Provably efficient algorithms for multilinear algebra

Edgar Solomonik

Department of Computer Science
ETH Zurich

University of California, Davis

March 28, 2016

Outline and highlights

1. Communication-optimal algorithms for linear solvers
   - algorithms with $p^{1/6}$ less communication on p processors for LU, QR, eigs
   - topology-aware implementations: 12X speed-up for MM, 2X for LU
   - novel lower bounds on communication and synchronization
2. Tensor (multidimensional matrix) computations
   - Cyclops Tensor Framework (CTF): first distributed-memory tensor contraction framework
   - sparse multidimensional arrays, arbitrary types, semirings
3. Massively-parallel electronic structure calculations
   - codes using CTF for wavefunction methods: Aquarius, QChem, VASP, Psi4
   - coupled cluster faster than NWChem by $>$ 10X, nearly 1 petaflop/s
4. Preserving symmetry in tensor contractions
   - factor of $\omega!$ fewer multiplications for symmetric contractions of cost n^{ω}
   - up to 9X speed-up for partially-symmetric contractions in coupled cluster

Cost model for parallel algorithms

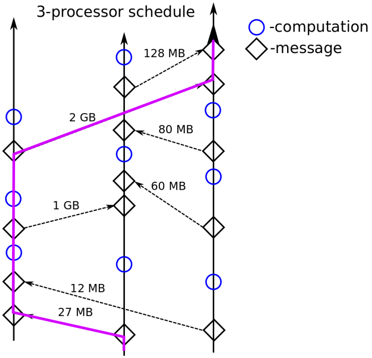
Algorithms should minimize communication, not just computation

- data movement and synchronization cost more energy than flops
- two types of data movement:
  - vertical (intranode memory–cache)
  - horizontal (internode network transfers)
- synchronization: number of messages, latency

Critical path costs

Given a schedule consider the following costs, accumulated along chains of tasks (as in $\alpha - \beta$, BSP, and LogGP models):

- F – computation cost
- Q – vertical communication cost
- W – horizontal communication cost
- S – synchronization cost



Communication lower bounds: previous work

Multiplication of $n \times n$ matrices

- horizontal communication lower bound[1]

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

- memory-dependent horizontal communication lower bound[2]

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

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

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

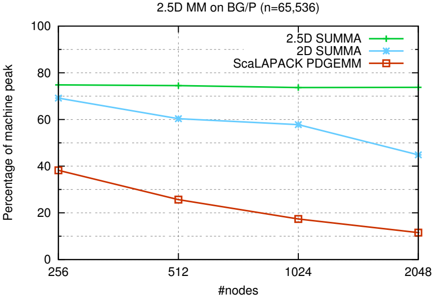
- libraries like ScaLAPACK, Elemental optimal only for $c = 1$

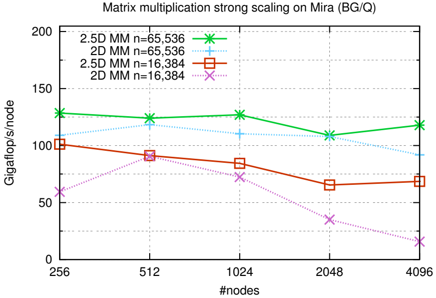
[1] Aggarwal, Chandra, Snir, TCS, 1990

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

Communication-avoiding algorithms for matrix multiplication have been studied extensively[1]

They continue to be attractive on modern architectures[2]





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

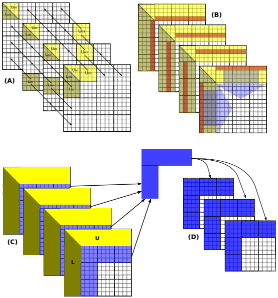
[1] Berntsen, Par. Comp., 1989; Agarwal, Chandra, Snir, TCS, 1990; Agarwal, Balle, Gustavson, Joshi, Palkar, IBM, 1995; McColl, Tiskin, Algorithmica, 1999; ...

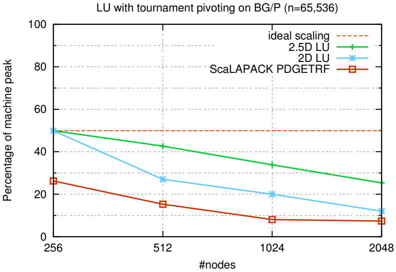
[2] S., Bhatele, Demmel, SC, 2011

Communication-efficient LU factorization

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

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





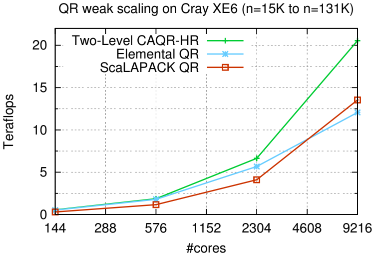
- LU with pairwise pivoting[1] extended to tournament pivoting[2]
- first implementation of a communication-optimal LU algorithm[2]

[1] Tiskin, FGCS, 2007

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

Communication-efficient QR factorization

- $W_{\mathrm{QR}} = O(n^2/\sqrt{cp})$ using Givens rotations[1]
- Householder form can be reconstructed quickly from TSQR[2]
$$Q = I - YTY^T \qquad \Rightarrow \qquad \mathrm{LU}(I - Q) \rightarrow (Y, TY^T)$$
- enables communication-optimal Householder QR[3]
- Householder aggregation yields performance improvements



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

[1] Tiskin, FGCS, 2007

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

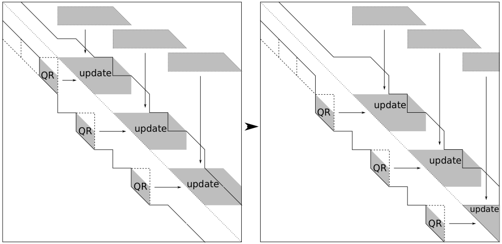
[3] S., UCB, 2014

Communication-efficient eigenvalue computation

For the dense symmetric matrix eigenvalue problem[1]

$$W_{\mathrm{SE}} = O(n^2/\sqrt{cp})$$

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



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

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

Synchronization cost lower bounds

Unlike matrix multiplication, dense matrix factorizations have polynomial depth (contain a long dependency path)

Given $M = cn^2/p$ memory:

- matrix multiplication synchronization cost bound[1]

$$S_{\text{MM}} = \Theta\left(\sqrt{p/c^3} + \log p\right)$$

- algorithms for Cholesky, LU, QR, SVD do not attain this bound

$$S_{\text{LU}} = \Theta\left(\sqrt{cp}\right)$$

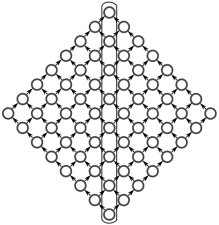
- need smaller block size for lower communication cost $\rightarrow$ higher synchronization cost

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

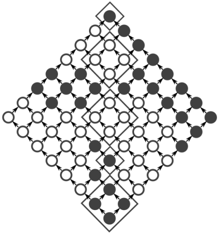
Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the $n \times n$ diamond DAG,[1]

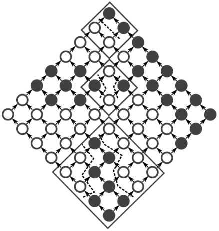
$$F \cdot S = \Omega(n^2)$$



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

We generalize this idea[2]

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

[1] Papadimitriou, Ullman, SIAM JC, 1987

[2] 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:[1]

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

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

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

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

Therefore, the costs

$$W_{\mathsf{CHOL}} = \Theta(n^2/\sqrt{cp}), \quad S_{\mathsf{CHOL}} = \Theta(\sqrt{cp}),$$

are optimal

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

Synchronization tradeoffs in stencils

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

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

- time-blocking lowers synchronization and vertical communication costs, but raises horizontal communication
- we suggest alternative approach that minimizes vertical and horizontal communication, but not synchronization
- further directions:
  - implementation of proposed algorithm
  - lower bounds for graph traversals

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

Bridging the gap between algorithms and applications

How can we package communication-avoiding algorithms for distributed-memory programs?

- challenges: complicated data layouts (multidimensional processor grids, cyclic distributions), global coordination, large tuning space
- need high-level abstractions, algebraic language, interoperability
- solution: library for algebraic multidimensional array computations
  - Cyclops Tensor Framework: C++ library, MPI+OpenMP+BLAS(+CUDA) `https://github/com/solomonik/ctf`
  - first lets consider matrices/vectors, then higher-order tensors

A library for tensor computations

Cyclops Tensor Framework[1]

- implicit for loops based on index notation (Einstein summation)
- matrix sums, multiplication, Hadamard product (tensor contractions)
- distributed symmetric-packed/sparse storage via cyclic layout

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

A library for tensor computations

Cyclops Tensor Framework

- implicit for loops based on index notation (Einstein summation)
- matrix sums, multiplication, Hadamard product (tensor contractions)
- distributed symmetric-packed/sparse storage via cyclic layout

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

```
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){
  ... // split A = R + diag(1./d)
  do {
    x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
    r["i"] = b["i"]-A["ij"]*x["j"]; // compute residual
  } while (r.norm2() > 1.E-6); // check for convergence
  return x;
}
```

A library for tensor computations

Cyclops Tensor Framework

- implicit for loops based on index notation (Einstein summation)
- matrix sums, multiplication, Hadamard product (tensor contractions)
- distributed symmetric-packed/sparse storage via cyclic layout

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

```
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){
  Matrix<> R(A);
  R["ii"] = 0.0;
  Vector<> x(n), d(n), r(n);
  Function<> inv([](double & d){ return 1./d; });
  d["i"] = inv(A["ii"]); // set d to inverse of diagonal of A
  do {
    x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
    r["i"] = b["i"]-A["ij"]*x["j"]; // compute residual
  } while (r.norm2() > 1.E-6); // check for convergence
  return x;
}
```

Algebraic shortest path computations

Tropical (geodetic) semiring

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

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

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

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

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

Can also express all-pairs shortest-paths (APSP) using tropical semiring

With other semirings/element-wise functions can formulate *any graph algorithm that updates adjacent edge $\leftrightarrow$ vertex labels*

Bellman–Ford Algorithm using CTF

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

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

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

... // Initialize A and v

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

Betweenness centrality

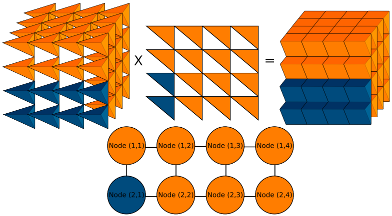
Betweenness centrality code snippet, for k of n nodes

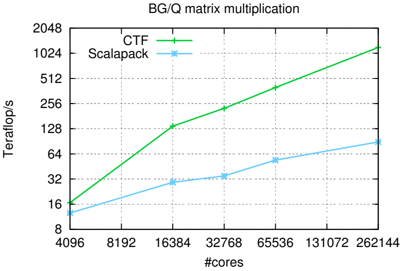
```
void btwn_central(Matrix<int> A, Matrix<path> P, int n, int k){
  Monoid<path> mon(...,
                  [](path a, path b){
                    if (a.w<b.w) return a;
                    else if (b.w<a.w) return b;
                    else return path(a.w, a.m+b.m);
                  }, ...);

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

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

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

Performance of CTF for dense computations

CTF is highly tuned for massively-parallel machines

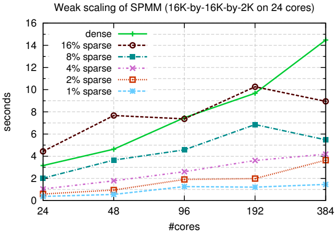
- virtualized multidimensional processor grids
- topology-aware mapping and collective communication
- *performance-model-driven algorithm selection done at runtime*
- optimized redistribution kernels for matrix/tensor transposition

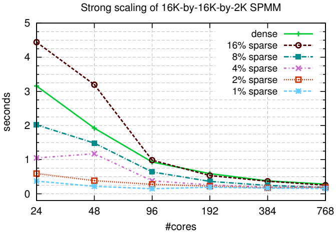




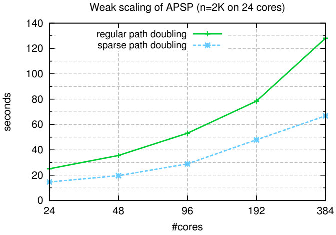
Performance of CTF for sparse computations

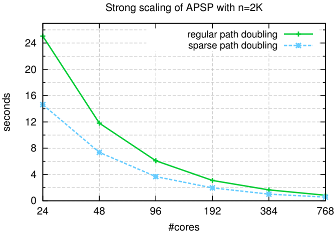
multiplication of a sparse matrix and a dense matrix[1]





All-pairs shortest-paths based on path doubling with sparsification[1]





[1] S., Hoefler, Demmel, arXiv, 2015

Tensor computations as programming abstractions

Tensors (scalars, vectors, matrices, etc.) are convenient abstractions for multidimensional data

- one type of object for any homogeneous dataset
- enable expression of symmetries, sparsity

Matrix computations $\subset$ tensor computations

- $=$ often reduce to or employ matrix algorithms
  - can leverage high performance matrix libraries
- $+$ high-order tensors can 'act' as many matrix unfoldings
- $+$ symmetries lower memory footprint and cost
- $+$ tensor factorizations (CP, Tucker, tensor train, ...)

Applications of high-order tensor representations

Numerical solution to differential equations

- higher-order Taylor series expansion terms
- nonlinear terms and differential operators

Computer vision and graphics

- 2D image $\otimes$ angle $\otimes$ time
- classification, compression (tensor factorizations, sparsity)

Machine learning

- convolutional neural networks, high-order statistics
- reduced-order models, recommendation systems (tensor factorizations)

Graph computations

- hypergraphs, time-dependent graphs
- clustering/partitioning/path-finding (eigenvector computations)

Divide-and-conquer algorithms representable by tensor folding

- bitonic sort, FFT, scans

Tensors for computational chemistry/physics

Manybody Schrödinger equation

- "curse of dimensionality" – exponential state space

Condensed matter physics

- tensor network models (e.g. DMRG), tensor per lattice site
- highly symmetric multilinear tensor representation
- exponential state space localized $\rightarrow$ factorized tensor form

Quantum chemistry (**electronic structure calculations**)

- models of molecular structure and chemical reactions
- methods for calculating electronic correlation:
  - "Post Hartree-Fock": configuration interaction, **coupled cluster**, Møller-Plesset perturbation theory
- multi-electron states as tensors,
  e.g. electron $\otimes$ electron $\otimes$ orbital $\otimes$ orbital
- nonlinear equations of partially (anti)symmetric tensors
- interactions diminish with distance $\rightarrow$ sparsity, low rank

Coupled cluster using CTF

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

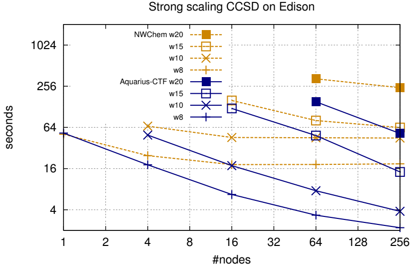
```
FMI["mi"]      += 0.5*WMNEF["mnef"]*T2["efin"];
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T2["efij"];
FAE["ae"]      -= 0.5*WMNEF["mnef"]*T2["afmn"];
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T2["afin"];

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

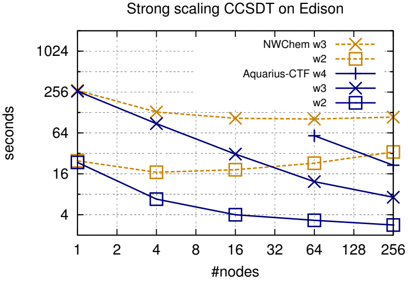
CTF is used within **Aquarius, QChem, VASP, and Psi4**

Comparison with NWChem

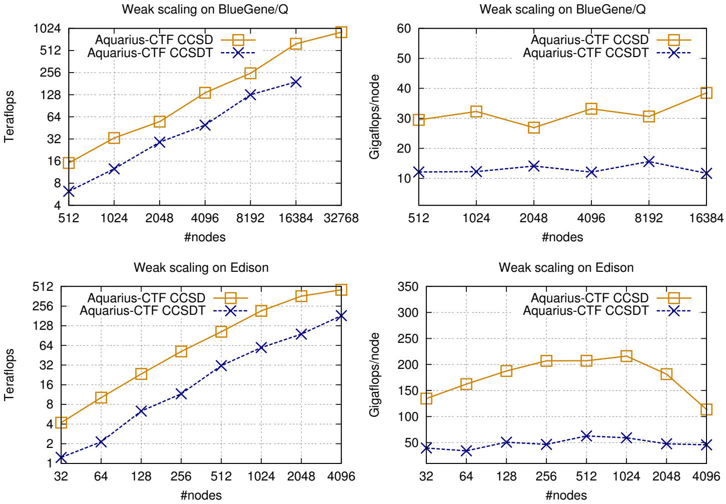
NWChem is the most commonly-used distributed-memory quantum chemistry method suite

- provides Coupled Cluster methods: CCSD and CCSDT
- derives equations via Tensor Contraction Engine (TCE)
- generates contractions as blocked loops leveraging Global Arrays





CCSD up to 55 (50) water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ[1]



[1] S., Matthews, Hammond, Demmel, JPDC, 2014

Symmetry preserving algorithms

Tensor symmetry (e.g. $A_{ij} = A_{ji}$) reduces memory and cost[1]

- for order d tensor, $d!$ less memory
- matrix-vector multiplication $(A_{ij} = A_{ji})$[1]

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

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

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

- numerically stable (by forward error bounds and in experiments)
- lower overall cost for partially symmetric contractions
- up to 9X for select contractions, 1.3X/2.1X for CCSD/CCSDT
- Hermitian BLAS/LAPACK operations with 25% less cost
- ongoing: relationship to fast structured matrix multiplication

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

Impact and future work

Many further directions (theory, implementation, application)
with much overlap (tensor equations, factorizations, symmetries)

- Cyclops Tensor Framework
  - ✓ already widely-adapted in quantum chemistry, many requests for features
  - integrate matrix/tensor factorizations
  - optimize across multiple tensor operations (scheduling, layout persistence)
  - *engage high-impact application domains*
    - tensor networks for condensed matter-physics
    - convolutional neural networks
    - recommender systems (tensor completion)
- communication-avoiding algorithms
  - ✓ existing fast implementations already used by applications (e.g. QBall)
  - find efficient methods of searching larger tuning spaces
  - algorithms for computing eigenvectors, SVD, tensor factorizations
  - analyze (randomized) algorithms for sparse matrix factorization
- symmetry in tensor computations
  - cost improvements ✓ $\rightarrow$ library implementations $\rightarrow$ application speed-ups
  - study symmetries in tensor equations and factorizations
  - consider other symmetries and relation to fast matrix multiplication

Backup slides

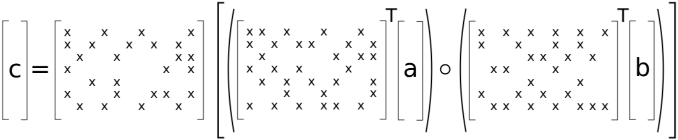
Lower bounds for symmetry preserving algorithms

Bilinear algorithms[1] enable robust communication lower bounds

- 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 derived based on matrix rank[2]

[1] Pan, Springer, 1984

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

Nesting of bilinear algorithms

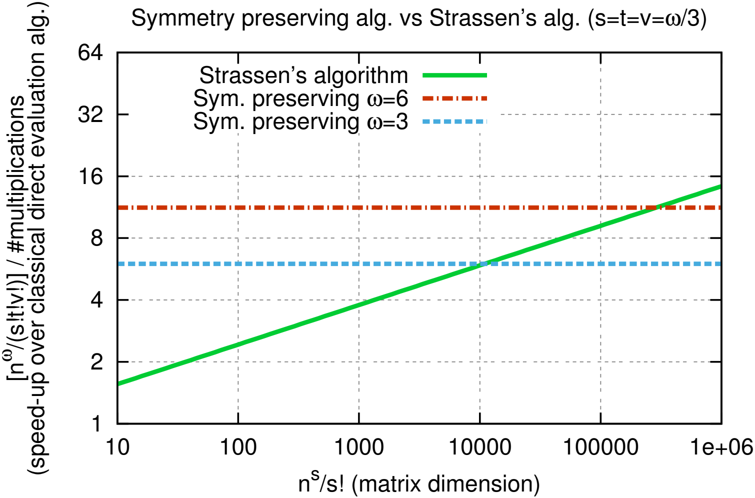
Given two bilinear algorithms:

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

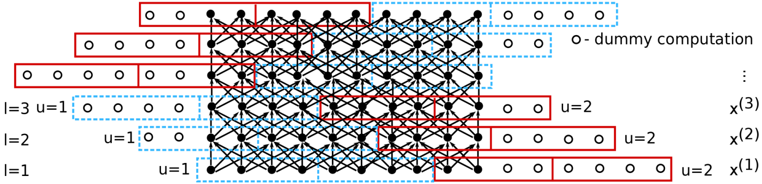
We can nest them by computing their tensor product

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

Symmetry preserving algorithm vs Strassen's algorithm



Block-cyclic algorithm for s-step methods

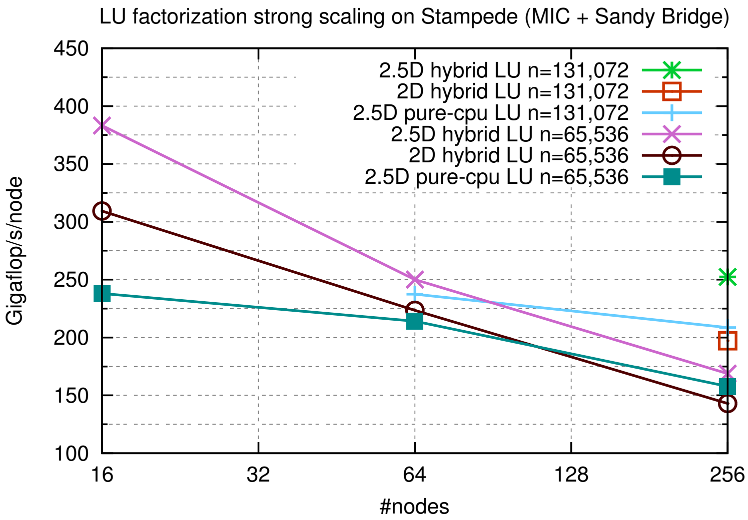


For s-steps of a $(2m+1)^d$-point stencil with block-size of $H^{1/d}/m$,

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

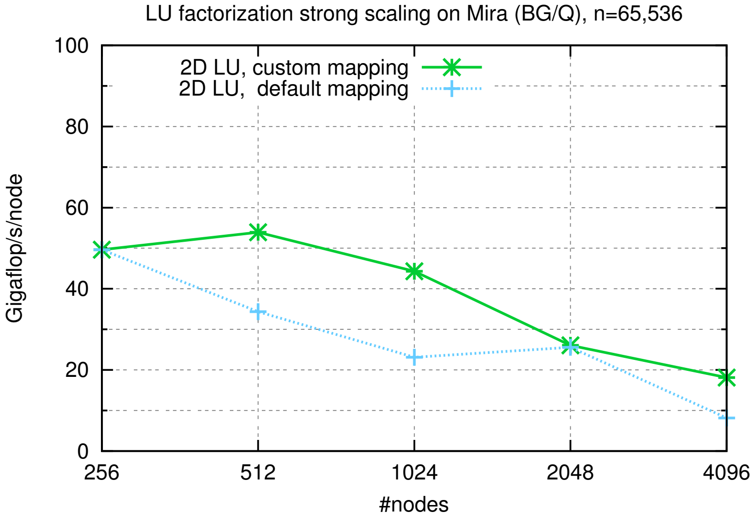
which are good when $H = \Theta(n^d/p)$, so the algorithm is useful when the cache size is a bit smaller than n^d/p

2.5D LU on MIC

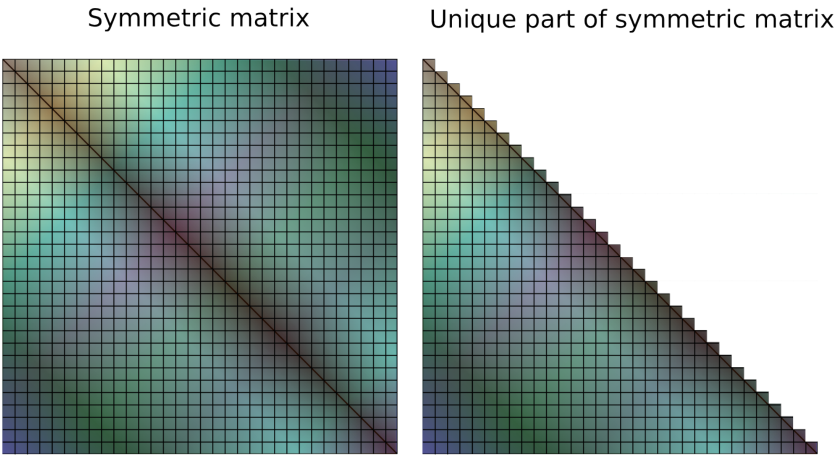


LU factorization strong scaling on Stampede (MIC + Sandy Bridge)
Gigaflop/s/node
#nodes
450
400
350
300
250
200
150
100
16
32
64
128
256
2.5D hybrid LU n=131,072
2D hybrid LU n=131,072
2.5D pure-cpu LU n=131,072
2.5D hybrid LU n=65,536
2D hybrid LU n=65,536
2.5D pure-cpu LU n=65,536

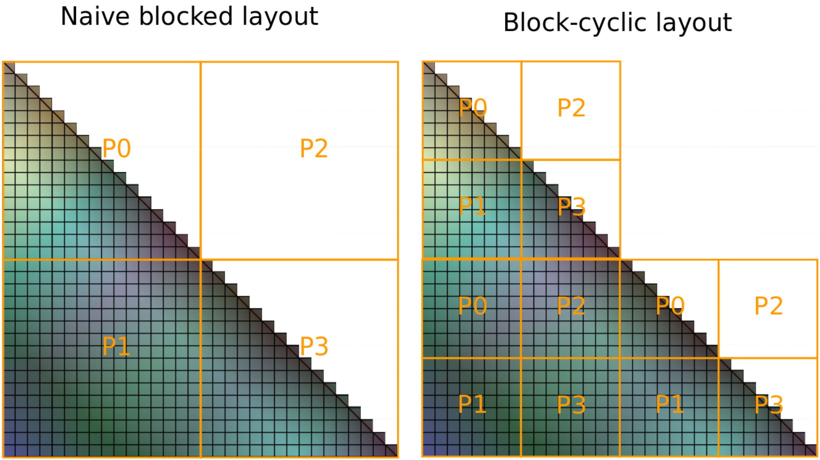
Topology-aware mapping on BG/Q



Symmetric matrix representation

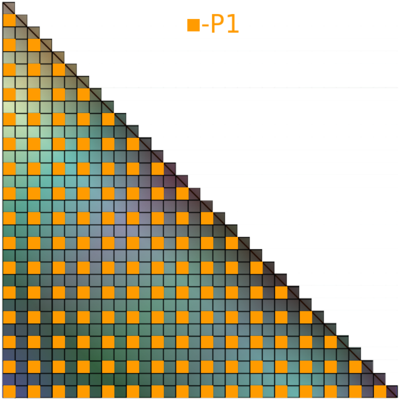


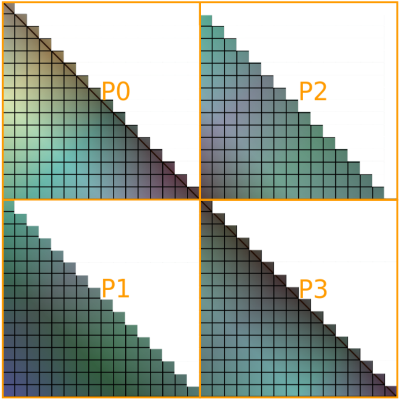
Blocked distributions of a symmetric matrix



Cyclic distribution of a symmetric matrix

Cyclic layout $\sim$ Improved blocked layout





Our CCSD factorization

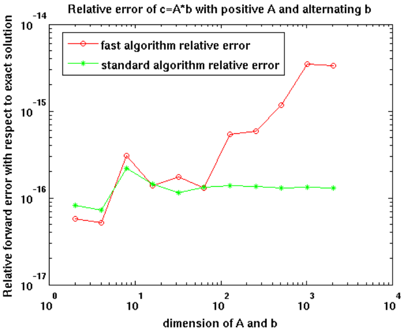
Credit to John F. Stanton and Jurgen Gauss

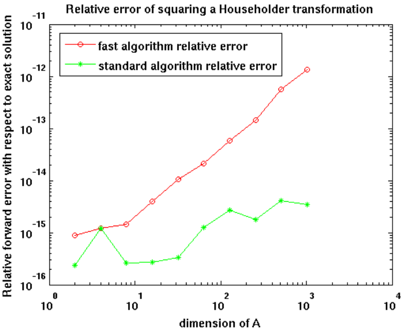
$$
\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\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

Our CCSD factorization

$$
\begin{aligned}
\tilde{W}^{mn}_{ei} &= v^{mn}_{ei} + \sum_f v^{mn}_{ef} t^f_i , \\
\tilde{W}^{mn}_{ij} &= v^{mn}_{ij} + P^i_j \sum_e v^{mn}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{mn}_{ef} \tau^{ef}_{ij} , \\
\tilde{W}^{am}_{ie} &= v^{am}_{ie} - \sum_n \tilde{W}^{mn}_{ei} t^a_n + \sum_f v^{ma}_{ef} t^f_i + \frac{1}{2} \sum_{nf} v^{mn}_{ef} t^{af}_{in} , \\
\tilde{W}^{am}_{ij} &= v^{am}_{ij} + P^i_j \sum_e v^{am}_{ie} t^e_j + \frac{1}{2} \sum_{ef} v^{am}_{ef} \tau^{ef}_{ij} , \\
z^a_i &= f^a_i - \sum_m \tilde{F}^m_i t^a_m + \sum_e f^a_e t^e_i + \sum_{em} v^{ma}_{ei} t^e_m + \sum_{em} v^{ae}_{im} \tilde{F}^m_e + \frac{1}{2} \sum_{efm} v^{am}_{ef} \tau^{ef}_{im} \\
&- \frac{1}{2} \sum_{emn} \tilde{W}^{mn}_{ei} t^{ea}_{mn} , \\
z^{ab}_{ij} &= v^{ab}_{ij} + P^i_j \sum_e v^{ab}_{ie} t^e_j + P^a_b P^i_j \sum_{me} \tilde{W}^{am}_{ie} t^{eb}_{mj} - P^a_b \sum_m \tilde{W}^{am}_{ij} t^b_m \\
&+ P^a_b \sum_e \tilde{F}^a_e t^{eb}_{ij} - P^i_j \sum_m \tilde{F}^m_i t^{ab}_{mj} + \frac{1}{2} \sum_{ef} v^{ab}_{ef} \tau^{ef}_{ij} + \frac{1}{2} \sum_{mn} \tilde{W}^{mn}_{ij} \tau^{ab}_{mn} ,
\end{aligned}
$$

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

Algebraic shortest path computations

All-pairs shortest-paths (APSP):

- distance matrix is the closure of A,

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

- Floyd–Warshall = Gauss–Jordan elimination ≈ Gaussian elimination
  - $O(n^3)$ cost, but contains length $n \log n$ dependency path[1]
- path doubling: $\log n$ steps, $O(n^3 \log n)$ cost:

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

- sparse path doubling[2]:
  - let C be subset of B^k corresponding to paths containing *exactly* k edges,

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

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

[1] S., Buluc, Demmel, IPDPS, 2013

[2] Tiskin, Springer LNCS, 2001

Coupled cluster methods

Coupled cluster provides a systematically improvable approximation to the manybody time-independent Schrödinger equation $H|\Psi\rangle = E|\Psi\rangle$

- the Hamiltonian has one- and two- electron components $H = F + V$
- Hartree-Fock (SCF) computes mean-field Hamiltonian: F, V
- Coupled-cluster methods (CCSD, CCSDT, CCSDTQ) consider transitions of (doubles, triples, and quadruples) of electrons to unoccupied orbitals, encoded by tensor operator, $T = T_1 + T_2 + T_3 + T_4$
- they use an exponential ansatz for the wavefunction, $\Psi = e^T\phi$ where ϕ is a Slater determinant
- expanding $0 = \langle\phi'|H|\Psi\rangle$ yields nonlinear equations for $\{T_i\}$ in F, V

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

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

Symmetry preserving algorithms

By exploiting symmetry, reduce multiplies (but increase adds)[1]

- rank-2 vector outer product

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

- squaring a symmetric matrix A (or $AB + BA$)

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

- for symmetrized contraction of symmetric order $s + v$ and $v + t$ tensors

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

  e.g. cases above are
  - $s = 1, t = 1, v = 0 \rightarrow$ reduction by 2X
  - $s = 1, t = 1, v = 1 \rightarrow$ reduction by 6X

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

Applications of symmetry preserving algorithms

Extensions and applications:

- algorithms generalize to antisymmetric and Hermitian tensors
- cost reductions in partially-symmetric coupled cluster contractions: 2X-9X for select contractions, 1.3X-2.1X for methods
- for Hermitian tensors, multiplies cost 3X more than adds
- $(2/3)n^3$ bilinear rank for squaring a *nonsymmetric* matrix
- decompose symmetric contractions into smaller symmetric contractions

Further directions:

- high performance implementation
- symmetry in tensor equations (e.g. Cholesky factors)
- generalization to other group actions
- relationships to fast matrix multiplication and structured matrices

Communication cost of symmetry preserving algorithms

For contraction of order $s + v$ tensor with order $v + t$ tensor[1]

- symmetry preserving algorithm requires $\frac{(s+v+t)!}{s!v!t!}$ fewer multiplies
- matrix-vector-like algorithms ($\min(s, v, t) = 0$)
  - vertical communication dominated by largest tensor
  - horizontal communication asymptotically greater if only unique elements are stored and $s \neq v \neq t$
- matrix-matrix-like algorithms ($\min(s, v, t) > 0$)
  - vertical and horizontal communication costs asymptotically greater for symmetry preserving algorithm when $s \neq v \neq t$
- further work: bounds for nested and iterative bilinear algorithms

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