Provably efficient algorithms for numerical tensor algebra

Edgar Solomonik

Department of EECS, Computer Science Division, UC Berkeley

Dissertation Defense

Adviser: James Demmel

Aug 27, 2014

Outline

Motivation

- search for robust communication-efficient algorithms deployable under the umbrella of mathematical abstractions

1. Theoretical model of parallel computation
2. Communication lower bound techniques
3. 2.5D algorithms for dense linear algebra
4. Krylov subspace computations
5. Symmetric tensor computations
6. Future work

Model of the computer architecture

We quantify interprocessor communication and synchronization costs of a parallelization via a flat network model

- γ - cost for a single computation (flop)
- β - cost for a transfer of each byte between any pair of processors
- α - cost for a synchronization between any pair of processors
- ϱ - cost for a transfer of each byte between local memory and local cache

We assume $\varrho < \beta$ and report only memory-bandwidth costs auxiliary to interprocessor-bandwidth costs.

Model of the parallel schedule

We measure the cost of a parallelization along the longest sequence of dependent computations and data transfers (critical path)

- F - critical path payload for computation cost
- W - critical path payload for communication (bandwidth) cost
- S - critical path payload for synchronization cost
- Q - critical path payload for memory bandwidth cost

The total time T is then bounded by

$$\max(F \cdot \gamma, W \cdot \beta, S \cdot \alpha, Q \cdot \varrho) \leq T \leq F \cdot \gamma + W \cdot \beta + S \cdot \alpha + Q \cdot \varrho$$

The problem, the algorithm, and the parallelization

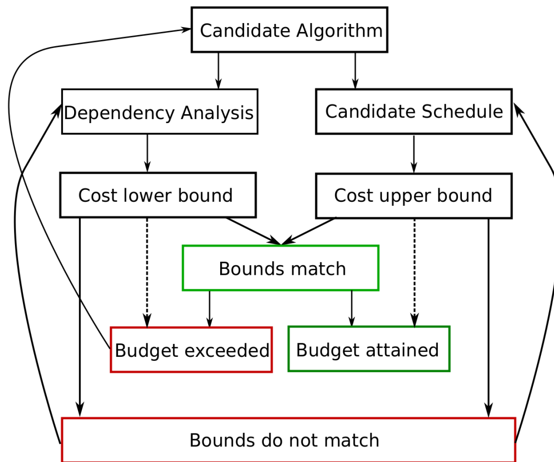


Figure : How do lower bounds facilitate the design of parallel algorithms?

Obtaining communication lower bounds for algorithms

We can represent an algorithm as a DAG $G = (V, E)$ where

- the vertices V are the input, intermediate, and output values
- the edges E encode the dependencies between pairs of values

Obtaining communication lower bounds for algorithms

We can represent an algorithm as a DAG $G = (V, E)$ where

- the vertices V are the input, intermediate, and output values
- the edges E encode the dependencies between pairs of values

What is the least amount of communication W needed to compute G with p processors?

- W is at least the size of the minimum vertex separator of G where each partition has at least $|V|/p$ vertices.

Obtaining communication lower bounds for algorithms

We can represent an algorithm as a DAG $G = (V, E)$ where

- the vertices V are the input, intermediate, and output values
- the edges E encode the dependencies between pairs of values

What is the least amount of communication W needed to compute G with p processors?

- W is at least the size of the minimum vertex separator of G where each partition has at least $|V|/p$ vertices.

How do we determine the minimum vertex separator of G?

- find hypergraph structure by merging disjoint sets of edges
- derive minimum hyperedge cut bounds for hypergraphs

Lattice hypergraphs

For any $m > r > 0$, we define a (m, r)-**lattice hypergraph** $H = (V, E)$ of breadth n,

- with $|V| = \binom{n}{m}$ vertices $v_{i_1 \ldots i_m} \in V$ with $i_1 < \cdots < i_m$
- with $|E| = \binom{n}{r}$ hyperedges $e_{j_1 \ldots j_r} \in E$ which connect all $v_{i_1 \ldots i_m}$ for which $\{j_1 \ldots j_r\} \subset \{i_1 \ldots i_m\}$.

Lattice hypergraphs

For any $m > r > 0$, we define a (m, r)-**lattice hypergraph** $H = (V, E)$ of breadth n,

- with $|V| = \binom{n}{m}$ vertices $v_{i_1 \ldots i_m} \in V$ with $i_1 < \cdots < i_m$
- with $|E| = \binom{n}{r}$ hyperedges $e_{j_1 \ldots j_r} \in E$ which connect all $v_{i_1 \ldots i_m}$ for which $\{j_1 \ldots j_r\} \subset \{i_1 \ldots i_m\}$.

Theorem

The minimum $\frac{1}{p}$-balanced hyperedge cut of a (m, r)-lattice hypergraph $H = (V, E)$ of breadth n is of size $\Omega(n^r / p^{r/m})$.

Lattice hypergraphs

For any $m > r > 0$, we define a (m, r)-**lattice hypergraph** $H = (V, E)$ of breadth n,

- with $|V| = \binom{n}{m}$ vertices $v_{i_1 \ldots i_m} \in V$ with $i_1 < \cdots < i_m$
- with $|E| = \binom{n}{r}$ hyperedges $e_{j_1 \ldots j_r} \in E$ which connect all $v_{i_1 \ldots i_m}$ for which $\{j_1 \ldots j_r\} \subset \{i_1 \ldots i_m\}$.

Theorem

The minimum $\frac{1}{p}$-balanced hyperedge cut of a (m, r)-lattice hypergraph $H = (V, E)$ of breadth n is of size $\Omega(n^r / p^{r/m})$.

Proof using generalized Loomis-Whitney inequality

Lemma

For any subset $\bar{V} \subset V$, let $\bar{E} \subset E$ be the hyperedges adjacent to $\bar{V}$

$$|\bar{E}| \geq |\bar{V}|^{r/m}$$

Path-expander graph

Definition (order-d-path-expander)

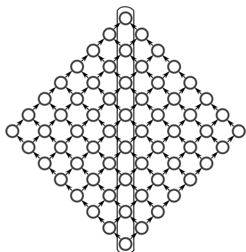
Graph $G = (V, E)$ is an **order-d-path-expander** if it has a path $(u_1 \dots u_n) \subset V$, and the union of all paths between u_i and u_j for all $j > i$ is a $(d, d-1)$-lattice hypergraph

Path-expander graph

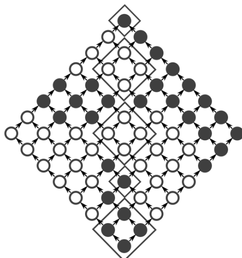
Definition (order-d-path-expander)

Graph $G = (V, E)$ is an **order-d-path-expander** if it has a path $(u_1 \dots u_n) \subset V$, and the union of all paths between u_i and u_j for all $j > i$ is a $(d, d-1)$-lattice hypergraph

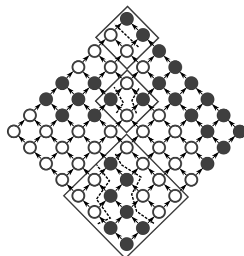
An example of a order-2-path-expander



Dependency path P



Computation chain



Communication chain

Scheduling tradeoffs of path-expander graphs

Theorem (Path-expander communication lower bound)

*Any schedule of an algorithm with an order-**d**-path-expander dependency graph about a path of length **n** incurs tradeoffs between computation (F), bandwidth (W), and latency (S) costs:*

$$F \cdot S^{d-1} = \Omega\left(n^d\right), \quad W \cdot S^{d-2} = \Omega\left(n^{d-1}\right).$$

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\text{Ch}} = (V_{\text{Ch}}, E_{\text{Ch}})$.

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\mathrm{Ch}} = (V_{\mathrm{Ch}}, E_{\mathrm{Ch}})$.

With $p \in [1, n^{3/2}]$ processors and a free parameter $c \in [1, p^{1/3}]$ [Tiskin 2002] and [S., Demmel 2011] achieve the costs

- computation: $F_{\mathrm{Ch}} = \Theta(n^3/p)$
- bandwidth: $W_{\mathrm{Ch}} = \Theta(n^2/\sqrt{cp})$
- synchronization: $S_{\mathrm{Ch}} = \Theta(\sqrt{cp})$

Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ of dimension n into a lower-triangular matrix $\mathbf{L}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

and has dependency graph $G_{\mathrm{Ch}} = (V_{\mathrm{Ch}}, E_{\mathrm{Ch}})$.

With $p \in [1, n^{3/2}]$ processors and a free parameter $c \in [1, p^{1/3}]$ [Tiskin 2002] and [S., Demmel 2011] achieve the costs

- computation: $F_{\mathrm{Ch}} = \Theta(n^3/p)$
- bandwidth: $W_{\mathrm{Ch}} = \Theta(n^2/\sqrt{cp})$
- synchronization: $S_{\mathrm{Ch}} = \Theta(\sqrt{cp})$

These schedules are optimal since G_{Ch} is an order-3-path-expander about the path corresponding to the diagonal of $\mathbf{L}$.

Other 2.5D algorithms

We give algorithms that attain the costs

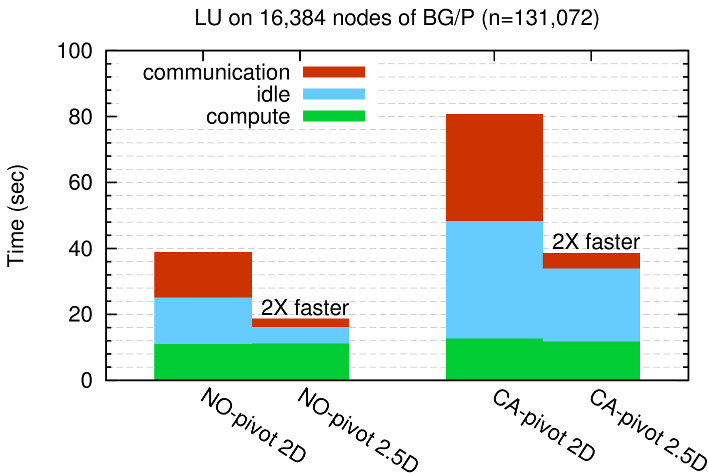
$$F = O(n^3/p) \qquad W = O(n^2/\sqrt{cp}) \qquad O(\sqrt{cp})$$

for the following problems

- LU factorization
- QR factorization
- symmetric eigenvalue problem
- all-pairs shortest-paths problem (Floyd-Warshall algorithm)

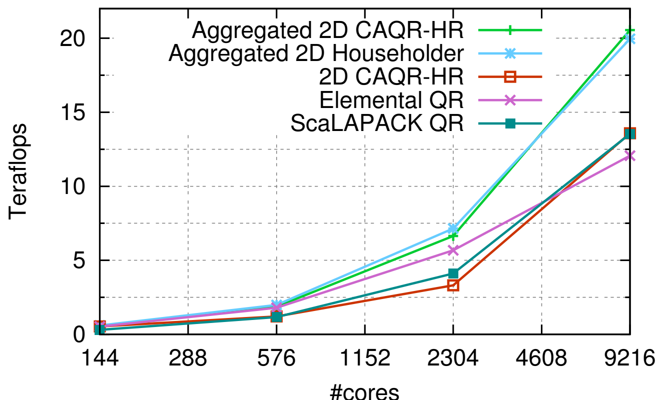
2.5D LU factorization

2.5D LU can be achieved by replicated storage of Schur complement updates and pairwise pivoting [Tiskin 2002] or Communication Avoiding (CA) pivoting



2.5D QR factorization

2.5D QR can be achieved by performing Tall-Skinny QR (TSQR) on each matrix panel, reconstructing the Householder representation, $\mathrm{LU}(Q_1 - I) = Y \cdot (TY_1^T)$, and aggregating the trailing matrix update (left-looking)



2.5D Symmetric eigensolver

We can reduce a symmetric matrix to a banded matrix with the same eigenvalues via a 2.5D algorithm

- instead of applying each trailing symmetric update $\bar{A} := A + UV^T + VU^T$, reformulate computation in the expanded form $A + UV^T + VU^T$ and delay update in order to aggregate
- one-step band-reduction algorithm achieves 2.5D interprocessor communication cost, but needs additional memory-to-cache traffic

2.5D Symmetric eigensolver

We can reduce a symmetric matrix to a banded matrix with the same eigenvalues via a 2.5D algorithm

- instead of applying each trailing symmetric update $\bar{A} := A + UV^T + VU^T$, reformulate computation in the expanded form $A + UV^T + VU^T$ and delay update in order to aggregate
- one-step band-reduction algorithm achieves 2.5D interprocessor communication cost, but needs additional memory-to-cache traffic

We can perform 2.5D SBR (Successive Band Reduction) to reduce the matrix to a small band in more steps

- achieves desired interprocessor and memory-bandwidth costs $Q = W = O(n^2 \log(p)/\sqrt{c \cdot p})$
- requires more computation during back transformations to obtain eigenvectors

We consider the **s**-step Krylov subspace basis computation

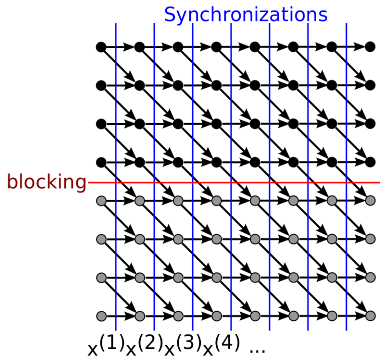
$$\mathbf{x}^{(l)} = \mathbf{A} \cdot \mathbf{x}^{(l-1)},$$

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

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

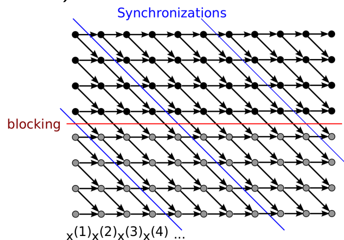
The standard algorithm (1D 2-pt stencil diagram)

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



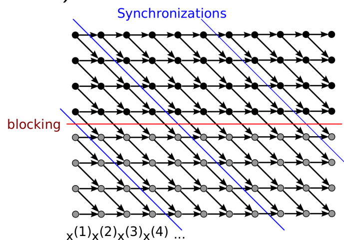
The matrix-powers kernel

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



The matrix-powers kernel

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

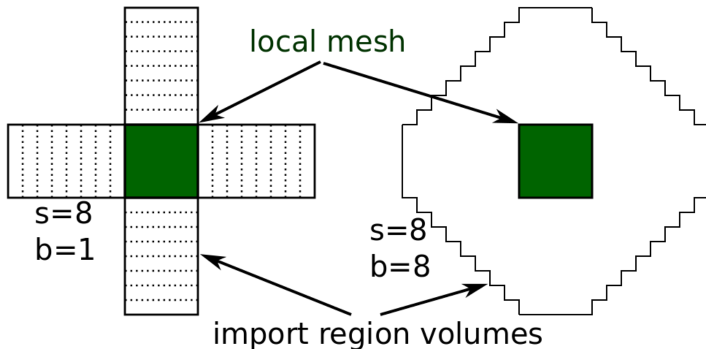


For a $(2m+1)^d$-point stencil, s/b invocations of the matrix-powers kernel compute an s-step Krylov subspace with communication cost

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

and synchronization cost $S_{\mathrm{PA1}} = O(s/b)$

Illustration of import region of the matrix-powers kernel



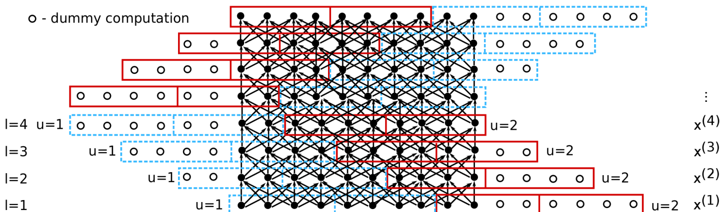
The dependency graph of an s-step $(2m+1)^d$-point Krylov subspace method is an order-$(\mathbf{d}+\mathbf{1})$-path-expander (with prefactors polynomial in m).

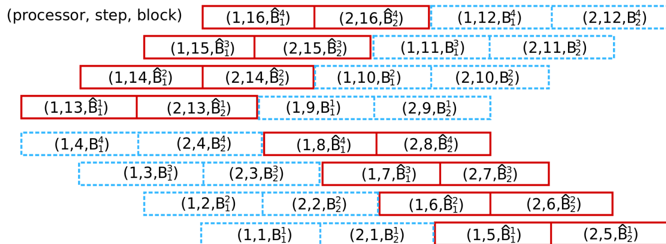
Theorem

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

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

Block the global problem in the time dimension and execute each block in parallel so that working set of data fits into cache





Tensor contractions

We now move on to tensor contractions as motivated by electronic structure applications, e.g. from the Coupled-Cluster Singles and Doubles (CCSD) method

$$Z^{ak}_{i\bar{c}} = \sum_{b\bar{j}} V^{\bar{j}k}_{b\bar{c}} \cdot T^{ab}_{i\bar{j}},$$

where **Z**, **V**, and **T** are fourth order tensors and **T** is antisymmetric in permutation between a and b, but this antisymmetry is 'broken' (a appears in **Z** while b appears in **V**).

We now move on to tensor contractions as motivated by electronic structure applications, e.g. from the Coupled-Cluster Singles and Doubles (CCSD) method

$$Z^{ak}_{i\bar{c}} = \sum_{b\bar{j}} V^{\bar{j}k}_{b\bar{c}} \cdot T^{ab}_{i\bar{j}},$$

where **Z**, **V**, and **T** are fourth order tensors and **T** is antisymmetric in permutation between a and b, but this antisymmetry is 'broken' (a appears in **Z** while b appears in **V**).

Another contraction from the CCSDTQ method is

$$Z^{abc\bar{d}}_{ijk\bar{l}} = \sum_{(a,(b,c))\in\chi(a,b,c)} \sum_{(i,(j,k))\in\chi(i,j,k)} \sum_{\bar{m}} T^{a\bar{d}}_{i\bar{m}} \cdot X^{\bar{m}bc}_{jk\bar{l}},$$

where **Z** is antisymmetric in (a, b, c) and in (i, j, k), while **X** is antisymmetric in (b, c) and (j, k) which are 'preserved'.

Standard algorithm for symmetric tensor contractions

We can express any contraction of fully-symmetric tensors using index set notation $i\langle s+t\rangle = (i_1, \ldots, i_{s+t})$ as

$$C_{i\langle s+t\rangle} = \sum_{(j\langle s\rangle, l\langle t\rangle) \in \chi(i\langle s+t\rangle)} \left(\sum_{k\langle v\rangle} A_{j\langle s\rangle k\langle v\rangle} \cdot B_{l\langle t\rangle k\langle v\rangle} \right)$$

where **C** is order $s+t$, **A** is order $s+v$, and **B** is order $t+v$ and the total number of contraction indices is $\omega = s+t+v$.

Standard algorithm for symmetric tensor contractions

We can express any contraction of fully-symmetric tensors using index set notation $i\langle s+t\rangle = (i_1, \ldots, i_{s+t})$ as

$$C_{i\langle s+t\rangle} = \sum_{(j\langle s\rangle, l\langle t\rangle) \in \chi(i\langle s+t\rangle)} \left(\sum_{k\langle v\rangle} A_{j\langle s\rangle k\langle v\rangle} \cdot B_{l\langle t\rangle k\langle v\rangle} \right)$$

where $\mathbf{C}$ is order $s+t$, $\mathbf{A}$ is order $s+v$, and $\mathbf{B}$ is order $t+v$ and the total number of contraction indices is $\omega = s+t+v$.

The standard method for tensor contractions, algorithm $\Phi^{(s,t,v)}$ exploits preserved symmetries: $j\langle s\rangle$, $l\langle t\rangle$, and $k\langle v\rangle$ to achieve the cost

$$T_\Phi(n,s,t,v) = \binom{n}{s}\binom{n}{t}\binom{n}{v} \cdot (\nu + \mu) + O(n^{\omega-1})$$

where ν is the cost of additions, μ is the cost of multiplications, and n is the dimension (range of each index).

Parallelization of the standard contraction algorithm

The standard symmetric contraction algorithm $\Phi^{(s,t,v)}$ 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)

- a library for distributed memory decomposition, redistribution, and contraction of tensors

Parallelization of the standard contraction algorithm

The standard symmetric contraction algorithm $\Phi^{(s,t,v)}$ 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)

- a library for distributed memory decomposition, redistribution, and contraction of tensors
- two-level parallelization via MPI+OpenMP

Parallelization of the standard contraction algorithm

The standard symmetric contraction algorithm $\Phi^{(s,t,v)}$ 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)

- a library for distributed memory decomposition, redistribution, and contraction of tensors
- two-level parallelization via MPI+OpenMP
- automated topology-aware runtime mapping of tensors via performance models

Parallelization of the standard contraction algorithm

The standard symmetric contraction algorithm $\Phi^{(s,t,v)}$ 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)

- a library for distributed memory decomposition, redistribution, and contraction of tensors
- two-level parallelization via MPI+OpenMP
- automated topology-aware runtime mapping of tensors via performance models
- supports distributed packed symmetric storage

Parallelization of the standard contraction algorithm

The standard symmetric contraction algorithm $\Phi^{(s,t,v)}$ 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)

- a library for distributed memory decomposition, redistribution, and contraction of tensors
- two-level parallelization via MPI+OpenMP
- automated topology-aware runtime mapping of tensors via performance models
- supports distributed packed symmetric storage
- uses 2.5D matrix multiplication for contractions

Parallelization of the standard contraction algorithm

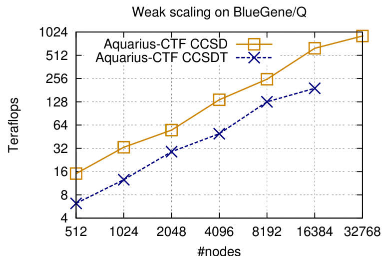
The standard symmetric contraction algorithm $\Phi^{(s,t,v)}$ is dominated by a matrix multiplication of an $\binom{n}{s}$-by-$\binom{n}{v}$ matrix with an $\binom{n}{v}$-by-$\binom{n}{t}$ matrix into an $\binom{n}{s}$-by-$\binom{n}{t}$ matrix

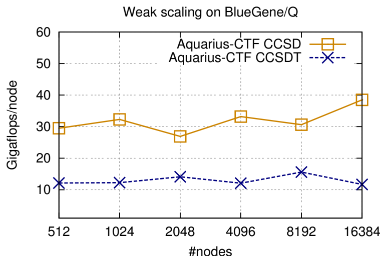
- the main parallelization challenge is to get the tensor data into the desired matrix layout

Cyclops Tensor Framework (CTF)

- a library for distributed memory decomposition, redistribution, and contraction of tensors
- two-level parallelization via MPI+OpenMP
- automated topology-aware runtime mapping of tensors via performance models
- supports distributed packed symmetric storage
- uses 2.5D matrix multiplication for contractions
- provides concise interface for tensor objects and contractions

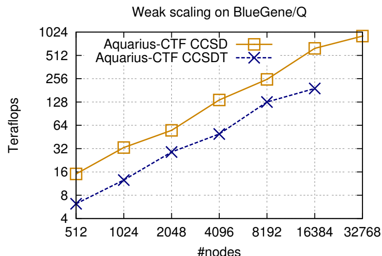
CCSD up to 55 (50) water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ

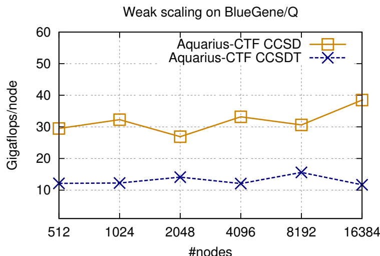


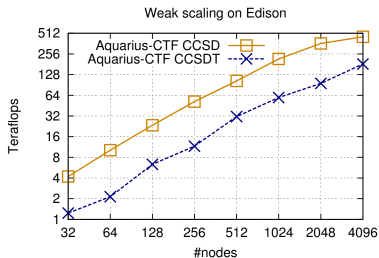


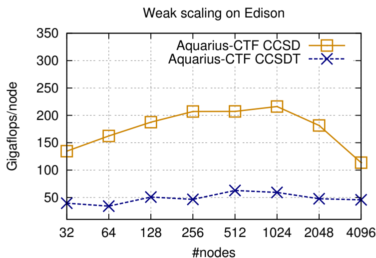
Coupled Cluster on IBM BlueGene/Q

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





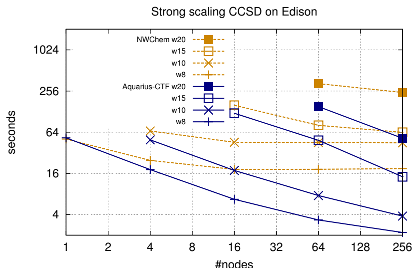


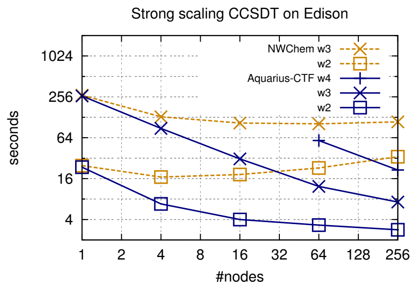


Comparison with NWChem

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)





Fast symmetric contraction algorithm

Recall that the standard algorithm $\Psi^{(s,t,v)}$ computes symmetric tensor contractions using $\frac{\mathbf{n}^{\omega}}{\mathbf{s!t!v!}}$ multiplications (where $\omega = s + t + v$).

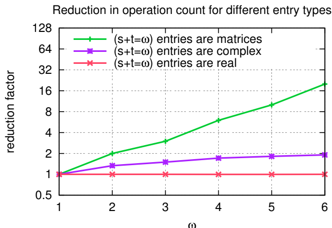
Fast symmetric contraction algorithm

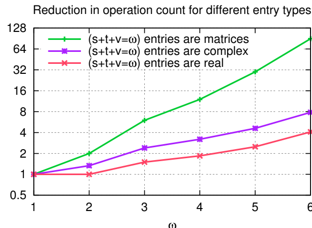
Recall that the standard algorithm $\Psi^{(s,t,v)}$ computes symmetric tensor contractions using $\frac{\mathbf{n}^{\omega}}{\mathbf{s}!\mathbf{t}!\mathbf{v}!}$ multiplications (where $\omega = s + t + v$).
We give an algorithm $\Phi^{(s,t,v)}$ that requires only $\frac{\mathbf{n}^{\omega}}{\omega!}$ multiplications but more additions.

Fast symmetric contraction algorithm

Recall that the standard algorithm $\Psi^{(s,t,v)}$ computes symmetric tensor contractions using $\frac{\mathbf{n}^{\omega}}{\mathbf{s!t!v!}}$ multiplications (where $\omega = s + t + v$).
We give an algorithm $\Phi^{(s,t,v)}$ that requires only $\frac{\mathbf{n}^{\omega}}{\omega!}$ multiplications but more additions. The speed-up depends on the relative cost of additions ν and multiplications μ. When multiplications are

- real floating point multiplications $\mu = \nu$
- complex floating point multiplications $\mu = 3\nu$
- matrix-matrix multiplications $\mu \gg \nu$





Fast symmetric algorithm

The algorithm $\Phi^{(s,t,v)}$ computes $\binom{n}{\omega}$ multiplications to leading order, corresponding to an order ω symmetric tensor

$$\hat{Z}_{i\langle\omega\rangle} = \left(\sum_{j\langle s+v\rangle \in \chi(i\langle\omega\rangle)} A_{j\langle s+v\rangle}\right) \cdot \left(\sum_{l\langle t+v\rangle \in \chi(i\langle\omega\rangle)} B_{l\langle t+v\rangle}\right).$$

This term includes all needed multiplications to compute **C** and some extra ones, which can be cancelled out (usually) with low order cost.

Fast symmetric algorithm

The algorithm $\Phi^{(s,t,v)}$ computes $\binom{n}{\omega}$ multiplications to leading order, corresponding to an order ω symmetric tensor

$$\hat{Z}_{i\langle\omega\rangle} = \left(\sum_{j\langle s+v\rangle \in \chi(i\langle\omega\rangle)} A_{j\langle s+v\rangle} \right) \cdot \left(\sum_{l\langle t+v\rangle \in \chi(i\langle\omega\rangle)} B_{l\langle t+v\rangle} \right).$$

This term includes all needed multiplications to compute $\mathbf{C}$ and some extra ones, which can be cancelled out (usually) with low order cost.

For example, when $s = 1, t = 0, v = 1$, the contraction is a multiplication of a vector by a symmetric matrix. The algorithm $\Psi^{(s,t,v)}$ ignores the symmetry of the matrix, while $\Phi^{(s,t,v)}$ computes

$$\hat{Z}_{ij} = A_{ij} \cdot (b_i + b_j), \qquad Z_i = \sum_k \hat{Z}_{ik},$$

$$A_i^{(1)} = \sum_k A_{ik}, \quad V_i = A_i^{(1)} \cdot b_i, \quad c_i = Z_i - V_i.$$

Analysis of the fast symmetric algorithm

Stability analysis:

- theoretical error bound weaker only by factors polynomial in ω and numerical experiments suggest error difference between $\Psi^{(s,t,v)}$ and $\Phi^{(s,t,v)}$ is insignificant

Analysis of the fast symmetric algorithm

Stability analysis:

- theoretical error bound weaker only by factors polynomial in ω and numerical experiments suggest error difference between $\Psi^{(s,t,v)}$ and $\Phi^{(s,t,v)}$ is insignificant

Communication cost analysis:

- $\Psi^{(s,t,v)}$ has communication cost proportional to matrix multiplication (dependency graph is a $(3,2)$-lattice hypergraph)

$$W_\Psi = \Theta(n^\omega / p^{2/3})$$

- $\Phi^{(s,t,v)}$ does more communication per multiplication (sometimes asymptotically) since its dependency graph is a $(\omega, \max[s+v, t+v, s+t])$-lattice hypergraph

$$W_\Phi = \Theta(n^\omega / p^{\max(s+v,t+v,s+t)/\omega})$$

Application of the fast symmetric algorithm to CCSD

Let $\Upsilon^{(s,t,v)}$ be the nonsymmetric contraction algorithm. Recall the CCSD contraction

$$Z^{ak}_{i\bar{c}} = \sum_{b\bar{j}} V^{\bar{j}k}_{b\bar{c}} \cdot T^{ab}_{i\bar{j}},$$

where $\mathbf{T}$ is antisymmetric in (a, b) the standard algorithm $\Psi^{(0,1,1)}\Upsilon^{(2,1,1)}(\mathbf{V}, \mathbf{T})$ has cost $2n^6$ (same as just $\Upsilon^{(2,2,2)}(\mathbf{V}, \mathbf{T})$). The fast algorithm can be used as

$$\Phi^{(0,1,1)}\Upsilon^{(2,1,1)}(\mathbf{V}, \mathbf{T})$$

with cost n^6 (since the 2X reduction in multiplications from $\Phi^{(0,1,1)}$ results in 2X fewer calls to $\Upsilon^{(2,1,1)}$).

Application of the fast symmetric algorithm to CCSDTQ

Recall the CCSDTQ contraction

$$Z^{abc\bar{d}}_{ijk\bar{l}} = \sum_{(a,(b,c))\in\chi(a,b,c)} \sum_{(i,(j,k))\in\chi(i,j,k)} \sum_{\bar{m}} T^{a\bar{d}}_{i\bar{m}} \cdot X^{\bar{m}bc}_{jk\bar{l}},$$

where $\mathbf{Z}$ is antisymmetric in (a, b, c) and (i, j, k), which can be computed as

$$\mathbf{Z} = \Psi^{(2,1,0)}\Psi^{(1,2,0)}\Upsilon^{(1,1,1)}(\mathbf{T}, \mathbf{X}),$$

with $n^9/2$ operations (4X savings from symmetry) or via

$$\mathbf{Z} = \Phi^{(2,1,0)}\Phi^{(1,2,0)}\Upsilon^{(1,1,1)}(\mathbf{T}, \mathbf{X}),$$

with $n^9/18$ operations (36X savings from symmetry).

Future work

- Implementations of 2.5D QR, symmetric eigensolver, and Tiskin's all-pairs shortest-paths algorithm
- Implementation and extensions of communication-efficient Krylov subspace method parallelization (e.g. Gram matrix handling and explicit-matrix case)
- Sparse adaptation and further optimization of Cyclops Tensor Framework
- Derivation of full coupled-cluster methods using fast symmetric contractions
- High performance implementation of the fast symmetric contraction algorithm

Backup slides

Collaborators and acknowledgements

Collaborators:

- James Demmel (UC Berkeley)
- Grey Ballard (formerly UC Berkeley, now Sandia National Laboratory)
- Nicholas Knight, Erin Carson, James Demmel, Kathy Yelick (UC Berkeley)
- Devin Matthews (UT Austin)
- Jeff Hammond (Argonne National Laboratory)

Grants:

- Krell DOE Computational Science Graduate Fellowship

$$\hat{Z}_{i\langle\omega\rangle} = \left(\sum_{j\langle s+v\rangle \in \chi(i\langle\omega\rangle)} A_{j\langle s+v\rangle}\right) \cdot \left(\sum_{l\langle t+v\rangle \in \chi(i\langle\omega\rangle)} B_{l\langle t+v\rangle}\right)$$

$$Z_{i\langle s+t\rangle} = \sum_{k\langle v\rangle} \hat{Z}_{i\langle s+t\rangle k\langle v\rangle}$$

$$\begin{aligned}
V_{i\langle s+t\rangle} = &\sum_{r=0}^{v-1} \binom{v}{r} \cdot \sum_{p=\max(0,v-t-r)}^{v-r} \binom{v-r}{p} \cdot \\
&\sum_{q=\max(0,v-s-r)}^{v-p-r} \binom{v-p-r}{q} \cdot n^{v-p-q-r} \cdot \\
&\sum_{k\langle r\rangle} \left[\sum_{j\langle s+v-p-r\rangle \in \chi(i\langle s+t\rangle)} \left(A^{(p)}_{j\langle s+v-p-r\rangle k\langle r\rangle}\right)\right] \\
&\cdot \left[\sum_{l\langle t+v-q-r\rangle \in \chi(i\langle s+t\rangle)} \left(B^{(q)}_{l\langle t+v-q-r\rangle k\langle r\rangle}\right)\right]
\end{aligned}$$

$$W_{i\langle s+t\rangle} = \sum_{r=1}^{\min(s,t)} \left(\sum_{(m\langle r\rangle, h\langle s+t-2r\rangle) \in \chi(i\langle s+t\rangle)} U^{(r)}_{m\langle r\rangle h\langle s+t-2r\rangle}\right)$$

$$C_{i\langle s+t\rangle} = Z_{i\langle s+t\rangle} - V_{i\langle s+t\rangle} - W_{i\langle s+t\rangle}$$

where $\mathbf{A}^{(0)} = \mathbf{A}$ and $\mathbf{B}^{(0)} = \mathbf{B}$ and

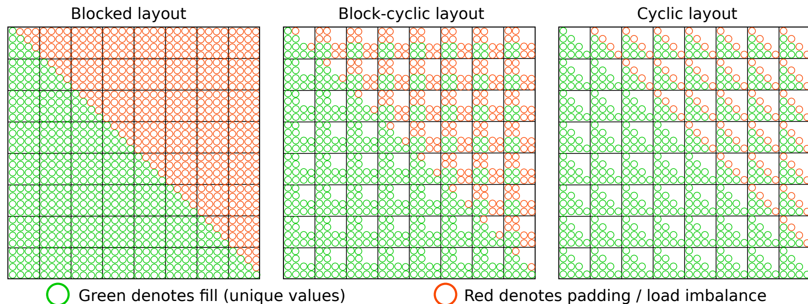
$$\forall p \in [1, v],\ \forall i\langle s+v-p\rangle,\ A^{(p)}_{i\langle s+v-p\rangle} = \sum_{k\langle p\rangle} A_{i\langle s+v-p\rangle k\langle p\rangle},$$

$$\forall q \in [1, v],\ \forall i\langle t+v-q\rangle,\ B^{(q)}_{i\langle t+v-q\rangle} = \sum_{k\langle q\rangle} B_{i\langle t+v-q\rangle k\langle q\rangle},$$

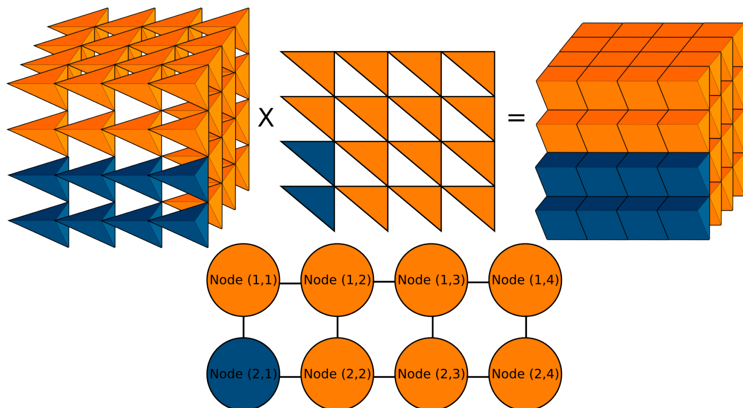
$$\forall r \in [1, \min(s,t)],\quad \forall i\langle s+t-2r\rangle,$$

$$U^{(r)}_{m\langle r\rangle i\langle s+t-2r\rangle} = \sum_{(j\langle s-r\rangle, l\langle t-r\rangle)\in\chi(i\langle s+t-2r\rangle)} \left(\sum_{k\langle v\rangle} A_{m\langle r\rangle j\langle s-r\rangle k\langle v\rangle} \cdot B_{m\langle r\rangle l\langle t-r\rangle k\langle v\rangle}\right).$$

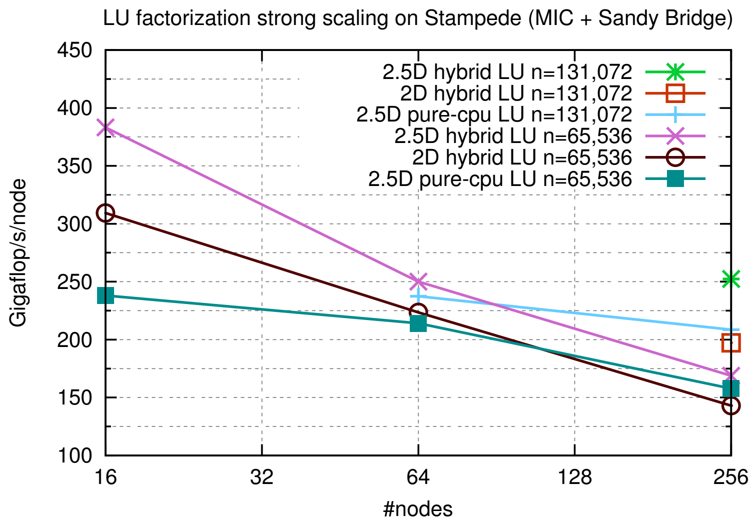
Blocked vs block-cyclic vs cyclic decompositions



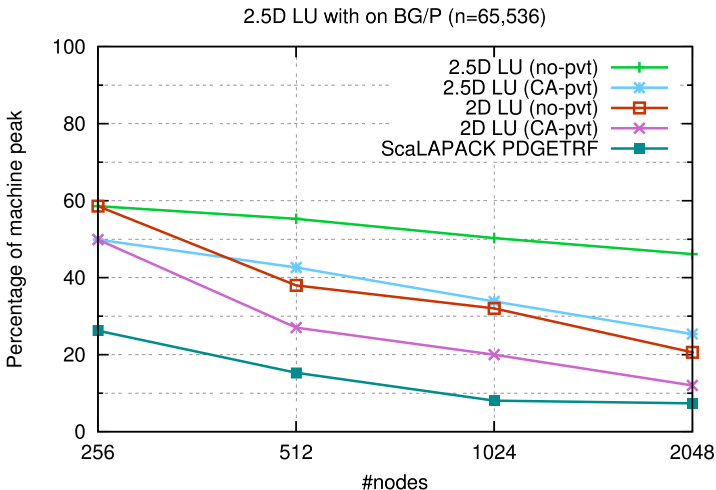
3D tensor mapping



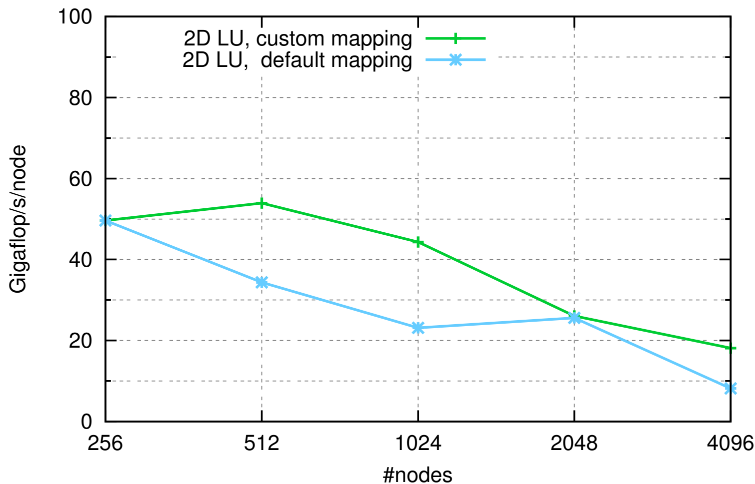
2.5D LU on MIC



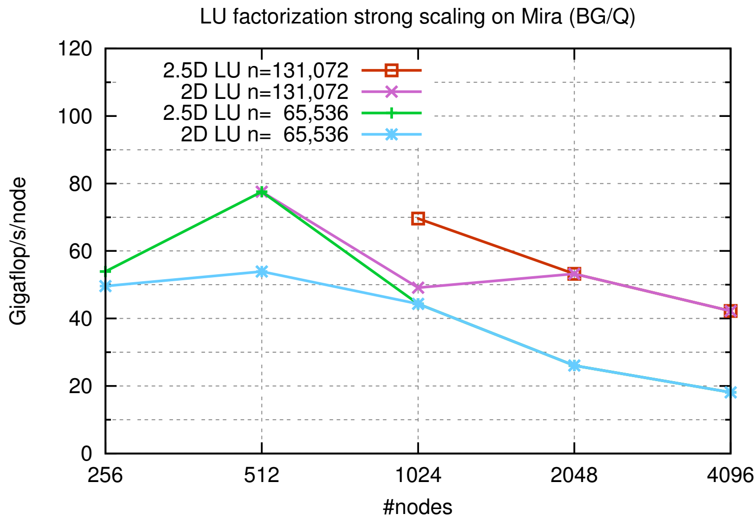
2.5D LU strong scaling



Topology-aware mapping on BG/Q



Benefit of replication on BG/Q



Parallel costs of Gauss-Jordan elimination

The floating point cost of Gauss-Jordan elimination is $F = \Theta(n^3/p)$. Our lower bounds may be applied since the computation has the same structure as Gaussian Elimination, so

$$F \cdot S^2 = \Omega(n^3), \quad W \cdot S = \Omega(n^2).$$

These costs are achieved for $W = O(n^2/p^{2/3})$ by schedules in

- Aggarwal, Chandra, and Snir 1990
- Tiskin 2007
- Solomonik, Buluc, and Demmel 2012

Lower synchronization cost via path doubling

We can compute the tropical semiring closure

$$\mathbf{A}^* = \mathbf{I} \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \ldots \oplus \mathbf{A}^n = (\mathbf{I} \oplus \mathbf{A})^n,$$

directly via repeated squaring (path-doubling)

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

with a total of $\log(n)$ matrix-matrix multiplications, with

$$F = O(n^3 \log(n)/p)$$

operations and $O(\log(n))$ synchronizations, which can be less than the $O(p^{1/2})$ required by Floyd-Warshall.

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

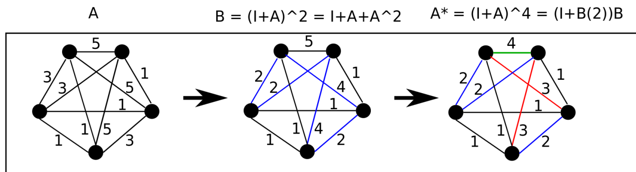
$$\mathbf{A}^l(l) \leq \mathbf{A}^{l+1}(l) \leq \ldots \leq \mathbf{A}^n(l) = \mathbf{A}^*(l),$$

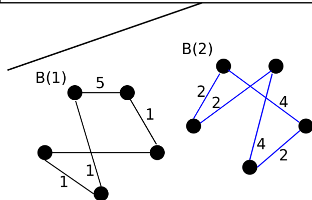
in particular $\mathbf{A}^*(l)$ corresponds to a sparse subset of $\mathbf{A}^l(l)$.
The algorithm works by picking $l \in [k/2, k]$ and computing

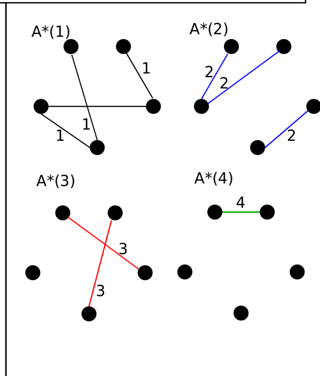
$$(\mathbf{I} \oplus \mathbf{A})^{3k/2} \leq (\mathbf{I} \oplus \mathbf{A}^k(l)) \otimes \mathbf{A}^k,$$

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.

Path-doubling (Tiskin's algorithm)







Path-doubling

Earlier caveat:

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

does not hold in general. The fundamental property used by the algorithm is really

$$\mathbf{A}^*(l) \otimes \mathbf{A}^*(k) = \mathbf{A}^*(l+k).$$

All shortest paths of up to any length are composible (factorizable), but not paths up to a limited length. However, the algorithm is correct because $\mathbf{A}^l \leq \mathbf{A}^k(l) \leq \mathbf{A}^*(k)$.

Cost of Tiskin's algorithm

Since the decomposition by path size is disjoint, one can pick $\mathbf{A}^k(l)$ for $l \in [k/2, k]$ to have size

$$|\mathbf{A}^k(l)| \geq 2n^2/k.$$

Each round of path doubling becomes cheaper than the previous, so the cost is dominated by the first matrix multiplication,

$$F = O(n^3/p) \quad W = O(n^2/p^{2/3}) \quad S = O(\log(n)),$$

solving the APSP problem with no $F \cdot S^2$ or $W \cdot S$ tradeoff and optimal flops.

More on Tiskin's APSP algorithms

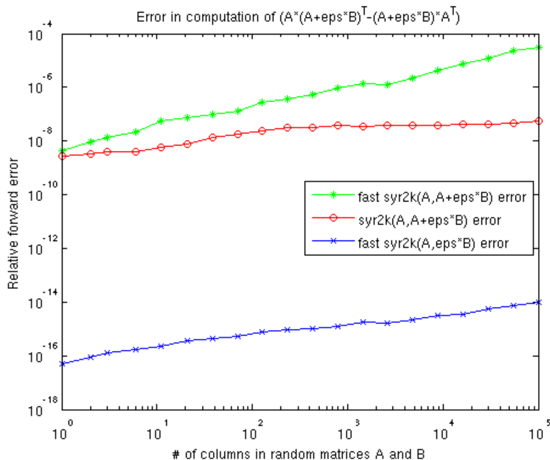
Tiskin gives a way to lower the synchronization from $S = O(\log(n))$ to $O(\log(p))$. For nonnegative edge lengths it is straightforward

- compute $\mathbf{A}^p$ via path-doubling
- pick a small $\mathbf{A}^p(l)$ for $l \in [p/2, p]$
- replicate $\mathbf{A}^p(l)$ and compute Dijkstra's algorithm for n/p nodes with each process, obtaining $(\mathbf{A}^p(l))^*$
- compute by matrix multiplication

$$\mathbf{A}^* = (\mathbf{A}^p(l))^* \otimes \mathbf{A}^p$$

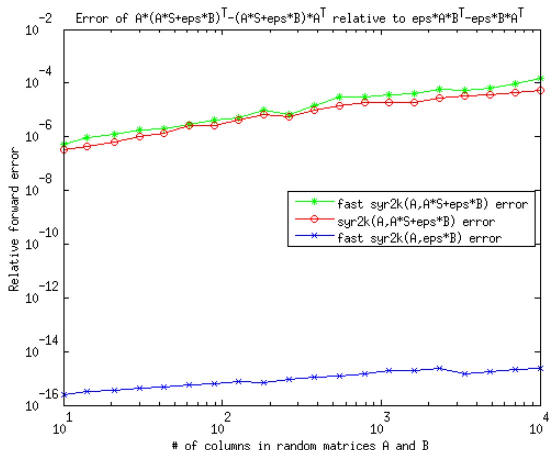
since all shortest paths are composed of a path of size that is a multiple of $l \le p$, followed by a shortest path of size up to p

Numerical test I



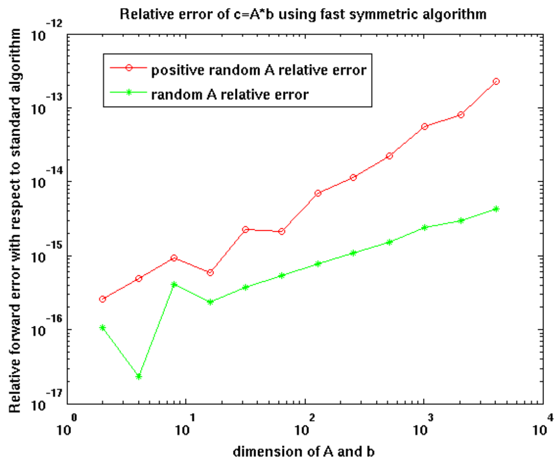
We measure the error in computation of $\mathbf{A} \cdot \mathbf{B}^T - \mathbf{B} \cdot \mathbf{A}^T$ where $\mathbf{B} = \mathbf{A} + \epsilon \cdot \bar{\mathbf{B}}$ for $\epsilon = 10^{-9}$ (antisymmetric rank-$2K$ update).

Numerical test II



Now we consider the computation of $\mathbf{A} \cdot \mathbf{B}^T - \mathbf{B} \cdot \mathbf{A}^T$ where $\mathbf{B} = \mathbf{A} \cdot \mathbf{S} + \epsilon \cdot \bar{\mathbf{B}}$ where $\mathbf{S}$ is a random symmetric matrix.

Numerical test III



Numerical test IV