Algorithms as Multilinear Tensor Equations

Edgar Solomonik

Department of Computer Science
ETH Zurich

University of Illinois

March 9, 2016

Pervasive paradigms in scientific computing

What commonalities exist in simulation and data analysis applications?

- multidimensional datasets (observations, discretizations)
- higher-order relations: equations, maps, graphs, hypergraphs
- **sparsity** and **symmetry** in structure of relations
- **algebraic descriptions of datasets and relations**

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
- reveal sparsity structure of relations in multidimensional space

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, ...)

What is the power of a parallel tensor library?

The ability to **optimally** orchestrate

- *algebraic transformations*
- *data movement*
- *synchronization*

for a **universal** class of algebraic computations

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
- compression (tensor factorizations, sparsity)

Machine learning

- sparse multi-feature discrete datasets
- 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

Applications to quantum systems

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

Outline and highlights

1. Massively-parallel electronic structure calculations
   - Cyclops Tensor Framework (CTF): first distributed-memory tensor contraction framework
   - codes using CTF for wavefunction methods: Aquarius, QChem, VASP, Psi4
   - coupled cluster faster than NWChem by > 10X, nearly 1 petaflop/s
2. Sparse and discrete tensor computations
   - CTF supports arbitrary sparse multidimensional arrays
   - sparsity used to accelerate algebraic all-pairs shortest-paths
3. Communication-optimal algorithms for linear solvers
   - novel tradeoffs: synchronization vs communication in Cholesky and stencils
   - 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
4. Preserving symmetry in tensor contractions
   - contraction of order $2s$ symmetric tensors in $\frac{(3s)!}{(s!)^3}$ fewer multiplies
   - up to 9X speed-up for partially-symmetric contractions in coupled cluster

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

A library for tensor computations

Cyclops Tensor Framework[1]

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

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

A library for tensor computations

Cyclops Tensor Framework

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

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

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

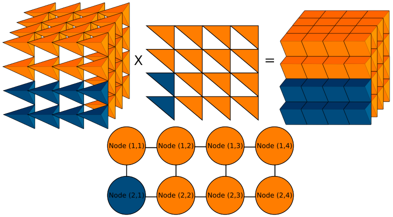
Jacobi iteration (solves $Ax = b$ iteratively) example code snippet

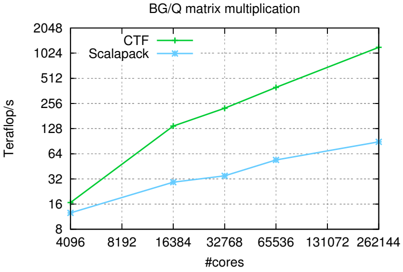
```
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){
  Matrix<> R(A);
  R["ii"] = 0.0;
  Vector<> x(n), d(n), r(n);
  Function<> inv([](double & d){ return 1./d; });
  d["i"] = inv(A["ii"]); // set d to inverse of diagonal of A
  do {
    x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
    r["i"] = b["i"]-A["ij"]*x["j"];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

Performance of CTF for dense computations

CTF is highly tuned for massively-parallel machines

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





CCSD using CTF

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

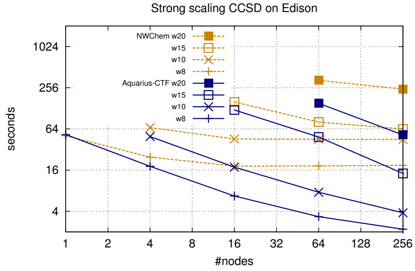
```
FMI["mi"]      += 0.5*WMNEF["mnef"]*T2["efin"];
WMNIJ["mnij"] += 0.5*WMNEF["mnef"]*T2["efij"];
FAE["ae"]      -= 0.5*WMNEF["mnef"]*T2["afmn"];
WAMEI["amei"] -= 0.5*WMNEF["mnef"]*T2["afin"];

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"];
```

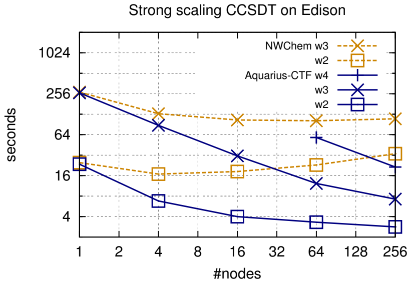
CTF is used within **Aquarius, QChem, VASP, and Psi4**

Comparison with NWChem

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

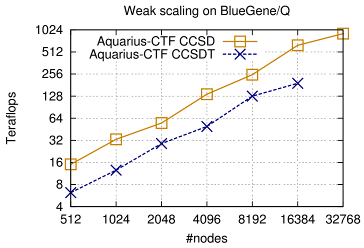
- provides CCSD and CCSDT
- derives equations via Tensor Contraction Engine (TCE)
- generates contractions as blocked loops leveraging Global Arrays

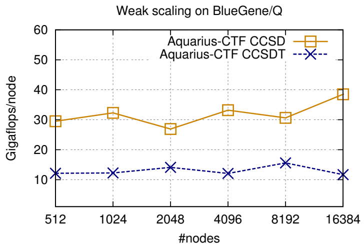


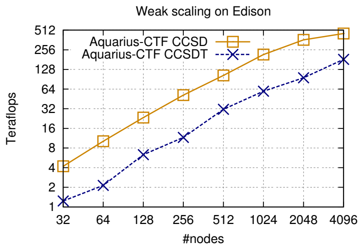


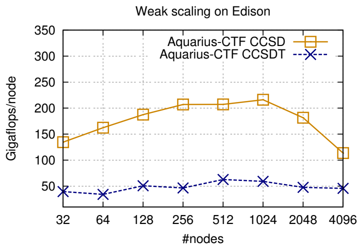
Coupled cluster on IBM BlueGene/Q and Cray XC30

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









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

Sparsity in electronic structure computations

Møller-Plesset perturbation theory (MP3) code snippet

```
Z["abij"] += Fab["af"]*T["fbij"];
Z["abij"] -= Fij["ni"]*T["abnj"];
Z["abij"] += 0.5*Vabcd["abef"]*T["efij"];
Z["abij"] += 0.5*Vijkl["mnij"]*T["abmn"];
Z["abij"] -= Vaibj["amei"]*T["ebmj"];
```

Consider sparse two-electron integrals: **`Vabcd`**, **`Vijkl`**, **`Vaibj`**

Algebraic shortest path computations

An n node graph can be represented by an $n \times n$ adjacency matrix

- a hypergraph with hyperedges of cardinality k, by an order k tensor

Tropical (geodetic) semiring

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

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

Algebraic shortest path computations

Bellman-Ford algorithm (SSSP) for adjacency matrix A:

1. initialize $v^{(1)} = (0, \infty, \infty, \ldots)$
2. compute $v^{(n)}$ via recurrence $\qquad v^{(i+1)} = v^{(i)} \oplus (v^{(i)} \otimes A)$

All-pairs shortest-paths (APSP):

- distance matrix is the closure of A, $\qquad A^* = I \oplus A \oplus A^2 \oplus \ldots A^n$
- Floyd–Warshall = Gauss–Jordan elimination $\approx$ Gaussian elimination
  - $O(n^3)$ cost, but contains length $n \log n$ dependency path
- 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[a]:
  1. let C be subset of B^k corresponding to paths containing *exactly* k edges
  2. $B^{2k} = B^k \oplus (C \otimes B^k)$
  3. $O(n^3)$ cost, dependency paths length $O(\log^2 n)$

[a] Tiskin, Springer LNCS, 2001

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"];
}
```

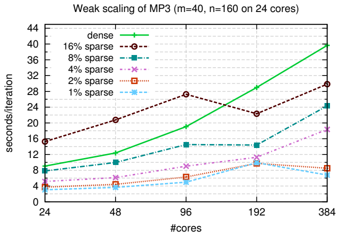
Betweenness centrality

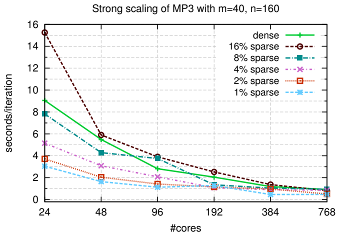
Betweenness centrality code snippet, for *k* of *n* nodes

```
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"]);
  ...
}
```

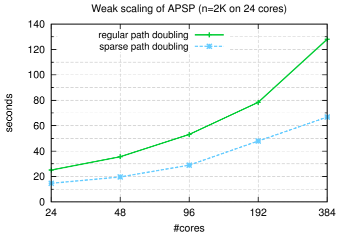
Performance of CTF for sparse computations

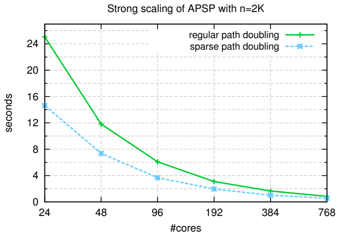
MP3 leveraging sparse-dense tensor contractions[a]





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





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

Beyond computation cost

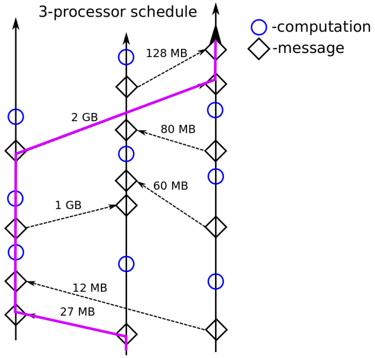
Algorithms should minimize communication, not just computation

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

Cost model for parallel algorithms

Given a schedule of work and communication tasks on p processors, 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[2]

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

- memory-dependent horizontal communication lower bound[3]

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

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

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

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

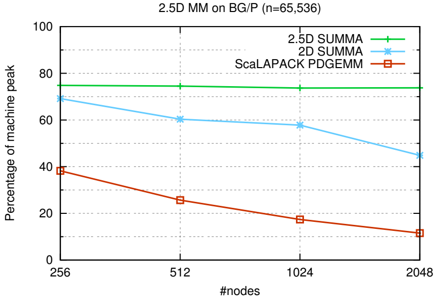
[2] Aggarwal, Chandra, Snir, TCS, 1990

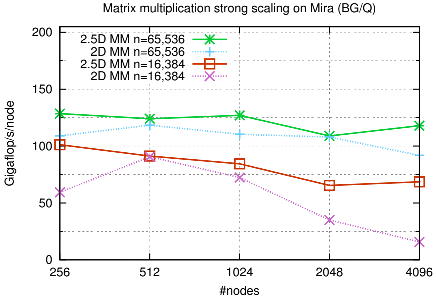
[3] Irony, Toledo, Tiskin, JPDC, 2004

Communication-efficient matrix multiplication

Communication-avoiding algorithms for matrix multiplication have been studied extensively[4]

They continue to be attractive on modern architectures[5]





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

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

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

Synchronization cost lower bounds

Unlike matrix multiplication, many algorithms in numerical linear algebra have polynomial depth (contain a long dependency path)

- matrix multiplication synchronization cost bound[6]

$$S_{\mathrm{MM}} = \Theta\left(\frac{n^3}{pM^{3/2}}\right)$$

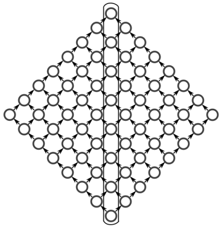
- algorithms for Cholesky, LU, QR, SVD do not attain this bound
- low granularity computation increases synchronization cost

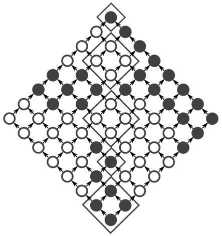
[6] Ballard, Demmel, Holtz, Schwartz, SIAM JMAA, 2011

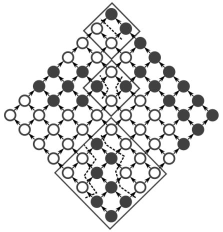
Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the $n \times n$ diamond DAG,[7]

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


Dependency chain P


Monochrome dependency intervals


Multicolored dependency intervals

We generalize this idea[8]

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

[7] Papadimitriou, Ullman, SIAM JC, 1987

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

Tradeoffs involving synchronization

We apply tradeoff lower bounds to dense linear algebra algorithms, represented via dependency hypergraphs:[a]

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

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

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

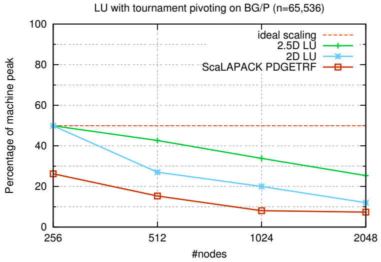
$$F_{\mathrm{CHOL}} \cdot S_{\mathrm{CHOL}}^2 = \Omega\left(n^3\right) \qquad W_{\mathrm{CHOL}} \cdot S_{\mathrm{CHOL}} = \Omega\left(n^2\right)$$

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

Communication-efficient LU factorization

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

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



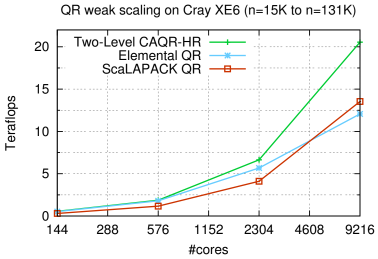
- LU with pairwise pivoting[9] extended to tournament pivoting[10]
- first implementation of a communication-optimal LU algorithm[10]

[9] Tiskin, FGCS, 2007

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

Communication-efficient QR factorization

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



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

[a] Tiskin, FGCS, 2007

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

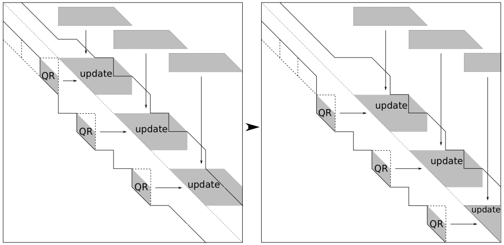
[c] S., UCB, 2014

Communication-efficient eigenvalue computation

For the dense symmetric matrix eigenvalue problem[a]

$$W_{\mathrm{SE}} = O(n^2/\sqrt{cp}), S_{\mathrm{QR}} = O(\sqrt{cp}\log^2 p)$$

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



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

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

Synchronization tradeoffs in stencils

Our lower bound analysis extends to sparse iterative methods:[11]
For computing s applications of a $(2m+1)^d$-point stencil,

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

- 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

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

Exploiting symmetry in tensors

Tensor symmetry (e.g. $A_{ij} = A_{ji}$) reduces memory and cost[12]

- for order d tensor, $d!$ less memory
- dot product $\sum_{i,j} A_{ij}B_{ij} = 2\sum_{i<j} A_{ij}B_{ij} + \sum_i A_{ii}B_{ii}$
- matrix-vector multiplication $(A_{ij} = A_{ji})^1$

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

- $A_{ij}b_j \neq A_{ji}b_i$ but $A_{ij}(b_i + b_j) = A_{ji}(b_j + b_i) \rightarrow (1/2)n^2$ multiplies
- partially-symmetric case: $A^{km}_{ij} = A^{km}_{ji}$

$$c^{kl}_i = \sum_{j,m} A^{km}_{ij} b^{ml}_j = \quad \sum_j \left(\sum_m A^{km}_{ij}(b^{ml}_i + b^{ml}_j) \right) - \sum_m \left(\sum_j A^{km}_{ij} \right) b^{ml}_i$$

- let $Z^{kl}_{ij} = \sum_m A^{km}_{ij}(b^{ml}_i + b^{ml}_j)$ and observe $Z^{kl}_{ij} = Z^{kl}_{ji}$
- Z^{kl}_{ij} can be computed using $(1/2)n^5$ multiplies and $(1/2)n^5$ adds

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

Symmetry preserving algorithms

By exploiting symmetry, reduce multiplies (but increase adds)[13]

- 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

[13] 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
  - Hermitian matrix multiplication and tridiagonal reduction (BLAS and LAPACK routines) with 25% fewer operations
- $(2/3)n^3$ 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

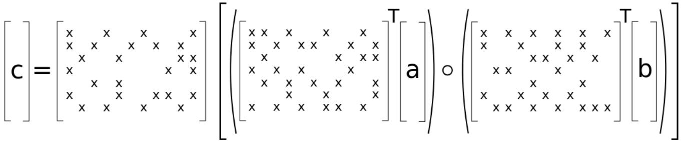
Lower bounds for symmetry preserving algorithms

Bilinear algorithms[14] 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[15]

[14] Pan, Springer, 1984

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

Communication cost of symmetry preserving algorithms

For contraction of order $s + v$ tensor with order $v + t$ tensor[16]

- 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

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

Summary of contributions

Novel results described in this talk:

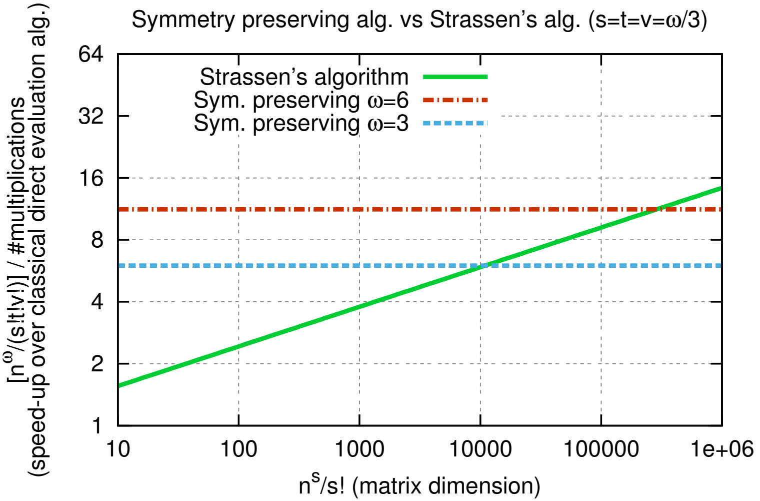
- Cyclops Tensor Framework
  - first fully robust distributed-memory tensor contraction library
  - supports symmetry, sparsity, general algebraic structures
  - coupled cluster performance more than 10X faster than state-of-the-art, reaching 1 petaflop/s performance
- communication-avoiding linear solvers
  - tradeoffs: synchronization vs computation or communication in TRSV, Cholesky, and stencils
  - new algorithms and implementations with up to $p^{1/6}$ less communication for LU, QR, symmetric eigenvalue problem
  - speed-ups of up to 2X for LU and QR over vendor-optimized libraries
- symmetry preserving algorithms
  - reduce number of multiplications in symmetric contractions by $\frac{(s+t+v)!}{s!t!v!}$
  - reduce cost of basic Hermitian matrix operations by 25%
  - reduce cost of some contractions in coupled cluster by 2X in CCSD (1.3X overall), 4X in CCSDT (2.1X overall), 9X in CCSDTQ
  - new algebraic formulation of communication lower bounds

Impact and future work

- Cyclops Tensor Framework
  - already widely-adapted in quantum chemistry, many requests for features
  - study algorithms for tensor expressions $\rightarrow$ factorization, scheduling, ...
  - engage new application domains (via sparsity and algebraic structures)
    - tensor networks for condensed matter-physics, particle methods
    - graph algorithms, discrete data analysis
    - graphics, computer vision, machine learning
- communication-avoiding algorithms
  - existing fast implementations already used by applications (e.g. QBox)
  - find efficient methods of searching larger tuning spaces
  - algorithms for computing eigenvectors, SVD, tensor factorizations
  - study (randomized) algorithms for sparse matrix factorization
- symmetry in tensor computations
  - cost improvements $\rightarrow$ fast library implementations $\rightarrow$ application speed-ups
  - study symmetries in tensor equations and factorizations
  - consider other symmetries and relation to fast matrix multiplication

Backup slides

Symmetry preserving algorithm vs Strassen's algorithm



Nesting of bilinear algorithms

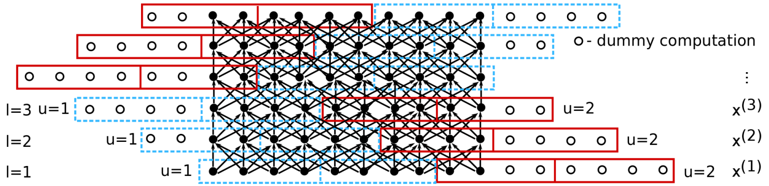
Given two bilinear algorithms:

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

We can nest them by computing their tensor product

$$\begin{aligned}\Lambda_1 \otimes \Lambda_2 :=& (\mathbf{F_1^{(A)}} \otimes \mathbf{F_2^{(A)}}, \mathbf{F_1^{(B)}} \otimes \mathbf{F_2^{(B)}}, \mathbf{F_1^{(C)}} \otimes \mathbf{F_2^{(C)}})\\ \text{rank}(\Lambda_1 \otimes \Lambda_2) =& \text{rank}(\Lambda_1) \cdot \text{rank}(\Lambda_2)\end{aligned}$$

Block-cyclic algorithm for s-step methods

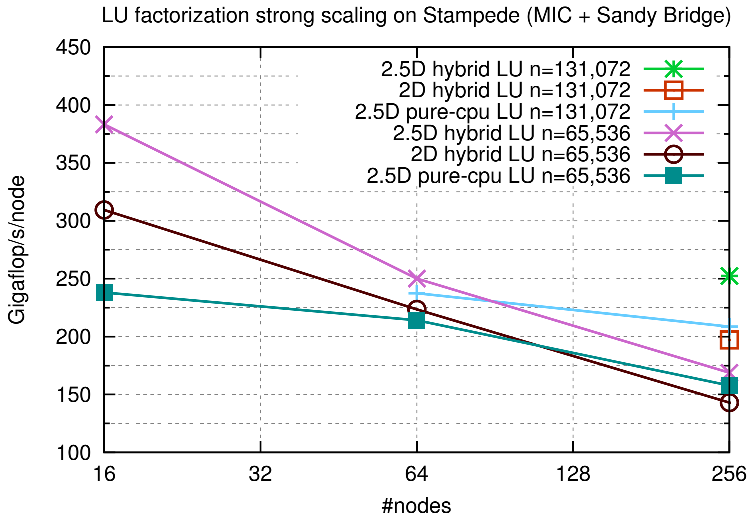


For s-steps of a $(2m+1)^d$-point stencil with block-size of $H^{1/d}/m$,

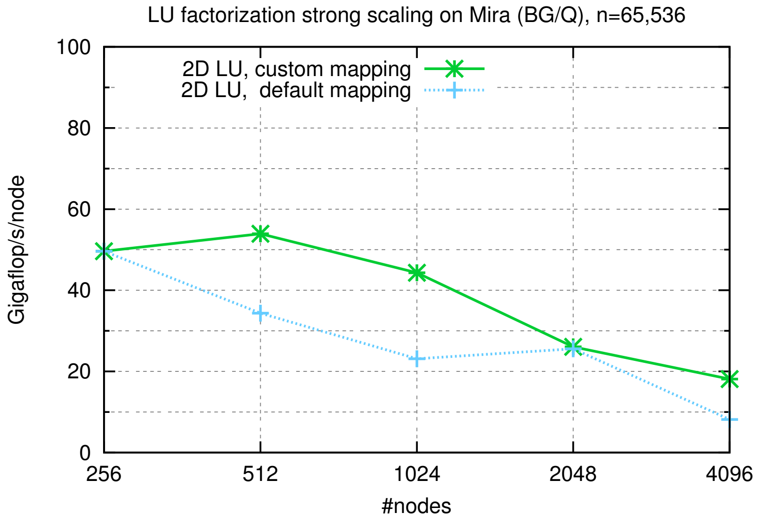
$$W_{\mathrm{Kr}} = O\left(\frac{msn^d}{H^{1/d}p}\right) \quad S_{\mathrm{Kr}} = O(sn^d/(pH)) \quad Q_{\mathrm{Kr}} = O\left(\frac{msn^d}{H^{1/d}p}\right)$$

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

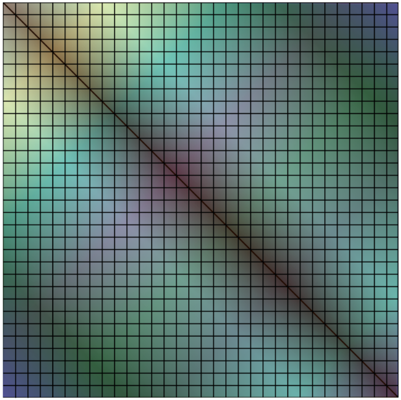

LU factorization strong scaling on Stampede (MIC + Sandy Bridge)
2.5D hybrid LU n=131,072
2D hybrid LU n=131,072
2.5D pure-cpu LU n=131,072
2.5D hybrid LU n=65,536
2D hybrid LU n=65,536
2.5D pure-cpu LU n=65,536
Gigaflop/s/node
450
400
350
300
250
200
150
100
16
32
64
128
256
#nodes

Topology-aware mapping on BG/Q

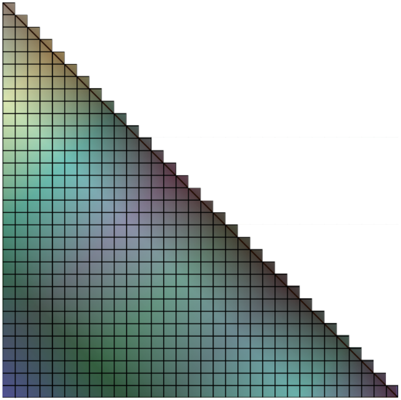


Symmetric matrix representation

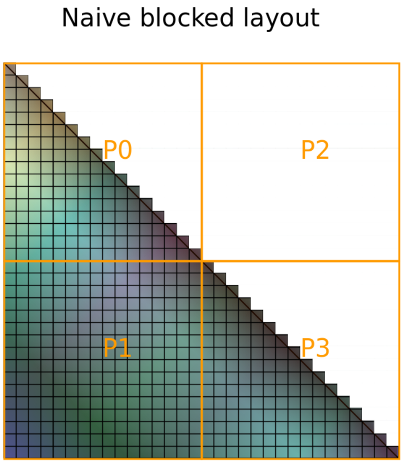
Symmetric matrix

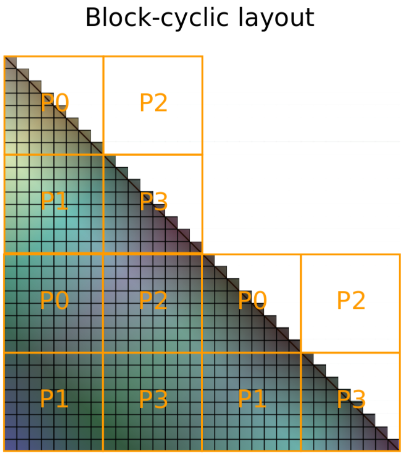


Unique part of symmetric matrix



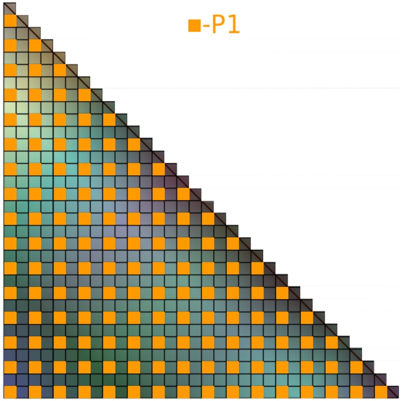
Blocked distributions of a symmetric matrix

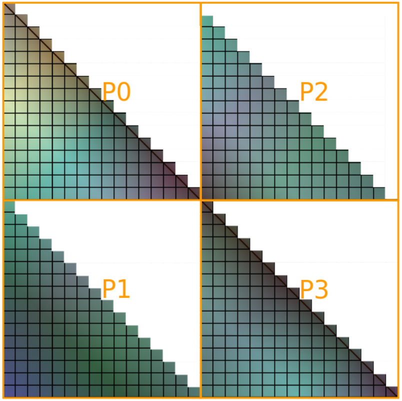




Cyclic distribution of a symmetric matrix

Cyclic layout ~ Improved blocked layout





Our CCSD factorization

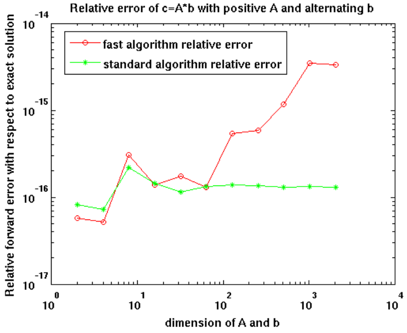
Credit to John F. Stanton and Jurgen Gauss

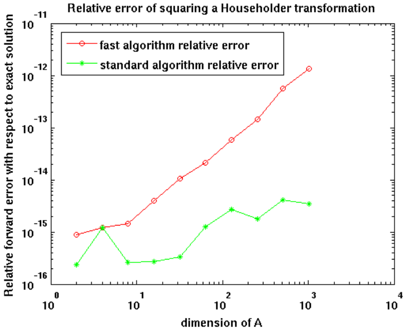
$$
\begin{aligned}
\tau_{ij}^{ab} &= t_{ij}^{ab} + \frac{1}{2} P_b^a P_j^i t_i^a t_j^b, \\
\tilde{F}_e^m &= f_e^m + \sum_{fn} v_{ef}^{mn} t_n^f, \\
\tilde{F}_e^a &= (1 - \delta_{ae}) f_e^a - \sum_m \tilde{F}_e^m t_m^a - \frac{1}{2} \sum_{mnf} v_{ef}^{mn} t_{mn}^{af} + \sum_{fn} v_{ef}^{an} t_n^f, \\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

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

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.