Algorithms as multilinear tensor equations

Edgar Solomonik

Department of Computer Science
ETH Zurich

Technische Universität München

18.1.2016

Tensors and algebraic structures

We consider the expression of data as indexable collections of elements and algorithms as applications of algebraic operators.

Definition (Algebraic structure)

A set of elements (type), potentially equipped with operators and identities

Examples: set, monoid, group, semiring, ring

Definition (Tensor)

A collection of elements of a single type, $\mathbf{T}$, with some order k and dimensions $(n_1, \ldots, n_k)$, with elements $T_{i_1 \ldots i_k}$

Examples: scalar, vector, matrix

An algebraic structure defines summation and contraction of tensors.

Numerical tensor computations

Classical matrix-based computations over the $(+, \cdot)$ ring

- stencil computations (iterative methods for sparse linear systems)

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

- dense matrix factorizations (direct solvers for dense linear systems)

$$\mathbf{A} \approx \mathbf{LU} \qquad \mathbf{A} \approx \mathbf{QR} \qquad \mathbf{A} \approx \mathbf{UDV}^{\mathsf{T}}$$

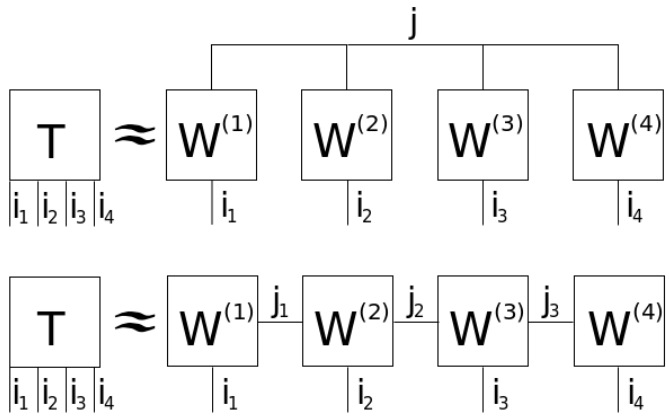
- tensor contractions (peturbation theory, solvers for nonlinear systems)

$$\sum_f F_f^a T_{ij}^{fb} - \sum_n F_i^n T_{nj}^{ab} + \frac{1}{2}\sum_{e,f} V_{ef}^{ab} T_{ij}^{ef} + \frac{1}{2}\sum_{m,n} V_{ij}^{mn} T_{mn}^{ab} - \sum_{e,m} V_{ei}^{am} T_{mj}^{eb}$$

- tensor decompositions (compression)

$$T_{i_1 \ldots i_k} \approx \sum_j W_{i_1 j}^{(1)} \cdots W_{i_k j}^{(k)}$$

$$T_{i_1 \ldots i_k} \approx \sum_{j_1 \ldots j_{k-1}} W_{i_1 j_1}^{(1)} W_{j_1 i_2 j_2}^{(2)} \cdots W_{j_{k-2} i_{k-1} j_{k-1}}^{(k-1)} W_{j_{k-1} i_k}^{(k)}$$



Discrete tensor algorithms

Alternative algebraic structures expand potential of tensor computations

- graph algorithms via tropical (geodetic) semiring $(\min, +)$
  - single-source shortest-paths via Bellman-Ford (stencil-like)
  - all-pairs shortest-paths (APSP) via Floyd-Warshall (LU-like)
  - APSP via path doubling (matrix-multiplication-like)
  - betweenness centrality
  - hypergraphs are representable by tensors
- recursion via higher order tensors
  - prefix sum, scan
  - FFT or other butterfly networks
  - bitonic sort

Cost model for parallel algorithms

Given a schedule that specifies all work and communication tasks on p processors, we consider the following costs, measured along dependent sequences of tasks (as in $\alpha - \beta$, BSP, and LogGP models).

Definition (F – computation cost)

Number of operations performed

Definition (Q – vertical communication cost)

Amount of data moved between memory and cache

Definition (W – horizontal communication cost)

Amount of data moved between processors

Definition (S – synchronization cost)

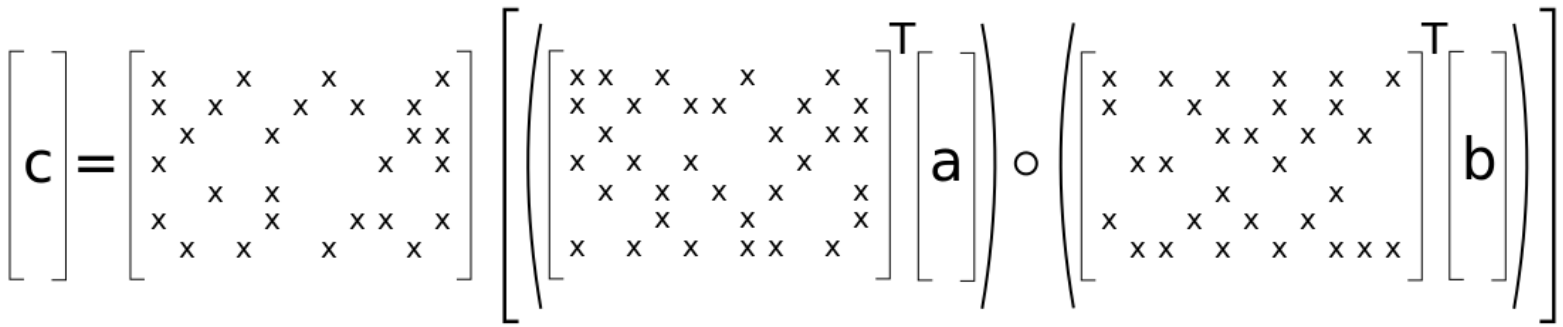
Number of distinct messages sent between processors

Bilinear algorithms

A bilinear algorithm Λ is defined by three matrices, $\Lambda = (\mathbf{F^{(A)}}, \mathbf{F^{(B)}}, \mathbf{F^{(C)}})$
Given input vectors $\mathbf{a}$ and $\mathbf{b}$, it computes vector,

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

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

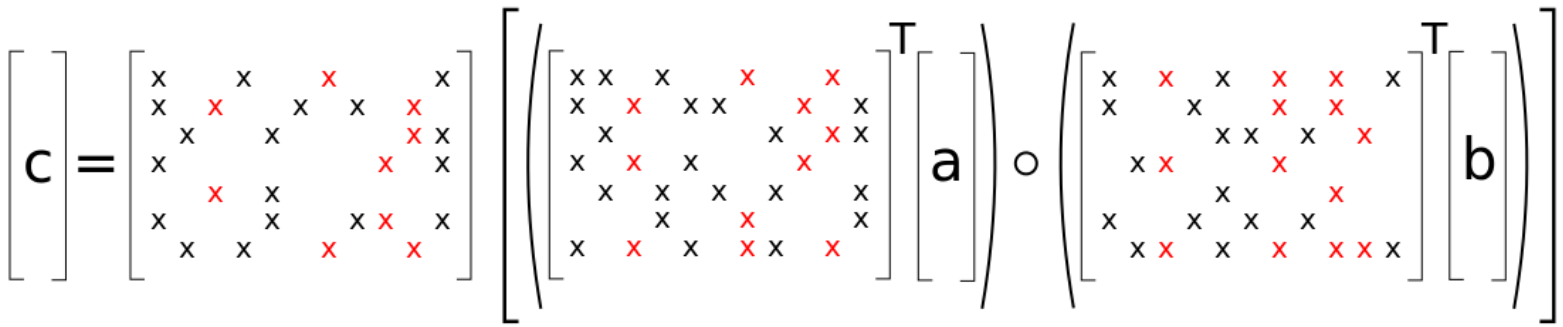


- the number of columns in the three matrices is equal and is the *bilinear algorithm rank*, denoted rank(Λ)
- the number of rows in each matrix corresponds to the number of inputs (dimensions of $\mathbf{a}$ and $\mathbf{b}$) and outputs (dimension of $\mathbf{c}$)

Bilinear algorithm expansion

A bilinear algorithm $\Lambda = (\mathbf{F}^{(\mathbf{A})}, \mathbf{F}^{(\mathbf{B})}, \mathbf{F}^{(\mathbf{C})})$ has expansion bound $\mathcal{E}_\Lambda : \mathbb{N}^3 \to \mathbb{N}$, if for all projection matrices $\mathbf{P}$,

$$\Lambda_{\text{sub}} = (\mathbf{F}^{(\mathbf{A})}\mathbf{P}, \mathbf{F}^{(\mathbf{B})}\mathbf{P}, \mathbf{F}^{(\mathbf{C})}\mathbf{P})$$



has rank bounded by $\mathcal{E}_\Lambda$,

$$\text{rank}(\Lambda_{\text{sub}}) \leq \mathcal{E}_\Lambda \left(\text{rank}(\mathbf{F}^{(\mathbf{A})}\mathbf{P}), \text{rank}(\mathbf{F}^{(\mathbf{B})}\mathbf{P}), \text{rank}(\mathbf{F}^{(\mathbf{C})}\mathbf{P})\right)$$

Communication lower bounds

Consider any algorithm $\Lambda = (\mathbf{F^{(A)}}, \mathbf{F^{(B)}}, \mathbf{F^{(C)}})$ and expansion bound $\mathcal{E}_\Lambda$.
For a cache size H, Λ requires total vertical communication cost,

$$Q \geq \left\lceil 2H \frac{\text{rank}(\Lambda)}{\mathcal{E}_\Lambda^{\max}(H)} \right\rceil$$

where $\mathcal{E}_\Lambda^{\max}(H) := \max_{c^{(A)}+c^{(B)}+c^{(C)}=3H} \mathcal{E}_\Lambda(c^{(A)}, c^{(B)}, c^{(C)})$.
Given p processors, Λ requires horizontal communication cost,

$$W \geq \min_{\mathcal{E}_\Lambda\left(c^{(A)}+\frac{r^{(A)}}{p}, c^{(B)}+\frac{r^{(B)}}{p}, c^{(B)}+\frac{r^{(C)}}{p}\right) \geq \frac{\text{rank}(\Lambda)}{p}} \left[c^{(A)} + c^{(B)} + c^{(C)} \right]$$

where $r^{(A)}$, $r^{(B)}$, and $r^{(C)}$ are the number of rows in $\mathbf{F^{(A)}}$, $\mathbf{F^{(B)}}$, and $\mathbf{F^{(C)}}$, respectively.

Dependency interval expansion

Consider a bilinear algorithm that computes a set of multiplications V with a partial ordering, we denote a dependency interval between $a, b \in V$ as

$$[a, b] = \{a, b\} \cup \{c : a < c < b, c \in V\}$$

If there exists $\{v_1, \ldots, v_n\} \in V$ with $v_i < v_{i+1}$ and $\left|[v_{i+1}, v_{i+k}]\right| = \Theta(k^d)$ for all $k \in \mathbb{N}$, then

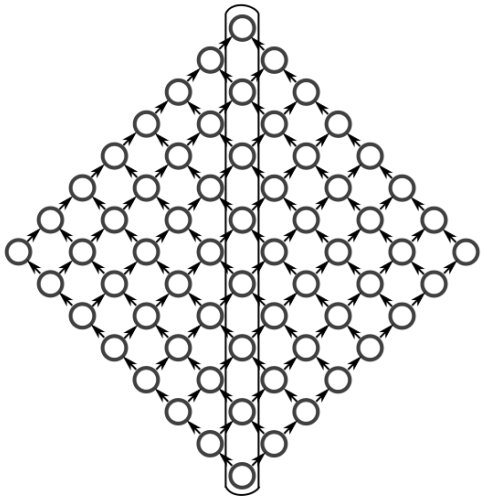
$$F \cdot S^{d-1} = \Omega(n^d)$$

where F is the computation cost and S is the synchronization cost

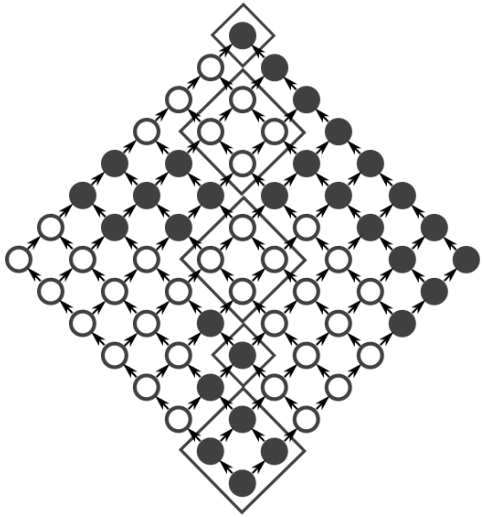
Further, if the algorithm has bilinear expansion $\mathcal{E}$, satisfying $\mathcal{E}^{\max}(H) = \Omega(H^{\frac{d}{d-1}})$, then

$$W \cdot S^{d-2} = \Omega(n^{d-1})$$

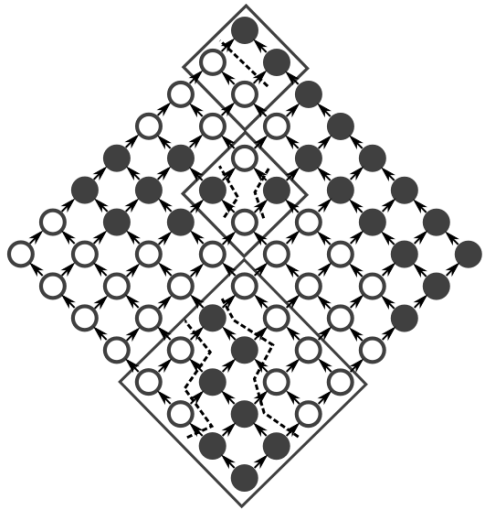
Example: diamond DAG



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

For the $n \times n$ diamond DAG ($d = 2$),

$$F \cdot S^{2-1} = F \cdot S = \Omega((n/b)b^2) \cdot \Omega(n/b) = \Omega(n^2)$$
$$W \cdot S^{2-2} = W \quad = \Omega((n/b)b) \qquad\qquad = \Omega(n)$$

idea goes back to Papadimitriou and Ullman, 1987

Tradeoffs involving synchronization

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

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

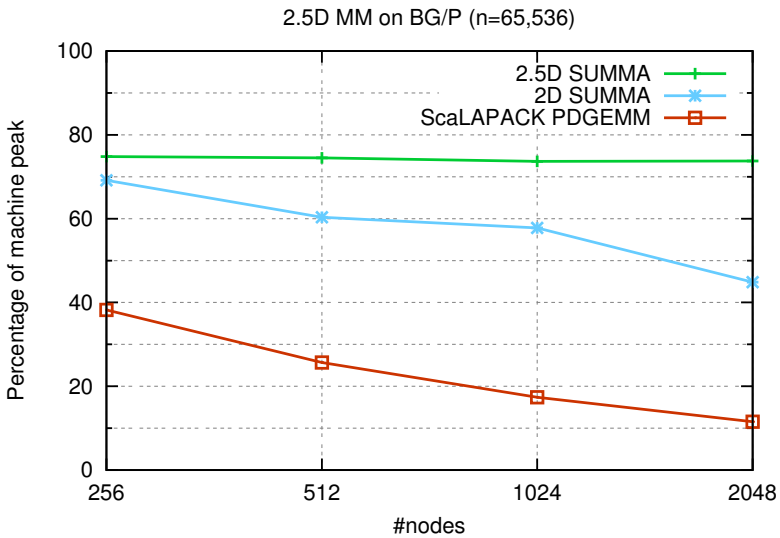
For computing s applications of a $(2m+1)^d$-point stencil

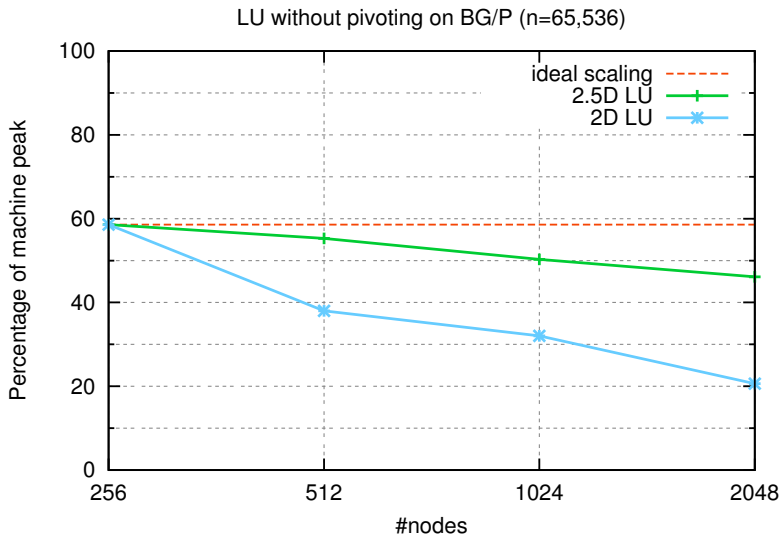
$$F_{\mathrm{St}} \cdot S_{\mathrm{St}}^d = \Omega\left(m^{2d} \cdot s^{d+1}\right) \qquad W_{\mathrm{St}} \cdot S_{\mathrm{St}}^{d-1} = \Omega\left(m^d \cdot s^d\right)$$

Communication-optimal dense matrix algorithms

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

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





- LU with pairwise pivoting extended to tournament pivoting
- QR with Givens rotations extended to Householder transformations
- full-to-banded reduction for symmetric eigenvalue problem
- successive band reduction for symmetric eigenvalue problem

Communication-efficient sparse matrix computations

Iterative stencil computations

- previous work: in-time blocking
  - lowers synchronization cost
  - lowers vertical communication cost
  - increases horizontal communication cost when mesh at least 2D
- new 'cyclic' algorithm, in-time blocks executed bulk synchronously
  - lowers vertical communication cost
  - maintains minimal horizontal communication cost
  - increases synchronization cost
- alternatives are both optimal in different lower bound regimes

Multiplication of a sparse matrix by a dense matrix

- key primitive with many applications
  - iterative solvers
  - tensor computations (MP3 or coupled cluster with localized orbitals)
  - graph algorithms (Bellman-Ford, APSP, betweenness centrality)
- communication-efficient 3D algorithms and lower bound analysis

Exploiting symmetry in tensors

Coupled cluster methods for electronic structure calculations

- approximates electronic wavefunction using $2r$-order tensor representing r-electron correlation
- systematically improvable, CCSD, CCSDT, CCSDTQ ($r = 1, 2, 3$)
- cost dominated by contractions of partially antisymmetric tensors

Exploiting tensor (anti)symmetry

- saves storage and provides easy reduction in cost if set of contraction multiplications is symmetric
- new symmetry-preserving algorithm uses algebraic reorganization to reduce cost (lowers bilinear algorithm rank) when set of contraction multiplications breaks tensor symmetry

$$W_{ikj} = A_{ik} \cdot B_{kj} \quad \rightarrow \quad Z_{ikj} = (A_{ik} + A_{kj} + A_{ji}) \cdot (B_{ik} + B_{kj} + B_{ji})$$

- nested use reduces cost of CCSD by about 1.3, CCSDT by about 2.1

Tensor algebra as a programming abstraction

Cyclops Tensor Framework

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

Tensor algebra as a programming abstraction

Cyclops Tensor Framework

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

Jacobi iteration example code snippet

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

Tensor algebra as a programming abstraction

Cyclops Tensor Framework

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

Møller-Plesset perturbation theory (MP3) code snippet

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

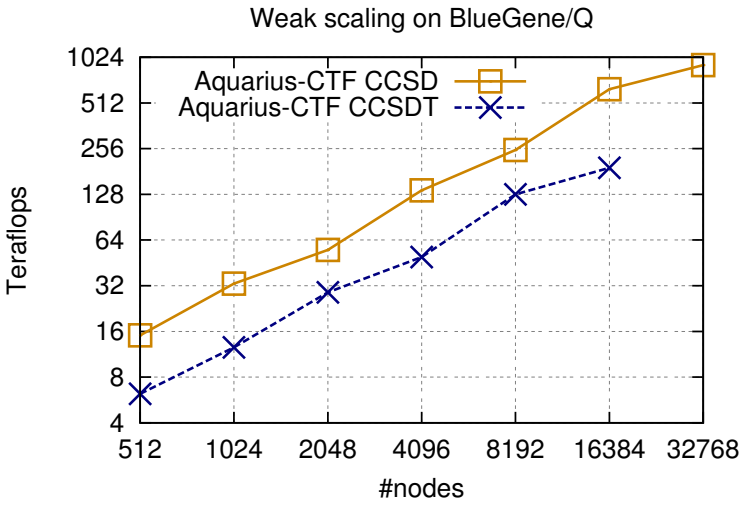
Betweenness centrality

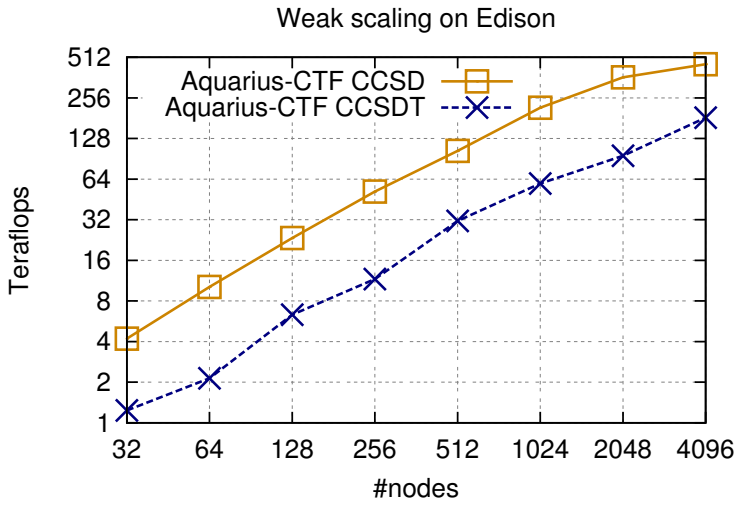
Betweenness centrality code snippet, for k of n nodes

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

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

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

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

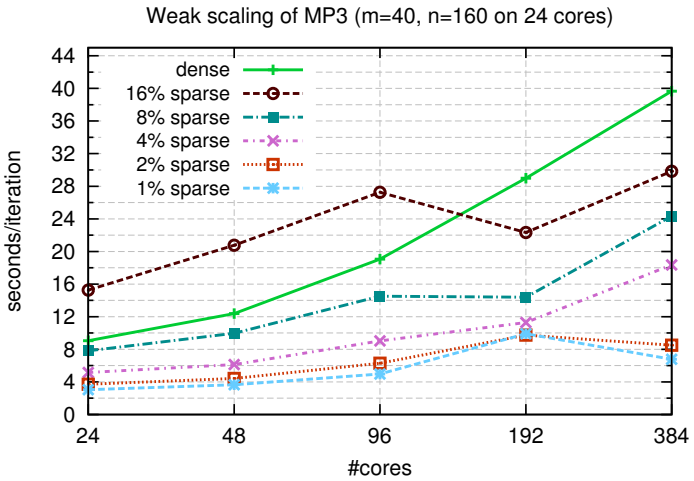
Performance highlights

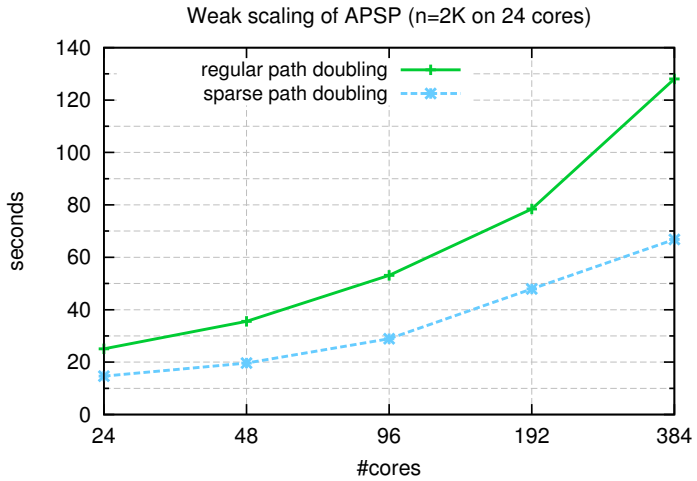
Coupled cluster calculations using dense tensors





MP3 and all-pairs shortest-paths using sparse tensors



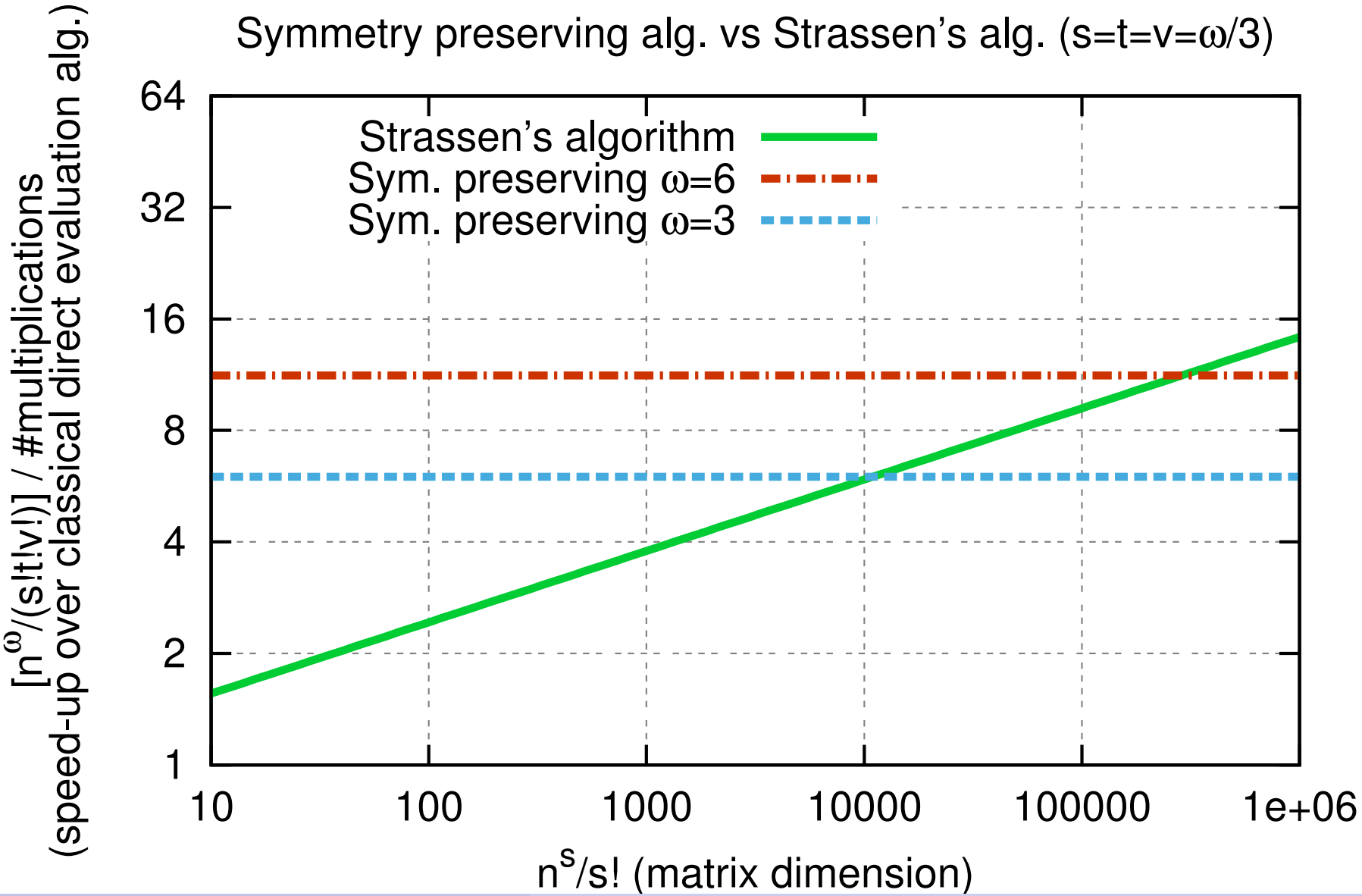


Future work

- further work sparse and symmetric tensor computations
  - bridging the gap between abstractions and application performance
  - bilinear algorithm complexity – fast matrix multiplication
- tensor decompositions
  - communication-efficient parallel algorithms and lower bounds
  - symmetry-preserving tensor decomposition algorithms
  - programming abstractions for dense and sparse tensors
- sets of tensor operations
  - most algorithms correspond to multiple dependent tensors operations
  - communication cost analysis for sets of contractions
  - scheduling, blocking, and decomposition of multiple tensor operations
  - higher-level programming abstractions
- application-driven development
  - tensor decompositions, sparsity, symmetry all motivated by electronic structure applications
  - optimization of primitives serves as feedback loop for development of new electronic structure methods

Backup slides

Symmetry preserving algorithm vs Strassen's algorithm



Nesting of bilinear algorithms

Given two bilinear algorithms:

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

We can nest them by computing their tensor product

$$\Lambda_1 \otimes \Lambda_2 := (\mathbf{F_1^{(A)}} \otimes \mathbf{F_2^{(A)}}, \mathbf{F_1^{(B)}} \otimes \mathbf{F_2^{(B)}}, \mathbf{F_1^{(C)}} \otimes \mathbf{F_2^{(C)}})$$
$$\mathsf{rank}(\Lambda_1 \otimes \Lambda_2) = \mathsf{rank}(\Lambda_1) \cdot \mathsf{rank}(\Lambda_2)$$

Comparison with NWChem

NWChem is a commonly-used distributed-memory quantum chemistry method suite

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