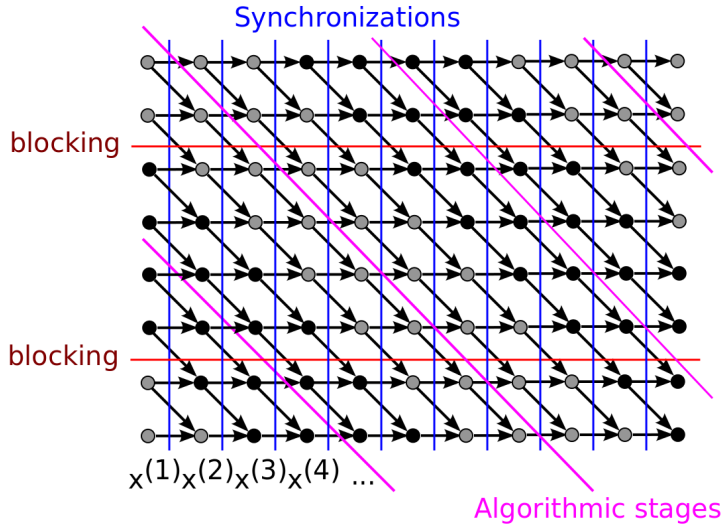


Multicolored dependency intervals

Idea of using monochrome dependency intervals chains to derive work–synchronization tradeoff [Papadimitriou and Ullman '87]

The matrix-powers kernel

Use block-cyclic layout, subdividing the global problem into stages, each executed in parallel



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

Electronic structure theory

Electronic structure calculations model the energies of chemical systems, taking into account of multi-electron interactions.
Density Functional Theory is the most common method

- cost is $O(n^3)$ for n electrons
- models system as a density functional, corrects for correlation
- good for metals and regular systems
- bad at molecules due to correlation effects on boundary

Coupled Cluster models electronic correlation explicitly

- cost is $O(n^{4+d})$, where $d \in \{2, 4, 6\}$
- the most accurate method used in practice

Coupled Cluster definition

Coupled Cluster (CC) is a method for computing an approximate solution to the time-independent Schrödinger equation of the form

$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle,$$

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

$$|\Psi\rangle = e^{\hat{\mathbf{T}}}|\Phi_0\rangle$$

where $\hat{\mathbf{T}}$ is as a sum of $\hat{\mathbf{T}}_n$ (the n'th excitation operators)

$$\begin{aligned}
\hat{\mathbf{T}}_{\mathrm{CCSD}} &= \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 \\
\hat{\mathbf{T}}_{\mathrm{CCSDT}} &= \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3 \\
\hat{\mathbf{T}}_{\mathrm{CCSDTQ}} &= \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3 + \hat{\mathbf{T}}_4
\end{aligned}$$

Coupled Cluster with Double excitations (CCD) equations

$e^{\hat{\mathbf{T}}_2}|\Phi_0\rangle$ turns into:

$$R_{ij}^{ab} = V_{ij}^{ab} + P(ia,jb)\Big[T_{ij}^{ae} I_e^b - T_{im}^{ab} I_j^m + \frac{1}{2} V_{ef}^{ab} T_{ij}^{ef} + \frac{1}{2} T_{mn}^{ab} I_{ij}^{mn} - T_{mj}^{ae} I_{ie}^{mb} - I_{ie}^{ma} T_{mj}^{eb} + (2T_{mi}^{ea} - T_{im}^{ea}) I_{ej}^{mb}\Big]$$

$$\begin{aligned}
I_b^a &= (-2V_{eb}^{mn} + V_{be}^{mn}) T_{mn}^{ea} \\
I_j^i &= (2V_{ef}^{mi} - V_{ef}^{im}) T_{mj}^{ef} \\
I_{kl}^{ij} &= V_{kl}^{ij} + V_{ef}^{ij} T_{kl}^{ef} \\
I_{jb}^{ia} &= V_{jb}^{ia} - \frac{1}{2} V_{eb}^{im} T_{jm}^{ea} \\
I_{bj}^{ia} &= V_{bj}^{ia} + V_{be}^{im} (T_{mj}^{ea} - \frac{1}{2} T_{mj}^{ae}) - \frac{1}{2} V_{be}^{mi} T_{mj}^{ae}
\end{aligned}$$

Exploiting symmetry by unfolding

Let $\mathbf{A}$ and $\mathbf{B}$ be two $n \times n$ antisymmetric matrices and consider the contraction,

$$c = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \cdot B_{ij} = 2 \sum_{i=1}^{n} \sum_{j=1}^{i-1} A_{ij} \cdot B_{ij}$$

This contraction may be unfolded into an inner product of vectors,

$$c = \langle \mathrm{vec}(\mathbf{A}), \mathrm{vec}(\mathbf{B}) \rangle = \langle \mathrm{vech}(\mathbf{A}), \mathrm{vech}(\mathbf{B}) \rangle$$

where vech (half-vectorization) takes only the unique entries.
This technique is 8X faster for the following CCSD contraction,

$$Z_{ij}^{ab} = \sum_{e,f} V_{ef}^{ab} \cdot T_{ij}^{ef} \quad \rightarrow \quad Z_{i<j}^{a<b} = \sum_{e<f} V_{e<f}^{a<b} \cdot T_{i<j}^{e<f}$$

as the tensors are antisymmetric in (a, b), (i, j), and (e, f).

Symmetry that does not conform to unfoldings

Consider the multiplication of an antisymmetric matrix **A** with a vector **b**,

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

while $A_{ij} = -A_{ji}$, the quantities $A_{ij}b_j$ and $A_{ji}b_i$ are arbitrarily different. Now consider another contraction from the CCSD method,

$$Z_{i\bar{c}}^{a\bar{k}} = \sum_{b,j} T_{ij}^{ab} \cdot V_{b\bar{c}}^{j\bar{k}}$$

where **T** is partially antisymmetric,

$$T_{ij}^{ab} = -T_{ij}^{ba} = -T_{ji}^{ab} = T_{ji}^{ba}$$

it is not possible to unfold these tensors and obtain a reduced-size matrix multiplication.

Symmetric-matrix–vector multiplication

- Consider symmetric $n \times n$ matrix $\mathbf{A}$ and vectors $\mathbf{b}, \mathbf{c}$
- $\mathbf{c} = \mathbf{A} \cdot \mathbf{b}$ is usually done by computing a *nonsymmetric* intermediate matrix $\mathbf{W}$,

$$W_{ij} = A_{ij} \cdot b_j \qquad\qquad c_i = \sum_{j=1}^{n} W_{ij}$$

  which requires n^2 multiplications and n^2 additions.
- The *symmetry preserving algorithm* employs a *symmetric* intermediate matrix $\mathbf{Z}$,

$$Z_{ij} = A_{ij} \cdot (b_i + b_j) \qquad\qquad c_i = \sum_{j=1}^{n} Z_{ij} - \left(\sum_{j=1}^{n} A_{ij} \right) \cdot b_i$$

  which requires $\frac{n^2}{2}$ multiplications and $\frac{5n^2}{2}$ additions.

Symmetrized rank-two outer product

- Consider vectors $\mathbf{a}, \mathbf{b}$ of dimension n
- Symmetric matrix $\mathbf{C} = \mathbf{a} \cdot \mathbf{b}^{\mathsf{T}} + \mathbf{b} \cdot \mathbf{a}^{\mathsf{T}}$ is usually done by computing a *nonsymmetric* intermediate matrix $\mathbf{W}$,

$$W_{ij} = a_i \cdot b_j \qquad\qquad C_{ij} = W_{ij} + W_{ji}$$

  which requires n^2 multiplications and $n^2/2$ additions.
- The *symmetry preserving algorithm* employs a *symmetric* intermediate matrix $\mathbf{Z}$,

$$Z_{ij} = (a_i + a_j) \cdot (b_i + b_j) \qquad C_{ij} = Z_{ij} - a_i \cdot b_i - a_j \cdot b_j$$

  which requires $\frac{n^2}{2}$ multiplications and $2n^2$ additions.

Symmetrized matrix multiplication

- Consider symmetric $n \times n$ matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$
- $\mathbf{C} = \mathbf{A} \cdot \mathbf{B} + \mathbf{B} \cdot \mathbf{A}$ is usually computed via a nonsymmetric intermediate order 3 tensor $\mathbf{W}$,

$$W_{ijk} = A_{ik} \cdot B_{kj} \qquad \bar{W}_{ij} = \sum_k W_{ijk} \qquad C_{ij} = W_{ij} + W_{ji}.$$

  which requires n^3 multiplications and n^3 additions.
- The *symmetry preserving algorithm* employs a *symmetric* intermediate tensor $\mathbf{Z}$ using $n^3/6$ multiplications and $7n^3/6$ additions,

$$Z_{ijk} = (A_{ij} + A_{ik} + A_{jk}) \cdot (B_{ij} + B_{ik} + B_{jk}) \qquad v_i = \sum_{k=1}^{n} A_{ik} \cdot B_{ik}$$

$$C_{ij} = \sum_{k=1}^{n} Z_{ijk} - n \cdot A_{ij} \cdot B_{ij} - v_i - v_j - \left(\sum_{k=1}^{n} A_{ik} \right) \cdot B_{ij} - A_{ij} \cdot \left(\sum_{k=1}^{n} B_{ik} \right)$$

Symmetry preserving algorithm

Consider contraction of symmetric tensors $\mathbf{A}$ of order $s+v$ and $\mathbf{B}$ of order $v+t$ that is symmetrized to produce a symmetric tensor $\mathbf{C}$ of order $s+t$

- Let $\omega = s+t+v$
- the symmetry preserving algorithm computes the order ω symmetric tensor $\hat{\mathbf{Z}}$, $\forall \vec{\mathbf{i}} = (i_1, \ldots, i_\omega)$, $1 \le i_1 \le \cdots \le i_\omega \le n$,

$$\begin{aligned} \vec{\mathbf{j}} \in \chi^{s+v}(\vec{\mathbf{i}}), \quad & \hat{A}_{\vec{\mathbf{i}}} \leftarrow A_{\vec{\mathbf{j}}} \\ \vec{\mathbf{l}} \in \chi^{v+t}(\vec{\mathbf{i}}), \quad & \hat{B}_{\vec{\mathbf{i}}} \leftarrow B_{\vec{\mathbf{l}}} \\ & \hat{Z}_{\vec{\mathbf{i}}} = \hat{A}_{\vec{\mathbf{i}}} \cdot \hat{B}_{\vec{\mathbf{i}}} \\ \vec{\mathbf{h}} \in \chi^{s+t}(\vec{\mathbf{i}}), \quad & Z_{\vec{\mathbf{h}}} \leftarrow \hat{Z}_{\vec{\mathbf{i}}} \end{aligned}$$

  where $\chi^k(\vec{\mathbf{i}})$ is the set of all $\binom{\omega}{k}$ combinations of k elements in $\vec{\mathbf{i}}$
- $\mathbf{C} = \mathbf{Z} - \ldots$ can then be computed with $O(n^{\omega-1})$ multiplications

Symmetry preserving algorithm costs

- Let $\Upsilon^{(s,t,v)}$ be the nonsymmetric contraction algorithm
- Let $\Psi^{(s,t,v)}$ be the direct evaluation algorithm
- Let $\Phi^{(s,t,v)}$ be the symmetry preserving algorithm

ω	s	t	v	F_Υ	F_Ψ	F_Φ	application cases
s+t+v	s	t	v	n^ω	$\binom{n}{s}\binom{n}{t}\binom{n}{v}$	$\binom{n}{\omega}$	generally
2	0	0	2	n^2	$n^2/2$	$n^2/2$	Frobenius norm of sym. mat.
2	1	0	1	n^2	n^2	$n^2/2$	symv, hemv, (symm, hemm)
2	1	1	0	n^2	n^2	$n^2/2$	syr2, her2, (syr2k, her2k)
3	1	1	1	n^3	n^3	$n^3/6$	matrix (anti)commutator

where F_X is the number of multiplications computed by algorithm X

Antisymmetry and matrix powers

The symmetry preserving algorithm can compute

- symmetrized products of two symmetric or two antisymmetric tensors
- antisymmetrized products of a symmetric and an antisymmetric tensor
- Hermitian tensor contractions
- $\mathbf{A}^2$ for symmetric or antisymmetric $\mathbf{A}$ with $n^3/6$ multiplications
- $\mathbf{A}^2$ for nonsymmetric $\mathbf{A}$ (or $\mathbf{A} \cdot \mathbf{B} + \mathbf{B} \cdot \mathbf{A}$ for nonsymmetric $\mathbf{A}$, $\mathbf{B}$) with $2n^3/3$ multiplications
- that CCSD contraction,

$$Z_{i\bar{c}}^{a\bar{k}} = \sum_{b,j} T_{ij}^{ab} \cdot V_{b\bar{c}}^{j\bar{k}}$$

in n^6 operations (2X fewer) via $\Phi^{(1,0,1)} \otimes \Upsilon^{(1,2,1)}$

Communication cost of direct evaluation of symmetric contractions

The communication cost of $\Psi^{(s,t,v)}$ is proportional to the communication cost of a matrix multiplication of dimensions $n^s \times n^t \times n^v$,

- when exactly one of s, t, v is zero

$$Q_\Psi = \Theta(n^\omega/p) \qquad W_\Psi = \Omega\left((n^\omega/p)^{1/2}\right)$$

- when $s, t, v > 0$

$$Q_\Psi = \Theta(n^\omega/(pH^{1/2})) \qquad W_\Psi = \Omega\left((n^\omega/p)^{2/3}\right)$$

When exactly one of s, t, v is zero, any load balanced schedule of $\Psi^{(s,t,v)}$ on a parallel machine with p processors has horizontal communication cost,

$$W_\Psi = \Theta\left((n^\omega/p)^{\max(s,t,v)/\omega}\right) = \Omega\left((n^\omega/p)^{1/2}\right)$$

Communication lower bounds for the symmetry preserving algorithm

The symmetry-preserving algorithm $\Phi^{(s,t,v)}$ has a fundamentally different dependency structure and requires more communication per multiplication

Let $\kappa := \max(s+v, v+t, s+t)$.

The vertical communication cost is

$$Q_\Phi = \Theta\left(\frac{n^\omega H}{H^{\omega/\kappa}} + n^\kappa\right) = \Omega\left(\frac{n^\omega}{H^{1/2}} + n^\kappa\right)$$

The horizontal communication cost is

$$W_\Phi = \begin{cases} \Theta\left((n^\omega/p)^{\kappa/\omega}\right) & = \Omega\left((n^\omega/p)^{2/3}\right) \quad : s,t,v > 0 \\ \Theta\left((n^\omega/p)^{\max(s,t,v)/\omega}\right) & = \Omega\left((n^\omega/p)^{1/2}\right) \quad : \kappa = \omega \end{cases}$$

Horizontal communication cost overview

- $\Upsilon^{(s,t,v)}$ is the nonsymmetric contraction algorithm
- $\Psi^{(s,t,v)}$ is the direct evaluation algorithm
- $\Phi^{(s,t,v)}$ is the symmetry preserving algorithm

s	t	v	F_Υ	F_Ψ	F_Φ	$Q_{\Upsilon,\Psi}$	Q_Φ	W_Υ	W_Ψ	W_Φ
1	0	1	n^2	n^2	$n^2/2$	$\frac{n^2}{p}$	$\frac{n^2}{p}$	$\left(\frac{n^2}{p}\right)^{1/2}$	$\left(\frac{n^2}{p}\right)^{1/2}$	$\left(\frac{n^2}{p}\right)^{1/2}$
1	1	1	n^3	n^3	$n^3/6$	$\frac{n^3}{pH^{1/2}}$	$\frac{n^3}{pH^{1/2}}$	$\left(\frac{n^3}{p}\right)^{2/3}$	$\left(\frac{n^3}{p}\right)^{2/3}$	$\left(\frac{n^3}{p}\right)^{2/3}$
2	0	1	n^3	$n^3/2$	$n^3/6$	$\frac{n^3}{p}$	$\frac{n^3}{p}$	$\left(\frac{n^3}{p}\right)^{1/2}$	$\left(\frac{n^3}{p}\right)^{2/3}$	$\left(\frac{n^3}{p}\right)^{2/3}$
2	1	1	n^4	$n^4/2$	$n^4/12$	$\frac{n^4}{pH^{1/2}}$	$\frac{n^4}{pH^{1/3}}$	$\left(\frac{n^4}{p}\right)^{2/3}$	$\left(\frac{n^4}{p}\right)^{2/3}$	$\left(\frac{n^4}{p}\right)^{3/4}$

All communication costs above are asymptotic

Parallelization of the standard contraction algorithm

The standard symmetric contraction algorithm $\Psi^{(s,t,v)}$ is dominated by a matrix multiplication of an $\binom{n}{s}$-by-$\binom{n}{v}$ matrix with an $\binom{n}{v}$-by-$\binom{n}{t}$ matrix into an $\binom{n}{s}$-by-$\binom{n}{t}$ matrix

- the main parallelization challenge is to get the tensor data into the desired matrix layout

Cyclops Tensor Framework (CTF)

- contraction/summation/functions of tensors
- distributed symmetric-packed/sparse storage via cyclic layout
- two-level parallelization via MPI+OpenMP
- performance-model-driven algorithm selection
- sparse/dense/blocked redistributions
- uses 2.5D matrix multiplication for contractions
- provides concise interface for tensor objects and contractions

Tensor algebra interface (credit: Devin Matthews)

CTF can express a tensor contraction like

$$Z_{ij}^{ab} = \frac{1}{2} \cdot W_{ij}^{ab} + 2 \cdot P(a,b) \sum_k F_k^a \cdot T_{ij}^{kb}$$

where $P(a,b)$ implies antisymmetrization of index pair ab, as

```
Z["abij"]  = 0.5*W["abij"];
Z["abij"] += 2.0*F["ak"]*T["kbij"];
```

- **for** loops and summations implicit in syntax
- $P(a,b)$ is applied implicitly if **Z** is antisymmetric in ab
- **Z**, **F**, **T**, **W** should all be defined on the same world and all processes in the world must call the contraction bulk synchronously
- user-defined (mixed-type) scalar tensor functions can be applied instead of $+$ and $*$

CCSD

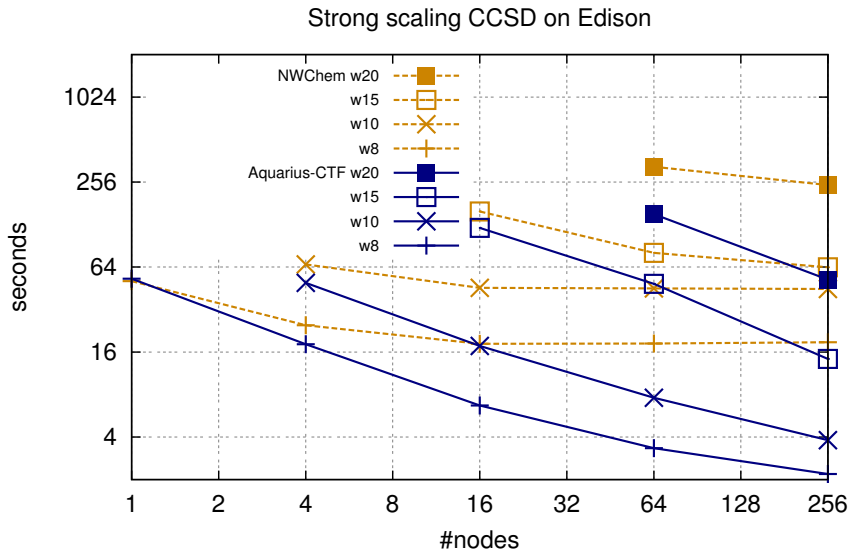
Extracted from Aquarius (Devin Matthews' code, https://github.com/devinamatthews/aquarius)

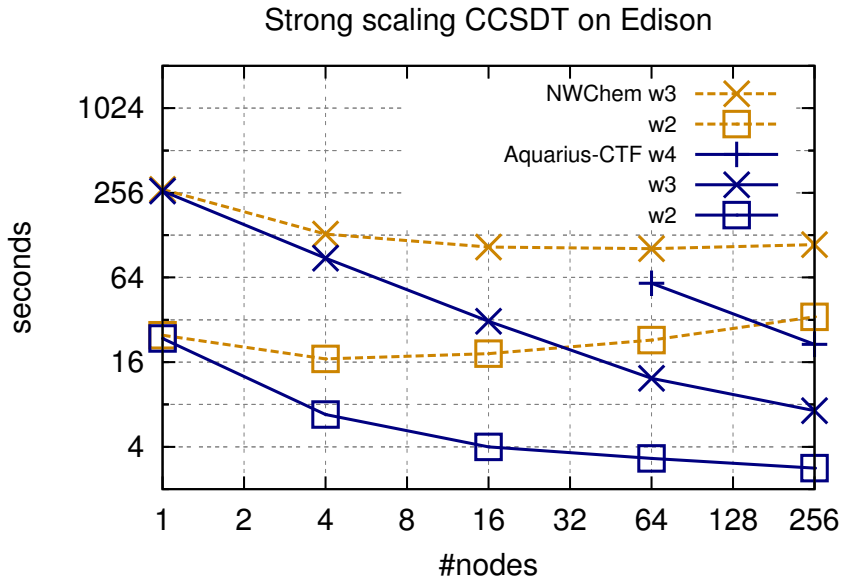
```
FMI["mi"]     += 0.5*WMNEF["mnef"]*T(2)["efin"];
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T(2)["efij"];
FAE["ae"]     -= 0.5*WMNEF["mnef"]*T(2)["afmn"];
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T(2)["afin"];

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"];
```

Comparison with NWChem

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

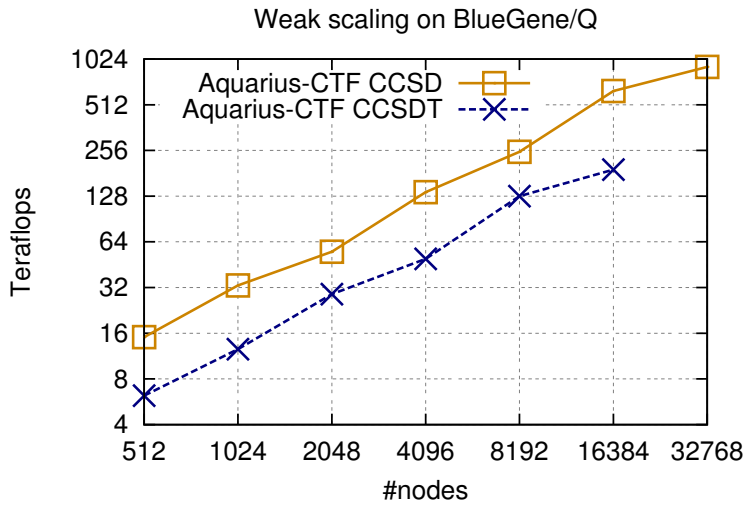
- provides CCSD and CCSDT
- uses Global Arrays a Partitioned Global Address Space (PGAS) backend for tensor contractions
- derives equations via Tensor Contraction Engine (TCE)

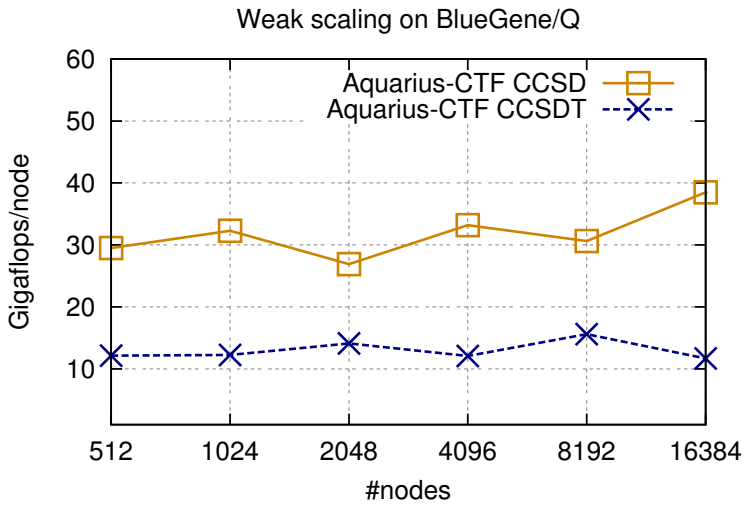


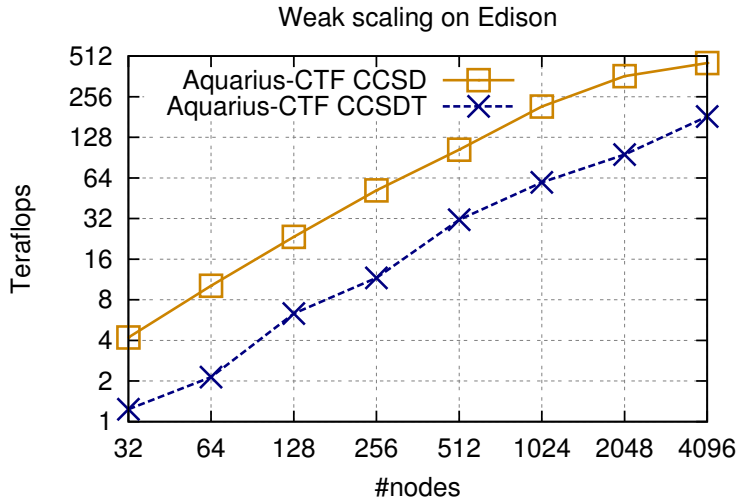


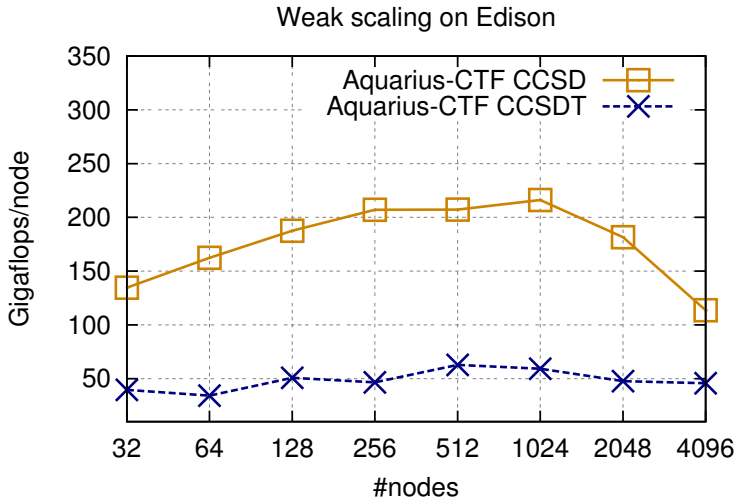
Coupled Cluster on IBM BlueGene/Q

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









Adaptation to Sparse–Dense Matrix Multiplication

Dense matrix-multiplication cost for $m \times k$ matrix A, $k \times n$ matrix B, and $m \times n$ matrix C on p processors is

$$W_{\text{MM}} = \Theta\left(\min_{p_1 p_2 p_3 = p} \left[\frac{mk}{p_1 p_2} + \frac{kn}{p_1 p_3} + \frac{mn}{p_2 p_3} \right] - \frac{mk + kn + mn}{p} \right).$$

Communicating only the $z \in [1, mk]$ nonzeros of A we obtain

$$W_{\text{SPMM}} = O\left(\min_{p_1 p_2 p_3 = p} \left[\frac{z}{p_1 p_2} + \frac{kn}{p_1 p_3} + \frac{mn}{p_2 p_3} \right] - \frac{z + kn + mn}{p} \right).$$

This sparse communication cost is not generally optimal, lower cost possible for certain sparsity structures.

Jacobi Iteration

Solve $Ax = b$ using

$$\forall i \in [1, n], \quad x_i^{l+1} = (1/A_{ii}) \cdot (b_i - \sum_{i=0, i \neq j}^{n} A_{ij} \cdot x_j^l),$$

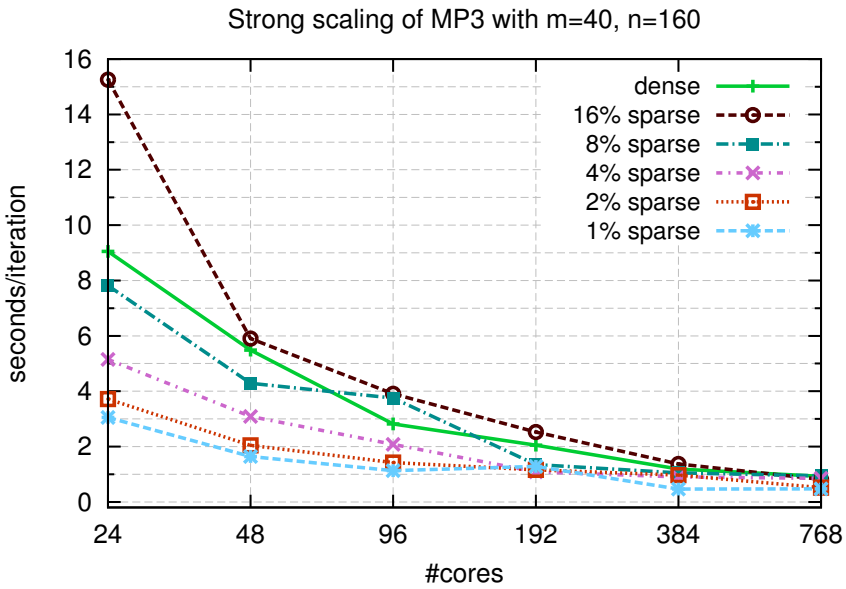
```
void Jacobi(Matrix<> & A, Vector<> & b, int n){
  Vector<> x(n), d(n), r(n);
  Matrix<> R(n,n,SP);
  d["i"] = A["ii"];
  Transform<> inv([](double & d){ d=1./d; }); inv(d["i"]);
  R["ij"] = A["ij"];
  R["ii"] = 0; //set the diagonal of R to zero
  do {
    x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
    r["i"] = b["i"]-A["ij"]*x["j"]; // compute residual
  } while (r.norm2() > 1.E-6) // check for convergence
}
```

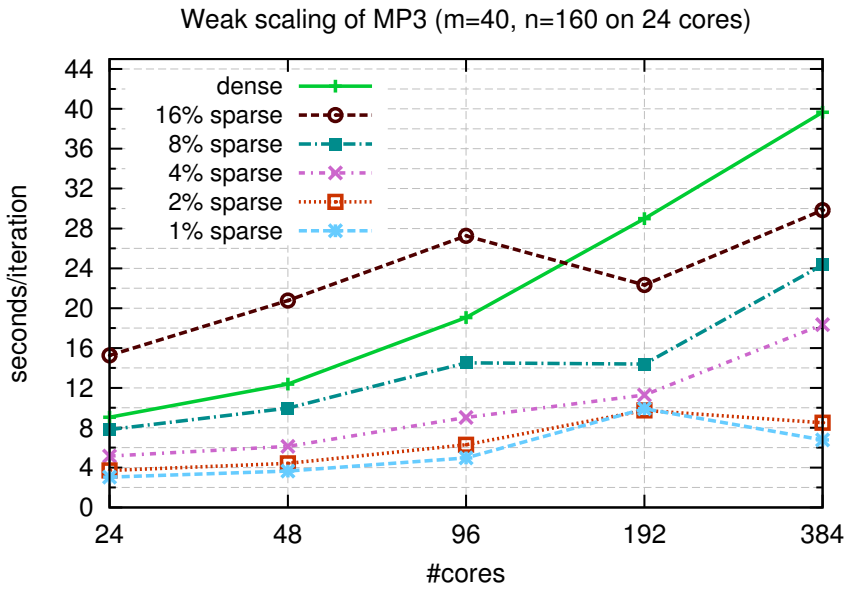
Third-Order Møller-Plesset Perturbation Theory (MP3)

Given integral tensors: **Ei**, **Ea**, **Vabij**, **Vijab**, **Vabcd**, **Vijkl**, **Vaibj**,

```
Tensor<> D(4,Vabij.lens,*Vabij.wrld);
D["abij"] += Ei["i"] + Ei["j"] - Ea["a"] - Ea["b"];
Transform<> div([](double & b){ b=1./b; });
div(D["abij"]);
Tensor<> T(4,Vabij.lens,*Vabij.wrld);
T["abij"] = Vabij["abij"]*D["abij"];

Tensor<> Z(4,Vabij.lens,*Vabij.wrld);
Z["abij"] = Vijab["ijab"];
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"];
T["abij"] += Z["abij"]*D["abij"];
double MP3_energy = T["abij"]*Vabij["abij"];
```

MP3 with Sparse Integral Tensors





Summary

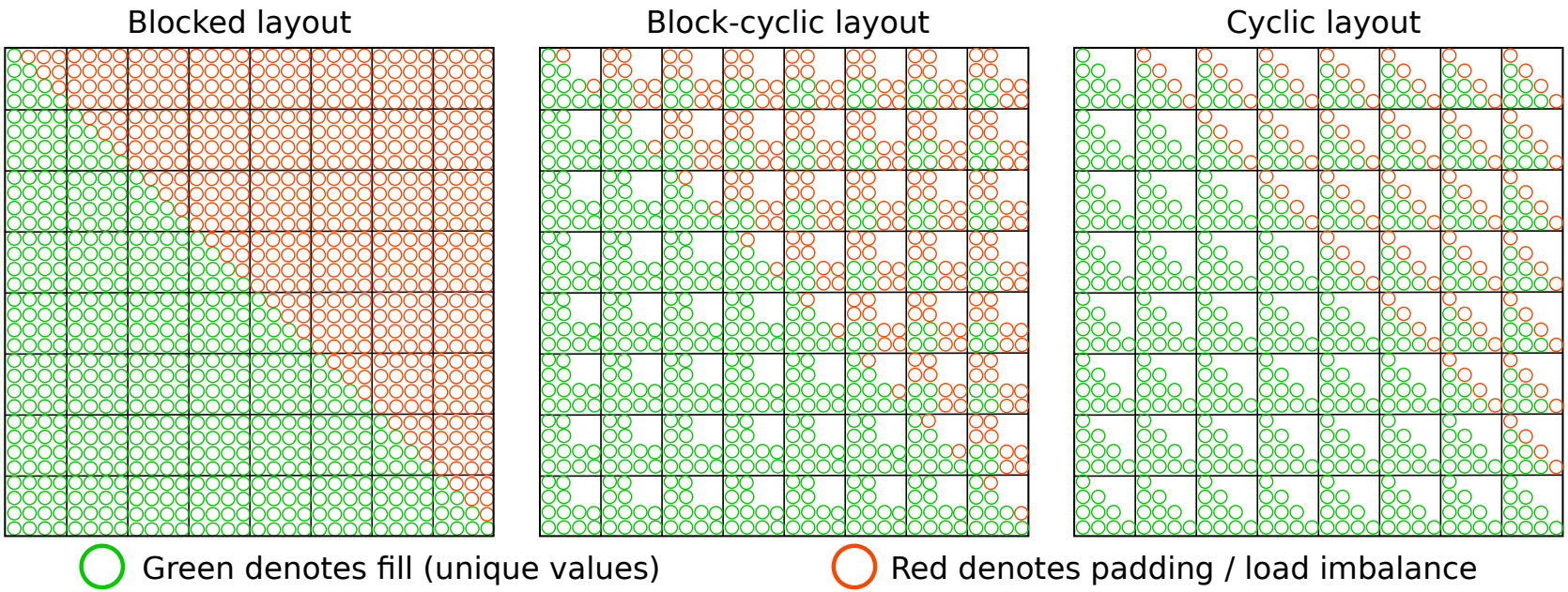
- Given an algebraic representation of an algorithm, its optimal (parallel) efficiency can be derived by lower+upper bounds on communication
- 2.5D algorithms lower communication cost for MM, LU, QR, SVD, but raise synchronization for LU, QR, and SVD
- Lowering synchronization cost of Krylov subspace computations requires an increase in horizontal communication cost
- When the cache size is not too small, a block-cyclic layout keeps horizontal and vertical communication costs low
- Coupled cluster requires contraction of tensors with symmetries
- The symmetry preserving algorithm decreases the cost of symmetric tensor contractions, but can require more communication
- Cyclops Tensor Framework (CTF) provides a domain specific language for parallel tensor computations

Future work

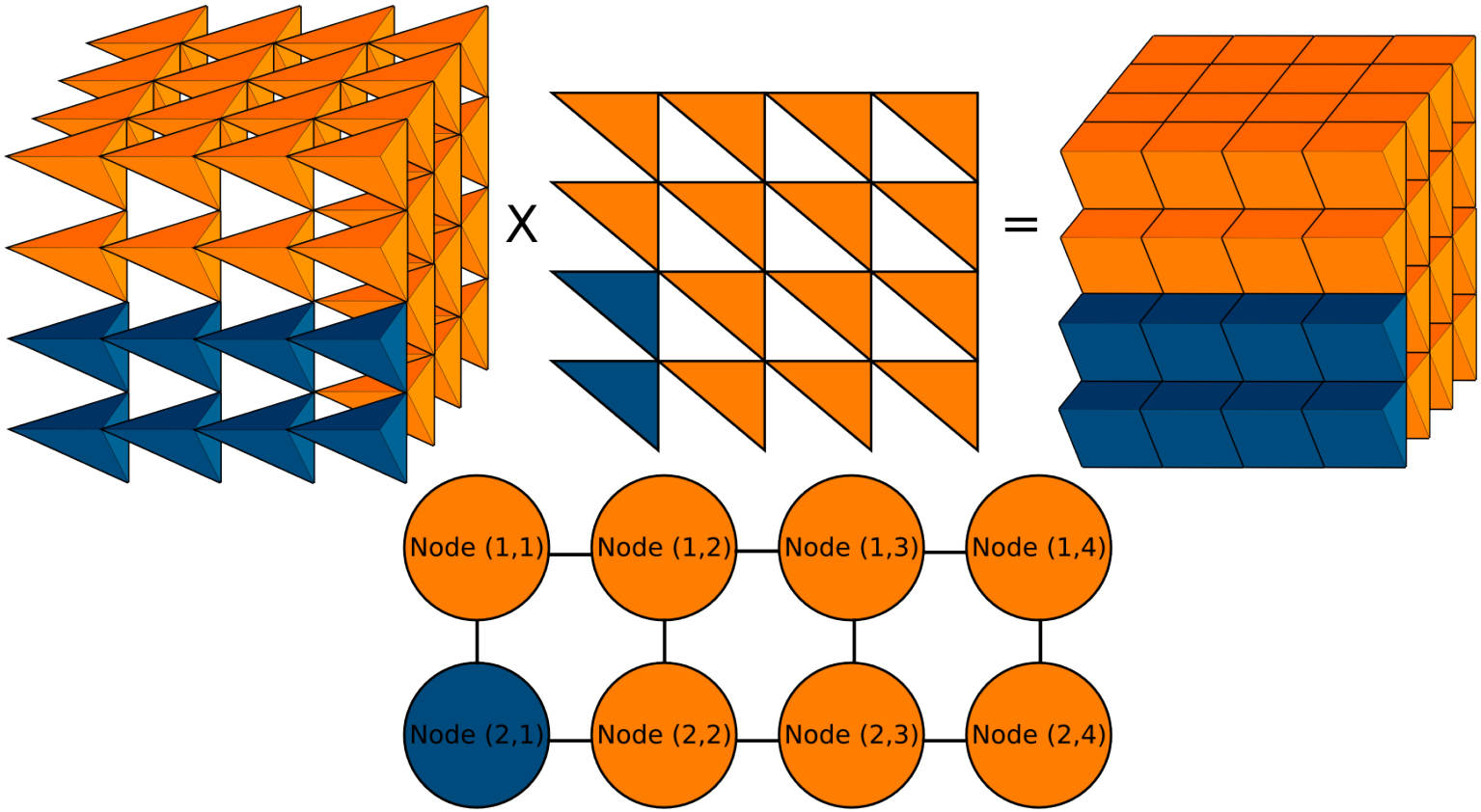
- investigation of the performance of 2.5D QR and 2.5D SVD
- high-performance implementation of the symmetry preserving algorithm
- communication lower bounds for sparse tensor contractions
- support for matrix and tensor factorizations within CTF

Backup slides

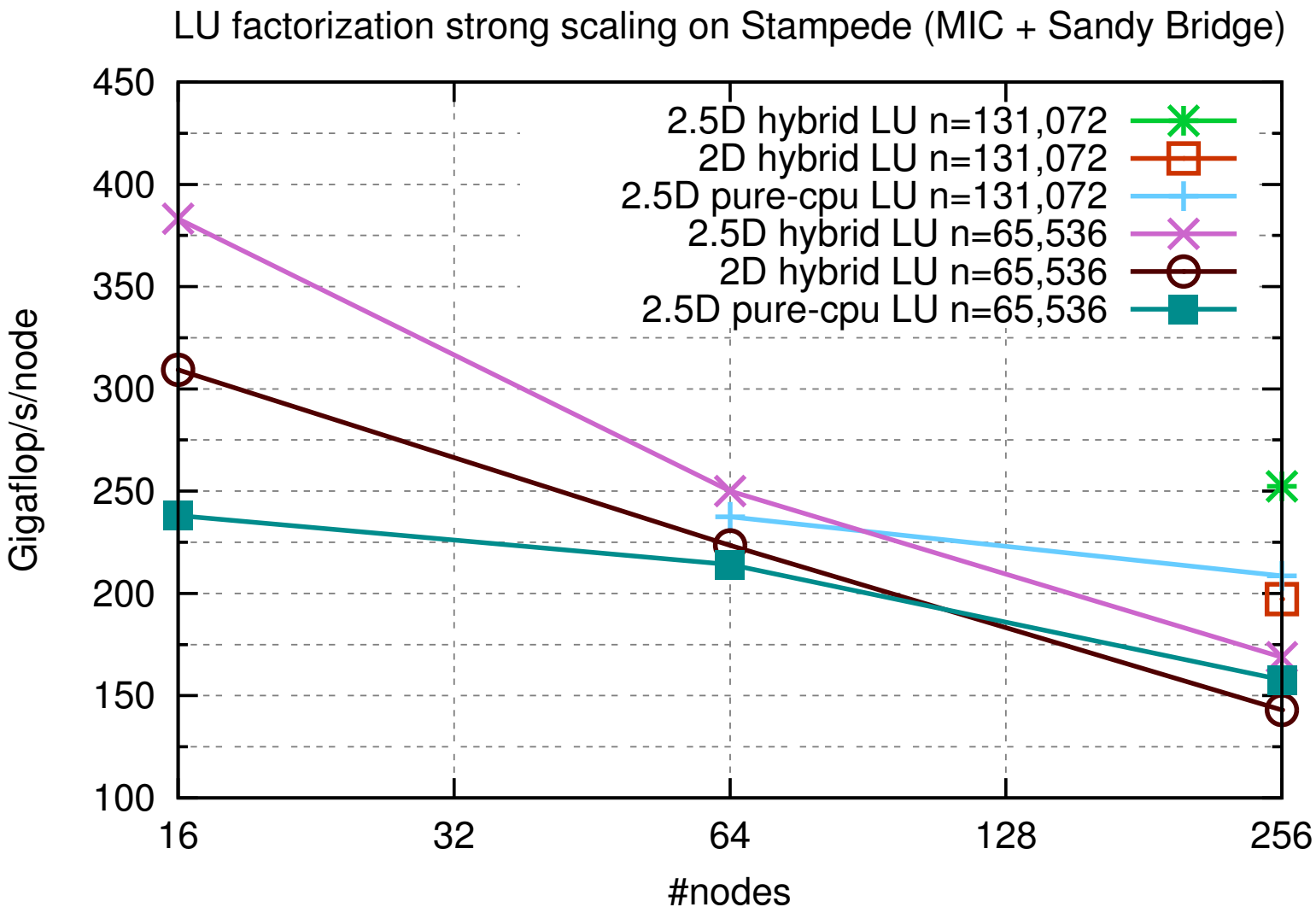
Blocked vs block-cyclic vs cyclic decompositions



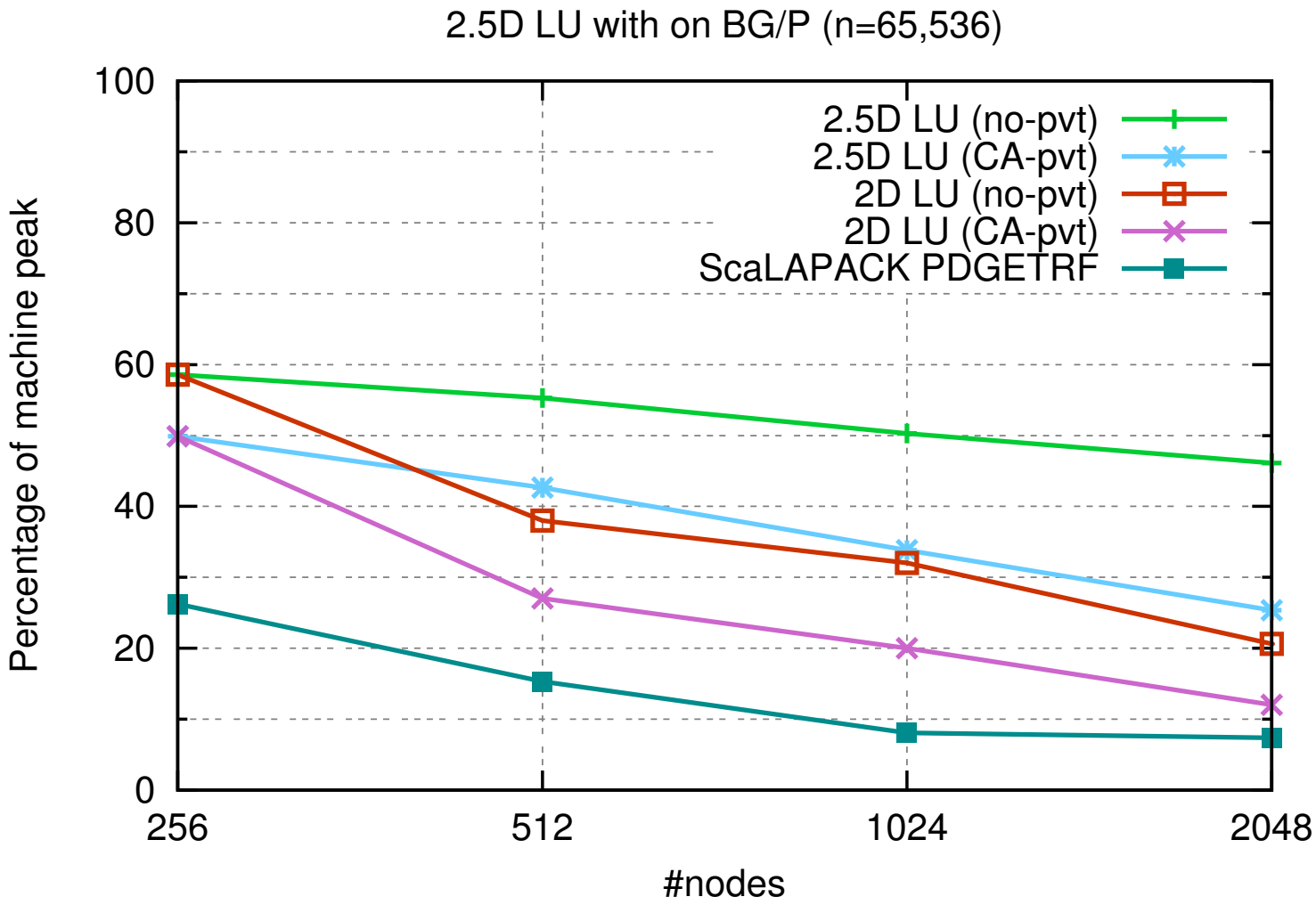
3D tensor mapping



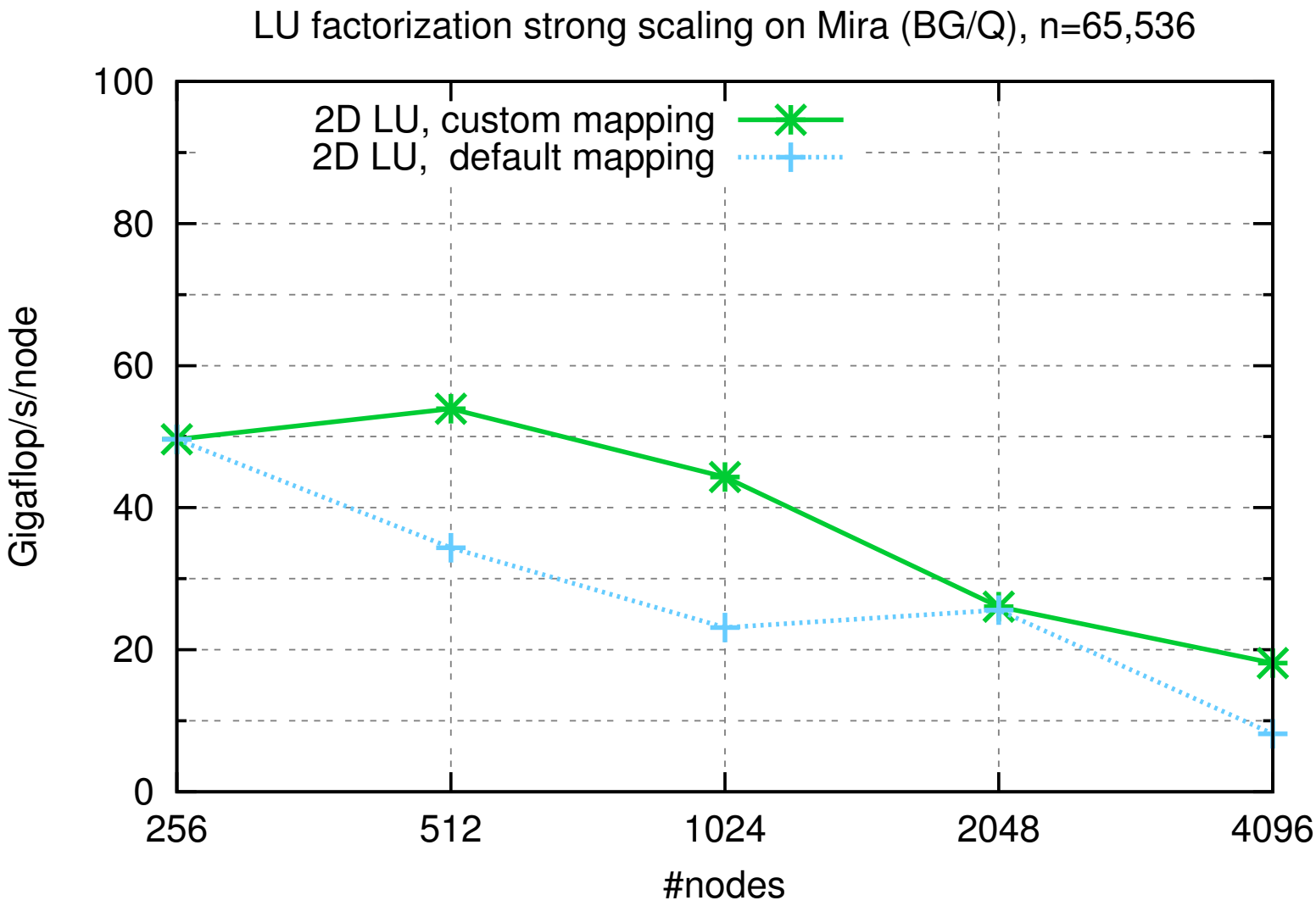
2.5D LU on MIC


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

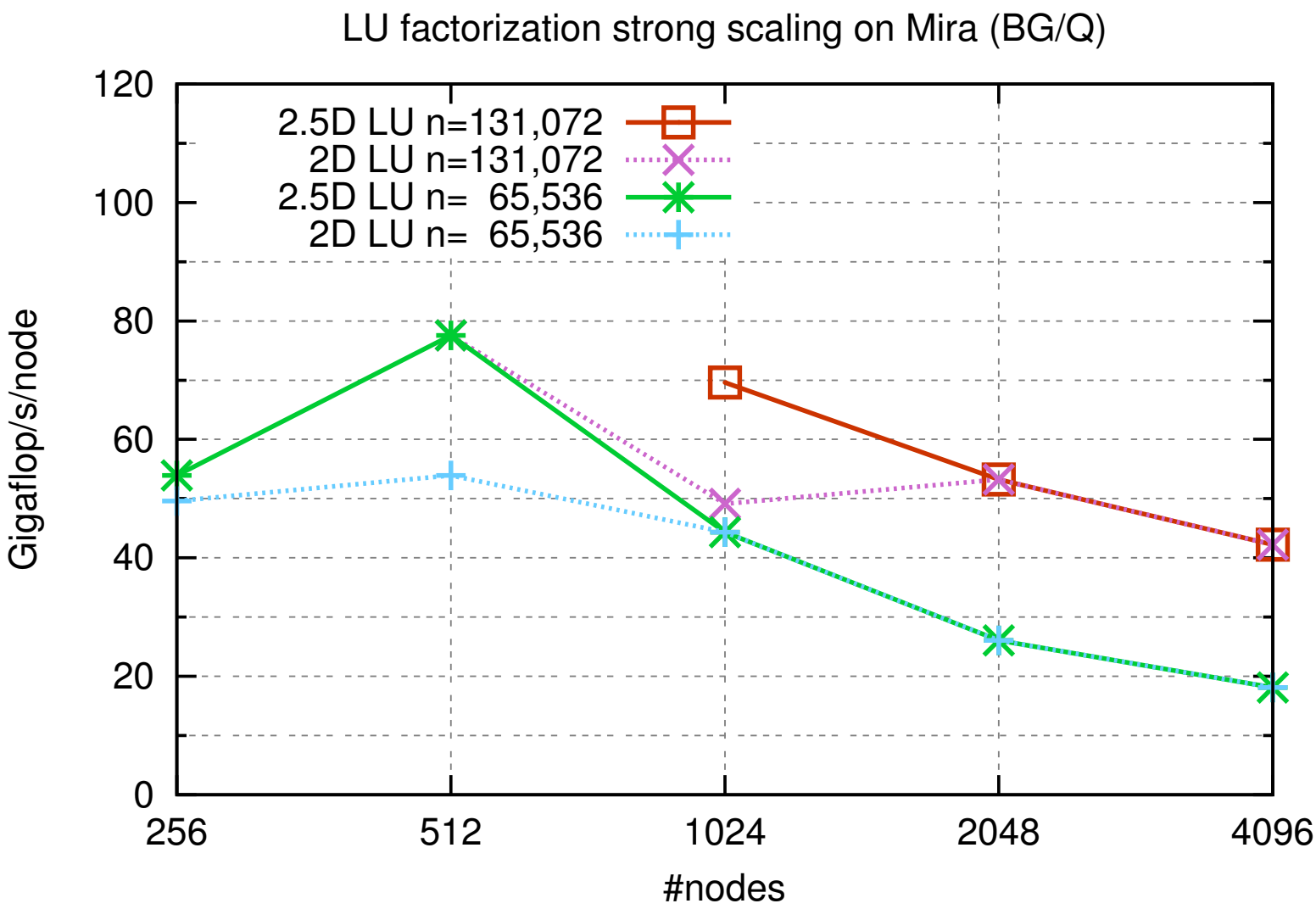
2.5D LU strong scaling



Topology-aware mapping on BG/Q

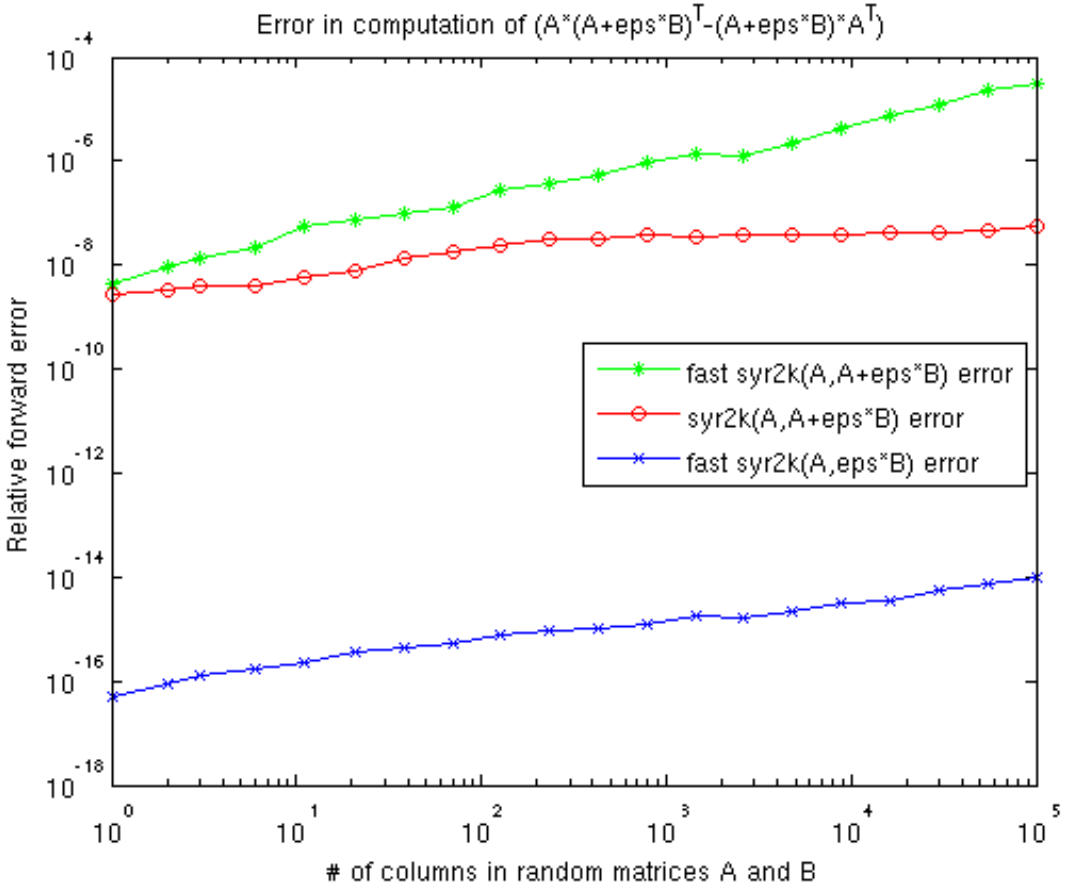


Benefit of replication on BG/Q



Disclaimer: numerical characteristics

The fast contraction algorithms have different numerical characteristics in floating-point precision



Symmetry preserving algorithm vs Strassen's algorithm