Strassen-like algorithms for symmetric tensor contractions

Edgar Solomonik

Theory Seminar
University of Illinois at Urbana-Champaign

September 18, 2017

Outline

Terminology

A tensor $\boldsymbol{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ has

- order d (i.e. d modes / indices)
- dimensions n_1-by-$\cdots$-by-n_d (in this talk, usually each $n_i = n$)
- elements $\boldsymbol{T}_{\boldsymbol{i}_1 \ldots \boldsymbol{i}_d} = \boldsymbol{T}_{\boldsymbol{i}}$ where $\boldsymbol{i} \in \{1, \ldots, n\}^d$

We say a tensor is symmetric if for any $j, k \in \{1, \ldots, n\}$

$$\boldsymbol{T}_{\boldsymbol{i}_1 \ldots \boldsymbol{i}_j \ldots \boldsymbol{i}_k \ldots \boldsymbol{i}_d} = \boldsymbol{T}_{\boldsymbol{i}_1 \ldots \boldsymbol{i}_k \ldots \boldsymbol{i}_j \ldots \boldsymbol{i}_d}$$

A tensor is antisymmetric (skew-symmetric) if for any $j, k \in \{1, \ldots, n\}$

$$\boldsymbol{T}_{\boldsymbol{i}_1 \ldots \boldsymbol{i}_j \ldots \boldsymbol{i}_k \ldots \boldsymbol{i}_d} = (-1)\boldsymbol{T}_{\boldsymbol{i}_1 \ldots \boldsymbol{i}_k \ldots \boldsymbol{i}_j \ldots \boldsymbol{i}_d}$$

A tensor is partially-symmetric if such index interchanges are restricted to be within subsets of $\{1, \ldots, n\}$, e.g.

$$\boldsymbol{T}^{ij}_{kl} = \boldsymbol{T}^{ji}_{kl} = \boldsymbol{T}^{ji}_{lk} = \boldsymbol{T}^{ij}_{lk}$$

Tensor contractions

We work with contractions of tensors

- $\boldsymbol{A}$ of order $s+v$, and
- $\boldsymbol{B}$ of order $v+t$ into
- $\boldsymbol{C}$ of order $s+t$, defined as

$$\boldsymbol{C_{ij}} = \sum_{\boldsymbol{k}\in\{1,\ldots,n\}^v} \boldsymbol{A_{ik}} \boldsymbol{B_{kj}}$$

- requires $O(n^{s+t+v})$ multiplications and additions, $\omega := s+t+v$
- assumes an index ordering, but does not lose generality
- works with any symmetries of $\boldsymbol{A}$ and $\boldsymbol{B}$
- is extensible to symmetries of $\boldsymbol{C}$ via symmetrization (sum all permutations of modes in $\boldsymbol{C}$, denoted $[\boldsymbol{C}]_{\boldsymbol{ij}}$)
- generalizes simple matrix operations, e.g.

$$\underbrace{(s,t,v) = (1,0,1)}_{\text{matrix-vector product}}, \quad \underbrace{(s,t,v) = (1,1,0)}_{\text{vector outer product}}, \quad \underbrace{(s,t,v) = (1,1,1)}_{\text{matrix-matrix product}}$$

Applications of symmetric tensor contractions

Symmetric and Hermitian matrix operations are part of the BLAS

- matrix-vector products: `symv` (`symm`), `hemv`, (`hemm`)
- symmetrized outer product: `syr2` (`syr2k`), `her2`, (`her2k`)
- these operations dominate symmetric/Hermitian diagonalization

Hankel matrices are order $2\log_2(n)$ partially-symmetric tensors

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_{11} & \boldsymbol{H}_{21}^{\mathsf{T}} \\ \boldsymbol{H}_{21} & \boldsymbol{H}_{22} \end{bmatrix}$$

where $\boldsymbol{H}_{11}, \boldsymbol{H}_{21}, \boldsymbol{H}_{22}$ are also Hankel.

In general, partially-symmetric tensors are nested symmetric tensors

- a nonsymmetric matrix is a vector of vectors
- $\boldsymbol{T}^{ij}_{kl} = \boldsymbol{T}^{ji}_{kl} = \boldsymbol{T}^{ji}_{lk} = \boldsymbol{T}^{ij}_{lk}$ is a symmetric matrix of symmetric matrices

Applications of partially-symmetric tensor contractions

High-accuracy methods in computational quantum chemistry

- solve the multi-electron Schrödinger equation $\boldsymbol{H}|\boldsymbol{\Psi}\rangle = E|\boldsymbol{\Psi}\rangle$, where $\boldsymbol{H}$ is a linear operator, but $\boldsymbol{\Psi}$ is a function of *all* electrons
- use wavefunction ansatze like $\boldsymbol{\Psi} \approx \boldsymbol{\Psi}^{(k)} = e^{\boldsymbol{T}^{(k)}}|\boldsymbol{\Psi}^{(k-1)}\rangle$ where $\boldsymbol{\Psi}^{(0)}$ is a mean-field (averaged) function and $\boldsymbol{T}^{(k)}$ is an order $2k$ tensor, acting as a multilinear excitation operator on the electrons
- coupled-cluster methods use the above ansatze for $k \in \{2, 3, 4\}$ (CCSD, CCSDT, CCSDTQ)
- solve iteratively for $\boldsymbol{T}^{(k)}$, where each iteration has cost $O(n^{2k+2})$, dominated by contractions of partially antisymmetric tensors
- for example, a dominant contraction in CCSD ($k = 2$) is

$$\boldsymbol{Z}^{a\bar{k}}_{i\bar{c}} = \sum_{b=1}^{n} \sum_{j=1}^{n} \boldsymbol{T}^{ab}_{ij} \cdot \boldsymbol{V}^{j\bar{k}}_{b\bar{c}}$$

  where $\boldsymbol{T}^{ab}_{ij} = -\boldsymbol{T}^{ba}_{ij} = \boldsymbol{T}^{ba}_{ji} = -\boldsymbol{T}^{ab}_{ji}$. We'll show an algorithm that requires n^6 rather than $2n^6$ operations.

Fast algorithms

Strassen's algorithm

$$\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$$

$$\begin{aligned}
\boldsymbol{M}_1 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\
\boldsymbol{M}_2 &= (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \cdot \boldsymbol{B}_{11} \\
\boldsymbol{M}_3 &= \boldsymbol{A}_{11} \cdot (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\
\boldsymbol{M}_4 &= \boldsymbol{A}_{22} \cdot (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\
\boldsymbol{M}_5 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \cdot \boldsymbol{B}_{22} \\
\boldsymbol{M}_6 &= (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\
\boldsymbol{M}_7 &= (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{21} + \boldsymbol{B}_{22})
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{C}_{11} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\
\boldsymbol{C}_{21} &= \boldsymbol{M}_2 + \boldsymbol{M}_4 \\
\boldsymbol{C}_{12} &= \boldsymbol{M}_3 + \boldsymbol{M}_5 \\
\boldsymbol{C}_{22} &= \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6
\end{aligned}$$

By minimizing number of products, minimize number of recursive calls

$$T(n) = 7T(n/2) + O(n^2) = O(7^{\log_2 n}) = O(n^{\log_2 7})$$

For convolution, DFT matrix reduces from naive $O(n^2)$ products to $O(n)$, both of these are bilinear algorithms

How can we formally define an algorithm?

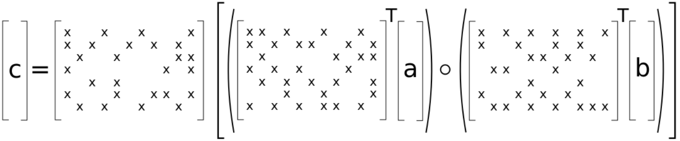
Formally defining a space of algorithms enables systematic exploration.

Definition (Bilinear algorithms (V. Pan, 1984))

A bilinear algorithm $\Lambda = (\boldsymbol{F}^{(A)}, \boldsymbol{F}^{(B)}, \boldsymbol{F}^{(C)})$ computes

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

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are inputs and $\circ$ is the Hadamard (pointwise) product.



Bilinear algorithms as tensor factorizations

A bilinear algorithm corresponds to a CP tensor decomposition

$$\begin{aligned}
\boldsymbol{c}_i &= \sum_{r=1}^{R} \boldsymbol{F}^{(\boldsymbol{C})}_{ir} \left(\sum_j \boldsymbol{F}^{(\boldsymbol{A})}_{jr} \boldsymbol{a}_j \right) \left(\sum_k \boldsymbol{F}^{(\boldsymbol{B})}_{kr} \boldsymbol{b}_k \right) \\
&= \sum_j \sum_k \left(\sum_{r=1}^{R} \boldsymbol{F}^{(\boldsymbol{C})}_{ir} \boldsymbol{F}^{(\boldsymbol{A})}_{jr} \boldsymbol{F}^{(\boldsymbol{B})}_{kr} \right) \boldsymbol{a}_j \boldsymbol{b}_k \\
&= \sum_j \sum_k \boldsymbol{T}_{ijk} \boldsymbol{a}_j \boldsymbol{b}_k \quad \text{where} \quad \boldsymbol{T}_{ijk} = \sum_{r=1}^{R} \boldsymbol{F}^{(\boldsymbol{C})}_{ir} \boldsymbol{F}^{(\boldsymbol{A})}_{jr} \boldsymbol{F}^{(\boldsymbol{B})}_{kr}
\end{aligned}$$

For multiplication of $n \times n$ matrices,

- $\boldsymbol{T}$ is $n^2 \times n^2 \times n^2$
- classical algorithm has rank $R = n^3$
- Strassen's algorithm has rank $R \approx n^{\log_2(7)}$

Symmetric matrix times vector

Lets consider the simplest tensor contraction with symmetry

- let $\boldsymbol{A}$ be an n-by-n symmetric matrix ($\boldsymbol{A}_{ij} = \boldsymbol{A}_{ji}$)
- the symmetry is not preserved in matrix-vector multiplication

$$\boldsymbol{c} = \boldsymbol{A} \cdot \boldsymbol{b}$$

$$\boldsymbol{c}_i = \sum_{j=1}^{n} \underbrace{\boldsymbol{A}_{ij} \cdot \boldsymbol{b}_j}_{\text{nonsymmetric}}$$

- generally n^2 additions and n^2 multiplications are performed
- we can perform only $\binom{n+1}{2}$ multiplications using

$$\boldsymbol{c}_i = \sum_{j=1, j\neq i}^{n} \underbrace{\boldsymbol{A}_{ij} \cdot (\boldsymbol{b}_i + \boldsymbol{b}_j)}_{\text{symmetric}} + \underbrace{\left(\boldsymbol{A}_{ii} - \sum_{j=1, j\neq i}^{n} \boldsymbol{A}_{ij}\right) \cdot \boldsymbol{b}_i}_{\text{low-order}}$$

Symmetrized outer product

Consider a rank-2 outer product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of length n into symmetric matrix $\boldsymbol{C}$

$$\boldsymbol{C} = \boldsymbol{a} \cdot \boldsymbol{b}^{\mathsf{T}} + \boldsymbol{b} \cdot \boldsymbol{a}^{\mathsf{T}}$$
$$\boldsymbol{C}_{ij} = \underbrace{\boldsymbol{a}_i \cdot \boldsymbol{b}_j}_{\text{nonsymmetric}} + \underbrace{\boldsymbol{a}_j \cdot \boldsymbol{b}_i}_{\text{permutation}}$$

usually computed via the n^2 multiplications and n^2 additions
new algorithm requires $\binom{n+1}{2}$ multiplications

$$\boldsymbol{C}_{ij} = \underbrace{\underbrace{(\boldsymbol{a}_i + \boldsymbol{a}_j) \cdot (\boldsymbol{b}_i + \boldsymbol{b}_j)}_{\boldsymbol{Z}_{ij}}}_{\text{symmetric}} \underbrace{- \underbrace{\boldsymbol{a}_i \cdot \boldsymbol{b}_i}_{\boldsymbol{w}_i} - \underbrace{\boldsymbol{a}_j \cdot \boldsymbol{b}_j}_{\boldsymbol{w}_j}}_{\text{low-order}}$$

Symmetrized matrix multiplication

For symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, compute

$$\boldsymbol{C}_{ij} = \sum_{k=1}^{n} \Big(\underbrace{\boldsymbol{A}_{ik} \cdot \boldsymbol{B}_{kj}}_{\text{nonsymmetric}} + \underbrace{\boldsymbol{A}_{jk} \cdot \boldsymbol{B}_{ki}}_{\text{permutation}} \Big)$$

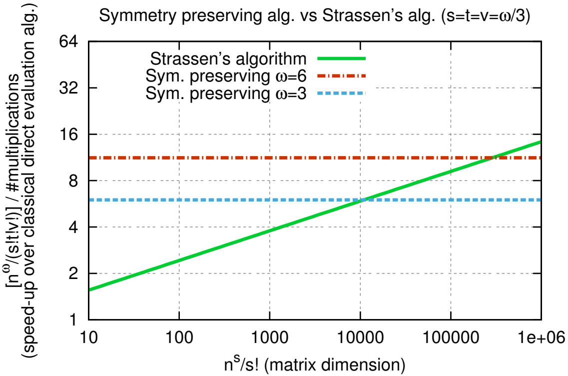
New algorithm requires $\binom{n}{3} + O(n^2)$ multiplications rather than n^3

$$\boldsymbol{C}_{ij} = \sum_{k} \underbrace{(\boldsymbol{A}_{ij} + \boldsymbol{A}_{ik} + \boldsymbol{A}_{jk}) \cdot (\boldsymbol{B}_{ij} + \boldsymbol{B}_{ik} + \boldsymbol{B}_{jk})}_{\boldsymbol{Z}_{ijk} \text{ -- symmetric}}$$

$$- \underbrace{\boldsymbol{A}_{ij} \cdot \Big(\sum_{k} \boldsymbol{B}_{ij} + \boldsymbol{B}_{ik} + \boldsymbol{B}_{jk} \Big)}_{\boldsymbol{U}_{ij} \text{ -- low-order}} - \underbrace{\boldsymbol{B}_{ij} \cdot \Big(\sum_{k} \boldsymbol{A}_{ij} + \boldsymbol{A}_{ik} + \boldsymbol{A}_{jk} \Big)}_{\boldsymbol{V}_{ij} \text{ -- low-order}}$$

$$- \underbrace{\sum_{k} \boldsymbol{A}_{ik} \cdot \boldsymbol{B}_{ik}}_{\boldsymbol{w}_{i} \text{ -- low-order}} - \underbrace{\sum_{k} \boldsymbol{A}_{jk} \cdot \boldsymbol{B}_{jk}}_{\boldsymbol{w}_{j} \text{ -- low-order}}$$

Comparison to Strassen's algorithm



Fully symmetric tensor contractions

For general symmetric tensor contraction algorithms,

- $\boldsymbol{A}$ of order $s+v$, and
- $\boldsymbol{B}$ of order $v+t$ into
- $\boldsymbol{C}$ of order $s+t$, defined as

we define the (nonsymmetrized) contraction as $\boldsymbol{C} = \boldsymbol{A} \odot_v \boldsymbol{B}$ where

$$\boldsymbol{C}_{\boldsymbol{ij}} = \sum_{\boldsymbol{k} \in \{1,\ldots,n\}^v} \boldsymbol{A}_{\boldsymbol{ik}} \boldsymbol{B}_{\boldsymbol{kj}}$$

then define the symmetrized tensor contraction as

$$\boldsymbol{C}_{\boldsymbol{i}} = [\boldsymbol{A} \odot_v \boldsymbol{B}]_{\boldsymbol{i}}$$

The usual method first computes $\boldsymbol{A} \odot_v \boldsymbol{B}$ with

$$\binom{n}{s}\binom{n}{t}\binom{n}{v} \approx \frac{n^{s+t+v}}{s!t!v!}$$

multiplications and additions

Fast symmetrized product of symmetric matrices

Using symmetrization notation for $s = t = v = 1$, we have fast algorithm:

$$\begin{aligned} \boldsymbol{C}_{ij} = [\boldsymbol{AB}]_{ij} &= \underbrace{\sum_k (\boldsymbol{A}_{ij} + \boldsymbol{A}_{ik} + \boldsymbol{A}_{jk}) \cdot (\boldsymbol{B}_{ij} + \boldsymbol{B}_{ik} + \boldsymbol{B}_{jk})}_{\sum_k [\boldsymbol{A}]_{ijk} \cdot [\boldsymbol{B}]_{ijk}} \\ &\quad \underbrace{- \boldsymbol{A}_{ij} \cdot \left(\sum_k \boldsymbol{B}_{ij} + \boldsymbol{B}_{ik} + \boldsymbol{B}_{jk} \right)}_{\boldsymbol{A}_{ij} \cdot \sum_k [\boldsymbol{B}]_{ijk}} \underbrace{- \boldsymbol{B}_{ij} \cdot \left(\sum_k \boldsymbol{A}_{ij} + \boldsymbol{A}_{ik} + \boldsymbol{A}_{jk} \right)}_{\boldsymbol{B}_{ij} \cdot \sum_k [\boldsymbol{A}]_{ijk}} \\ &\quad \underbrace{- \sum_k \boldsymbol{A}_{ik} \cdot \boldsymbol{B}_{ik} - \sum_k \boldsymbol{A}_{jk} \cdot \boldsymbol{B}_{jk}}_{[\boldsymbol{A}_{ik} \circ_1 \boldsymbol{B}]_{ij}} \\ &= \boxed{\sum_k [\boldsymbol{A}]_{ijk} \cdot [\boldsymbol{B}]_{ijk} - \boldsymbol{A}_{ij} \sum_k [\boldsymbol{B}]_{ijk} - \boldsymbol{B}_{ij} \sum_k [\boldsymbol{A}]_{ijk} - [\boldsymbol{A} \circ_1 \boldsymbol{B}]_{ij}} \end{aligned}$$

where $\boldsymbol{A} \circ_1 \boldsymbol{B} = \sum_k \boldsymbol{A}_{ik} \cdot \boldsymbol{B}_{jk}$

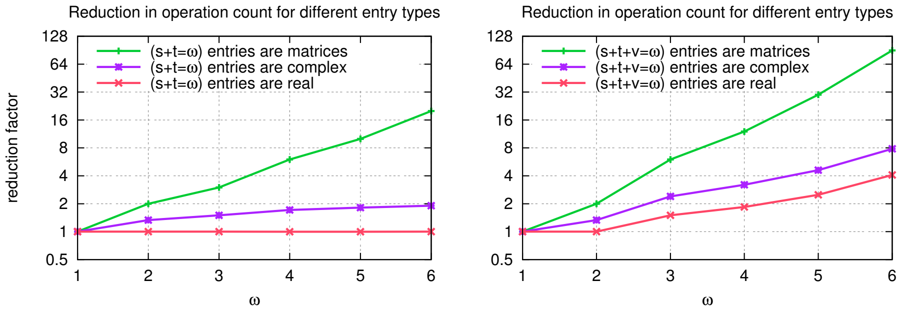
Fast fully-symmetric contraction algorithm

The fast algorithm is defined as follows (using $\omega = s + t + v$)

$$
\begin{aligned}
\boldsymbol{C_i} = &\underbrace{\sum_{\boldsymbol{k}\in\{1,\ldots,n\}^v} [\boldsymbol{A}]_{\boldsymbol{ik}} \cdot [\boldsymbol{B}]_{\boldsymbol{ik}}}_{\text{symmetric, requires } \binom{n+\omega-1}{\omega} \text{ multiplications}} \\
&- \underbrace{\sum_{p+q=1}^{v} \sum_{\boldsymbol{k}\in\{1,\ldots,n\}^{v-p-q}} \left(\sum_{\boldsymbol{p}\in\{1,\ldots,n\}^p} [\boldsymbol{A}]_{\boldsymbol{ikp}} \right) \cdot \left(\sum_{\boldsymbol{q}\in\{1,\ldots,n\}^q} [\boldsymbol{B}]_{\boldsymbol{ikq}} \right)}_{\text{requires } O(n^{\omega-1}) \text{ multiplications}} \\
&- \underbrace{\sum_{r=1}^{\min(s,t)} [\boldsymbol{A} \odot_{v+r} \boldsymbol{B}]_{\boldsymbol{i}}}_{\text{requires } O(n^{\omega-1}) \text{ multiplications}}
\end{aligned}
$$

Overall need $\binom{n}{\omega} + O(n^{\omega-1}) = \frac{n^{s+t+v}}{(s+t+v)!} + O(n^{s+t+v-1})$ multiplications, i.e. $(s+t+v)!/(s!t!v!)$ factor better

Reduction in operation count of fast algorithm with respect to standard



(s, t, v) values for left and right graph tabulated below

ω	1	2	3	4	4	6
Left graph	$(1,0,0)$	$(1,1,0)$	$(2,1,0)$	$(2,2,0)$	$(3,2,0)$	$(3,3,0)$
Right graph	$(1,0,0)$	$(1,1,0)$	$(1,1,1)$	$(2,1,1)$	$(2,2,1)$	$(2,2,2)$

Nesting the fast algorithm

For partially-(anti)symmetric contractions we can

- nest the new algorithm over each group of symmetric modes
- reduction in mults can translate to reduction in the number of operations
- for Hankel matrices, yields $O(n^{1.585})$ algorithm, which is better than naive ($O(n^2)$) but worse than DFT ($O(n)$)
- for coupled-cluster contractions, significant reductions in cost (number of operations) can be achieved
  - CCSD 1.3X on a typical system
  - CCSDT 2.1X on a typical system
  - CCSDTQ 5.7X on a typical system

Theoretical error bounds

We express error bounds in terms of $\gamma_n = \frac{n\epsilon}{1-n\epsilon}$, where ϵ is the machine precision.

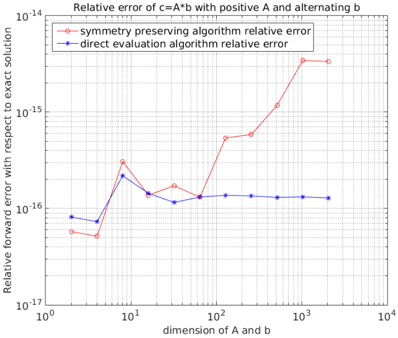
Let Ψ be the standard algorithm and Φ be the fast algorithm. The error bound for the standard algorithm arises from matrix multiplication

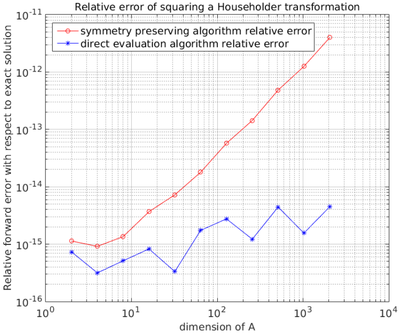
$$||fl\left(\Psi(\mathbf{A},\mathbf{B})\right) - \mathbf{C}||_\infty \leq \gamma_m \cdot ||\mathbf{A}||_\infty \cdot ||\mathbf{B}||_\infty \ \text{ where } \ m = \binom{n}{v}\binom{\omega}{v}.$$

The following error bound holds for the fast algorithm

$$||fl\left(\Phi(\mathbf{A},\mathbf{B})\right) - \mathbf{C}||_\infty \leq \gamma_m \cdot ||\mathbf{A}||_\infty \cdot ||\mathbf{B}||_\infty \ \text{ where } \ m = 3\binom{n}{v}\binom{\omega}{t}\binom{\omega}{s}.$$

Stability of symmetry preserving algorithms





Communication complexity

Parallelization is crucial for BLAS-like operations and scientific-applications

- can assess parallel scalability by considering communication complexity
- various notions of communication *complexity* or *cost* exist
- for the problems studied (which have high degree of concurrency), a simple measure is most important

  W – maximum number of words sent or received by any processor

- can derive lower and upper bounds for W for a given bilinear algorithm
- generally assume that tensor data is evenly distributed initially

Expansion in bilinear algorithms

The communication complexity of a bilinear algorithm depends on the amount of data needed to compute subsets of the bilinear products.

Definition (Bilinear subalgorithm)

Given $\Lambda = (\boldsymbol{F}^{(\boldsymbol{A})}, \boldsymbol{F}^{(\boldsymbol{B})}, \boldsymbol{F}^{(\boldsymbol{C})})$, $\Lambda_{\text{sub}} \subseteq \Lambda$ if $\exists$ projection matrix $\boldsymbol{P}$, so

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

The projection matrix extracts $\#\text{cols}(\boldsymbol{P})$ columns of each matrix.

Definition (Bilinear algorithm expansion)

A bilinear algorithm Λ has expansion bound $\mathcal{E}_\Lambda : \mathbb{N}^3 \to \mathbb{N}$, if for all

$$\Lambda_{\text{sub}} := (\boldsymbol{F}^{(\boldsymbol{A})}_{\textbf{sub}}, \boldsymbol{F}^{(\boldsymbol{B})}_{\textbf{sub}}, \boldsymbol{F}^{(\boldsymbol{C})}_{\textbf{sub}}) \subseteq \Lambda$$

we have $\text{rank}(\Lambda_{\text{sub}}) \leq \mathcal{E}_\Lambda \left(\text{rank}(\boldsymbol{F}^{(\boldsymbol{A})}_{\textbf{sub}}), \text{rank}(\boldsymbol{F}^{(\boldsymbol{B})}_{\textbf{sub}}), \text{rank}(\boldsymbol{F}^{(\boldsymbol{C})}_{\textbf{sub}})\right)$

For matrix mult., Loomis-Whitney inequality $\to \mathcal{E}_{\text{MM}}(x, y, z) = \sqrt{xyz}$

Communication cost of the standard algorithm

We consider communication bandwidth cost on a sequential machine with cache size M.

The intermediate formed by the standard algorithm may be computed via matrix multiplication with communication cost,

$$W(n,s,t,v,M) = \Theta\left(\frac{\binom{n}{s}\binom{n}{t}\binom{n}{v}}{\sqrt{M}} + \binom{n}{s+v} + \binom{n}{t+v} + \binom{n}{s+t}\right).$$

The cost of symmetrizing the resulting intermediate is low-order or the same.

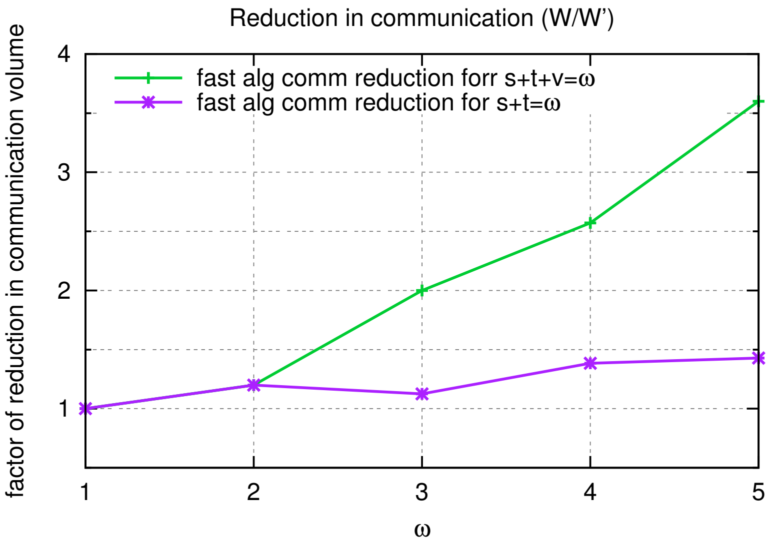
Communication cost of the fast algorithm

We can lower bound the cost of the fast algorithm using the Hölder-Brascamp-Lieb inequality.

An algorithm that blocks $\boldsymbol{Z}$ symmetrically nearly attains the cost

$$W'(n,s,t,v,M) = O\Bigg(\frac{\binom{n}{\omega}}{M^{\omega/(\omega-\min(s,t,v))}} \cdot \left[\binom{\omega}{t} + \binom{\omega}{s} + \binom{\omega}{v}\right] + \binom{n}{s+v} + \binom{n}{t+v} + \binom{n}{s+t}\Bigg).$$

which is not far from the lower bound and attains it when $s = t = v$.


Reduction in communication (W/W')
fast alg comm reduction forr s+t+v=ω
fast alg comm reduction for s+t=ω
factor of reduction in communication volume
ω
1
2
3
4
5
1
2
3
4

Communication cost of symmetry preserving algorithms

For contraction of order $s + v$ tensor with order $v + t$ tensor

- 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 $s \neq v \neq t$

Summary of results

The following table lists the leading order number of multiplications F required by the standard algorithm and F' by the fast algorithm for various cases of symmetric tensor contractions

ω	s	t	v	F	F'	applications
2	1	1	0	n^2	$n^2/2$	syr2, syr2k, her2, her2k
2	1	0	1	n^2	$n^2/2$	symv, symm, hemv, hemm
3	1	1	1	n^3	$n^3/6$	symmetrized matmul
s+t+v	s	t	v	$\binom{n}{s}\binom{n}{t}\binom{n}{v}$	$\binom{n}{\omega}$	any symmetric tensor contraction

High-level conclusions:

- Algebraic complexity result for leveraging symmetry in contractions
- Applications for basic complex arithmetic and partially symmetric contractions
- Caveats: more communication per flop, slightly higher numerical error

Acknowledgements and references

Collaborators on various parts:

- James Demmel
- Torsten Hoefler
- Devin Matthews

S., Demmel; Technical Report, ETH Zurich, December 2015.
S., Demmel, Hoefler; Technical Report, ETH Zurich, January 2015.

Backup slides

The fast algorithm for computing $\boldsymbol{C}$ forms the following intermediates with $\binom{n}{\omega}$ multiplications (where $\omega = s + t + v$),

$$
\begin{aligned}
Z_{\boldsymbol{i}} =& \left(\sum_{\boldsymbol{j}\in\chi(\boldsymbol{i})} A_{\boldsymbol{j}}\right) \cdot \left(\sum_{\boldsymbol{l}\in\chi(\boldsymbol{i})} B_{\boldsymbol{l}}\right) \\
V_{\boldsymbol{i}} =& \left(\sum_{\boldsymbol{j}\in\chi(\boldsymbol{i})} A_{\boldsymbol{j}}\right) \cdot \left(\sum_{k_1} \sum_{\boldsymbol{l}\in\chi(\boldsymbol{i}\cup\boldsymbol{k})} B_{\boldsymbol{l}}\right) \\
&+ \left(\sum_{k_1} \sum_{\boldsymbol{j}\in\chi(\boldsymbol{i}\cup\boldsymbol{k})} A_{\boldsymbol{j}}\right) \cdot \left(\sum_{\boldsymbol{l}\in\chi(\boldsymbol{i})} B_{\boldsymbol{l}}\right) \\
W_{\boldsymbol{i}} =& \left(\sum_{\boldsymbol{j}\in\chi(\boldsymbol{i})} A_{\boldsymbol{j}}\right) \cdot \left(\sum_{\boldsymbol{l}\in\chi(\boldsymbol{i})} B_{\boldsymbol{l}}\right) \\
C_{\boldsymbol{i}} =& \sum_{\boldsymbol{k}} Z_{\boldsymbol{i}\cup\boldsymbol{k}} - \sum_{\boldsymbol{k}} V_{\boldsymbol{i}\cup\boldsymbol{k}} \\
&- \sum_{\boldsymbol{j}\in\chi(\boldsymbol{i})} \left(\sum_{\boldsymbol{k}} W_{\boldsymbol{j}\cup\boldsymbol{k}}\right)
\end{aligned}
$$