Tensor Algorithms and Libraries for Quantum Chemistry and Materials Science

Edgar Solomonik

L P N A @CS@Illinois

Department of Computer Science
University of Illinois at Urbana-Champaign

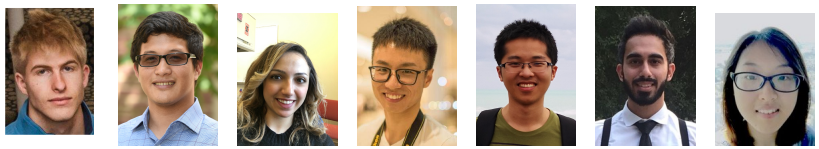
MolSSI Workshop on HPC in Computational Chemistry and Materials Science

Laboratory for Parallel Numerical Algorithms

Group research foci

- numerical linear algebra
- numerical optimization
- parallel algorithms
- tensor decompositions
- tensor networks
- quantum chemistry
- quantum simulation
- software automation

L P N A @CS@Illinois



DOE CSGF

NSF

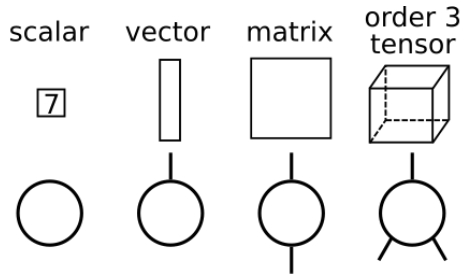
http://lpna.cs.illinois.edu

Outline

Tensors

A tensor is a collection of elements

- its dimensions define the size of the collection
- its order is the number of different dimensions
- specifying an index along each tensor mode defines an element of the tensor



A few examples of tensors are

- Order 0 tensors are scalars, e.g., $s \in \mathbb{R}$
- Order 1 tensors are vectors, e.g., $\boldsymbol{v} \in \mathbb{R}^n$
- Order 2 tensors are matrices, e.g., $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
- An order 3 tensor with dimensions $s_1 \times s_2 \times s_3$ is denoted as $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{s_1 \times s_2 \times s_3}$ with elements t_{ijk} for $i \in \{1, \ldots, s_1\}, j \in \{1, \ldots, s_2\}, k \in \{1, \ldots, s_3\}$

Tensor Contractions

A tensor contraction describes a set of products and sums of elements from two tensors

tensor contraction	formula
inner product	$w = \sum_i u_i v_i$
outer product	$w_{ij} = u_i v_{ij}$
pointwise product	$w_i = u_i v_i$
Hadamard product	$w_{ij} = u_{ij} v_{ij}$
matrix multiplication	$w_{ij} = \sum_k u_{ik} v_{kj}$
batched mat.-mul.	$w_{ijl} = \sum_k u_{ikl} v_{kjl}$
tensor times matrix	$w_{ilk} = \sum_j u_{ijk} v_{lj}$

Tensor contractions are prevalent in quantum chemistry methods

General Tensor Contractions

Given tensor $\mathcal{U}$ of order $s+v$ and $\mathcal{V}$ of order $v+t$, a tensor contraction summing over v modes can be written as

$$w_{i_1 \ldots i_s j_1 \ldots j_t} = \sum_{k_1 \ldots k_v} u_{i_1 \ldots i_s k_1 \ldots k_v} v_{k_1 \ldots k_v j_1 \ldots j_t}$$

- Other contractions can be mapped to this form after transposition

Unfolding tensors reduces the tensor contraction to matrix multiplication

- Combine consecutive indices in appropriate groups of size s, t, and v
- If all tensor modes are of dimension n, obtain matrix–matrix product $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ where $\boldsymbol{C} \in \mathbb{R}^{n^s \times n^t}$, $\boldsymbol{A} \in \mathbb{R}^{n^s \times n^v}$, and $\boldsymbol{B} \in \mathbb{R}^{n^v \times n^t}$
- Assuming classical matrix multiplication, contraction requires n^{s+t+v} elementwise products and $n^{s+t+v} - n^{s+t}$ additions

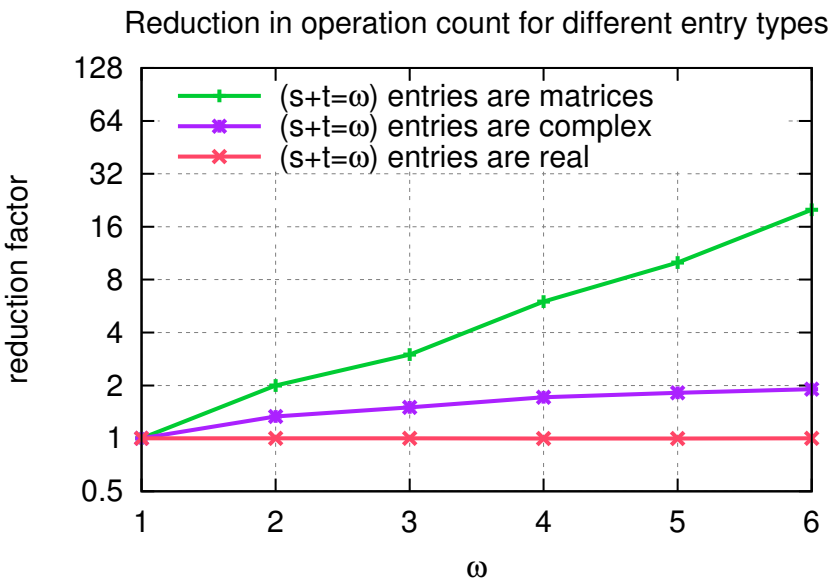
Symmetric Tensor Contractions

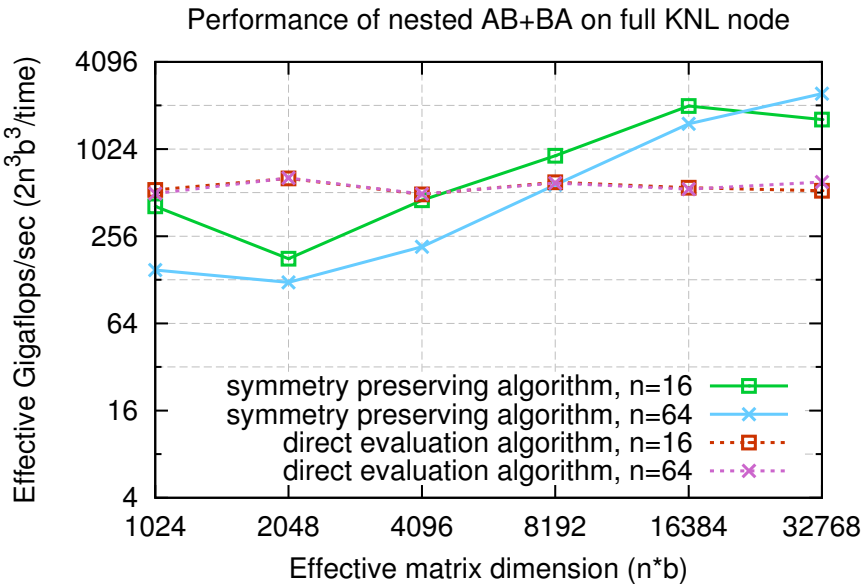
- A symmetric tensor is defined by e.g.,
  $t_{ijk} = t_{ikj} = t_{kij} = t_{jki} = t_{jik} = t_{kji}$
- Tensors can also have skew-symmetry (also known as antisymmetry, permutations have $+/-$ signs), partial symmetry (only some modes are permutable), or group symmetry (blocks are zero if indices satisfy modular equation)
- The simplest example of a symmetric tensor contraction is

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \quad \text{where } \boldsymbol{A} = \boldsymbol{A}^T$$

  it is not obvious how to leverage symmetry to reduce cost of this contraction

Permutational Symmetry in Tensor Contractions





New contraction algorithms reduce cost via permutational symmetry[1]

- Symmetry is hard to use in contraction e.g. $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ with $\boldsymbol{A}$ symmetric
- For contraction of order $s + v$ and $v + t$ tensors to produce an order $s + t$ tensor, previously known approaches reduce cost by $s!t!v!$
- New algorithm reduces number of *products* by $\omega!$ where $\omega = s + t + v$, leads to same reduction in *cost* for partially-symmetric contractions

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A} \Rightarrow c_{ij} = \sum_k [(a_{ij} + a_{ik} + a_{jk}) \cdot (b_{ij} + b_{ik} + b_{jk})] - \ldots$$

[1] E.S, J. Demmel, Computational Methods in Applied Mathematics 2020

Communication Cost of Symmetry Preserving Algorithms

- Preserving symmetry reduces memory footprint and cost, but can entail additional data dependencies and communication cost
- We have introduced a framework of communication lower bounds for bilinear algorithms[1] and applied it to symmetric tensor contractions[2,3]
- These lower bounds show that asymptotically more communication is necessitated by both symmetric packed layouts and symmetry preserving contraction algorithms
- However, the overheads are present only for sophisticated tensor contractions (high-order and with different number of contracted/uncontracted modes)

[1] V. Pan, SIAM Review 1984

[2] E.S., J. Demmel, T. Hoefler, SIAM Journal on Scientific Computing 2021

[3] C. Ju, Y. Zhang, E.S., arXiv:2107.09834

Group Symmetry

- Abelian group symmetries can be mapped to the cyclic group, which can be used to define a block-sparse form of the tensors (here represented using extra modes), e.g.,

$$w_{aA,bB,iI,jJ} = \sum_{k,K,l,L} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}$$

where for some group size G, we have symmetries, e.g.,

$$w_{aA,bB,iI,jJ} \neq 0 \text{ if } A + B - I - J \equiv 0 \pmod{G}$$
$$u_{aA,bB,kK,lL} \neq 0 \text{ if } A + B + K + L \equiv 0 \pmod{G}$$
$$v_{kK,lL,iI,jJ} \neq 0 \text{ if } K + L - I - J \equiv 0 \pmod{G}$$

- We can write each of these tensors using a reduced form and a Kronecker delta tensor,

$$w_{aA,bB,iI,jJ} = r^{(W)}_{aA,bB,iI,j} \delta^{(W)}_{ABIJ}$$

where $\delta^{(W)}_{ABIJ} = 1$ if $A + B - I - J \equiv 0 \pmod{G}$ and $\delta^{(W)}_{ABIJ} = 0$ otherwise

Block Contraction Approach to Group Symmetry

Such symmetries are often handled by indirect indexing in nested loops

Algorithm 2.1 Loop nest to perform group symmetric contraction $w_{aA,bB,iI,jJ} = \sum_{k,K,l,L} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}$ using standard reduced forms $\bar{w}_{aA,bB,iI,j}$, $\bar{u}_{aA,bB,kK,l}$, and $\bar{v}_{kK,lL,iI,j}$.

```
for A = 1, ..., G do
    for B = 1, ..., G do
        for I = 1, ..., G do
            J = A + B − I mod G
            for K = 1, ..., G do
                L = A + B − K mod G
                ∀a, b, i, j,   w̄_{aA,bB,iI,j} = w̄_{aA,bB,iI,j} + Σ_{k,l} ū_{aA,bB,kK,l} v̄_{kK,lL,iI,j}
            end for
        end for
    end for
end for
```

However, transformations of tensors are also possible to reduce such contractions to a "direct product", which has previously been done for group symmetric tensor contractions in quantum chemistry[1,2]

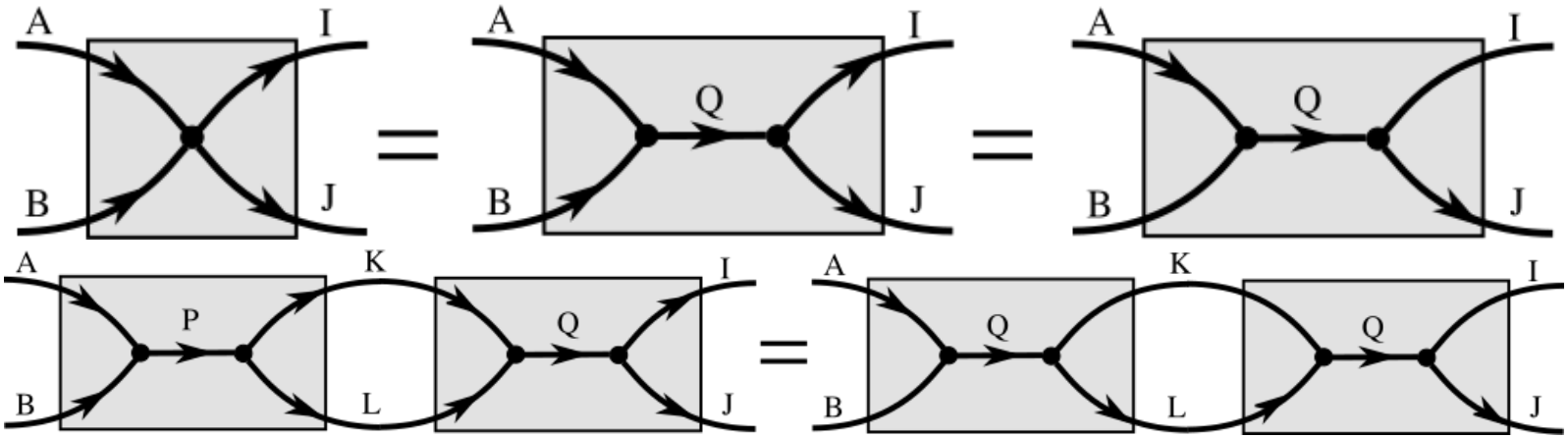
[1] J.F. Stanton, J. Gauss, J.D. Watts, and R.J. Bartlett, The Journal of Chemical Physics 1991

[2] D. Matthews, Molecular Physics 2019

Group Symmetry in Tensor Contractions

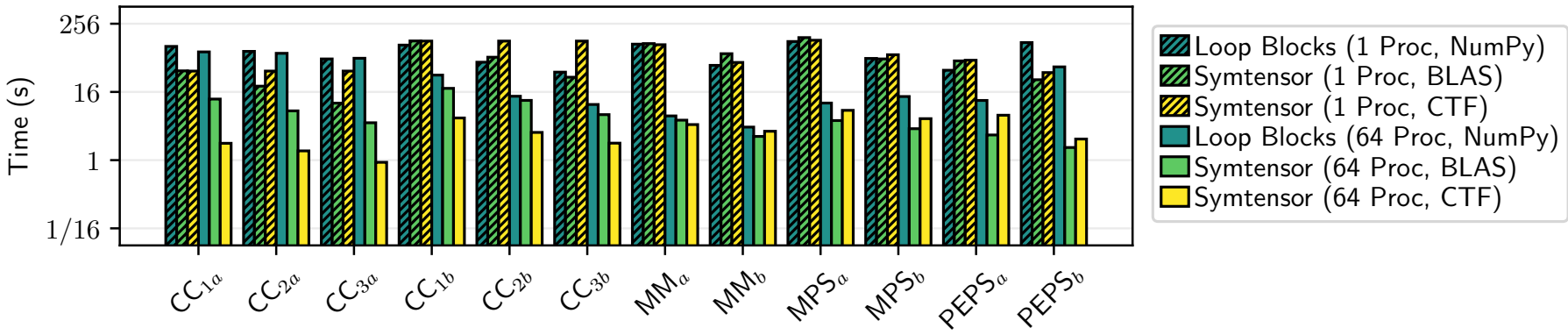
New contraction algorithm, *irreducible representation alignment* uses new reduced form to handle group symmetry (momentum conservation, spin, quantum numbers, etc.) without looping over blocks or sparsity[1]

$$w_{ABIJ} = \sum_{KL} \bar{r}^{(U)}_{ABK} \underbrace{\delta^{(U)}_{ABKL}\delta^{(V)}_{KLIJ}}_{\sum_Q \delta^{(1)}_{ABQ}\delta^{(2)}_{IJQ}\delta^{(3)}_{KLQ}} \bar{r}^{(V)}_{KIJ} = \sum_Q \delta^{(1)}_{ABQ}\delta^{(3)}_{IJQ} \underbrace{\sum_K r^{(U)}_{AKQ} r^{(V)}_{KIQ}}_{r^{(W)}_{AIQ}}$$



[1] Y. Gao, P. Helms, G. Chan, and E.S., arXiv:2007.08056

Automation of Group Symmetric Contractions



- Group symmetric tensors represented programmatically by
  - a dense reduced tensor (containing unique data)
  - an implicit sparse tensor (Kronecker delta tensor) describing the group symmetry
- At contraction time reduced form are aligned by contraction with Kronecker delta tensor (Q index is introduced)
- Users can write symmetry-oblivious code

Library for Massively-Parallel Tensor Computations

Cyclops Tensor Framework[1]: sparse/dense generalized tensor algebra

- Cyclops is a C++ library that distributes each tensor over MPI
- Used in chemistry (PySCF, QChem, CC4S)[2], quantum circuit simulation (by IBM/LLNL)[3], and graph analysis (betweenness centrality[4], minimum spanning tree[5])
- Summations and contractions specified via Einstein notation

```
E["aixbjy"] += X["aixbjy"] - U["abu"]*V["iju"]*W["xyu"]
```

- Best distributed contraction algorithm selected at runtime via models
- Support for Python (`numpy.ndarray` backend), OpenMP, and GPU
- Simple interface to core ScaLAPACK matrix factorization routines

[1] https://github.com/cyclops-community/ctf

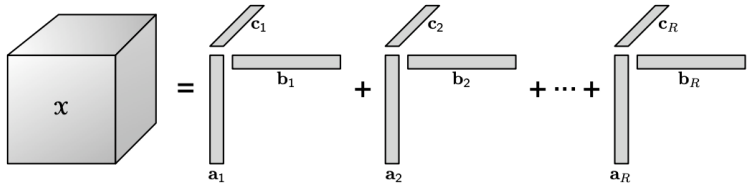
[2] E.S., D. Matthews, J. Hammond, J.F. Stanton, J. Demmel, JPDC 2014

[3] E. Pednault, J.A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E.S., E. Draeger, E. Holland, and R. Wisnieff, 2017

[4] E.S., M. Besta, F. Vella, T. Hoefler, SC 2017

[5] T. Baer, R. Kanakagiri, E.S., SIAM PP 2022

CP Decomposition



- For a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n \times n \times n}$, the CP decomposition[1,2] is defined by matrices $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{W}$ such that

$$t_{ijk} = \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr}$$

[1] F.L. Hitchcock, Studies in Applied Mathematices 1927

[2] T. Kolda and B. Bader, SIAM Review 2009

CP Decomposition for Tensor Hypercontraction

- The cost of CCSD can be reduced to $O(n^5)$ by density fitting, which is a truncated Cholesky decomposition of the ERI tensor

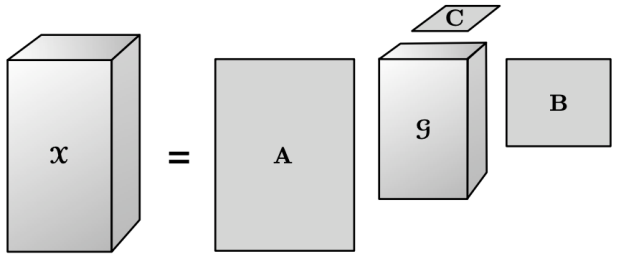
$$(ab|ij) = \sum_p d_{abp} d^*_{ijp}$$

- The tensor hypercontraction method factorizes the density fitting tensor as

$$d_{ijp} = \sum_r x_{ir} x_{jr} y_{pr}$$

  which is a *canonical polyadic (CP) decomposition* with a repeating factor matrix $\boldsymbol{X}$
- When this factorization is also applied to the amplitude tensor, CCSD scaling can be theoretically further reduced to $O(n^4)$

Tucker Decomposition



- The Tucker decomposition[1] expresses an order d tensor via a smaller order d core tensor and d factor matrices
  - For a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n \times n \times n}$, the Tucker decomposition is defined by core tensor $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ and factor matrices $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{W}$ with orthonormal columns, such that

$$t_{ijk} = \sum_{p=1}^{R_1} \sum_{q=1}^{R_2} \sum_{r=1}^{R_3} z_{pqr} u_{ip} v_{jq} w_{kr}$$

  - If an exact Tucker decomposition exists, it can be computed via SVD (HoSVD)
  - HOOI method optimizes in an alternating manner among $(\boldsymbol{U}, \boldsymbol{\mathcal{Z}})$, $(\boldsymbol{V}, \boldsymbol{\mathcal{Z}})$, $(\boldsymbol{W}, \boldsymbol{\mathcal{Z}})$

[1] T. Kolda and B. Bader, SIAM Review 2009

Recent Work on Tensor Decompositions

Our group has a number of recent developments in algorithms and parallel software for tensor decomposition optimization algorithms

- Navjot Singh, Linjian Ma, Hongru Yang, and ES. *Comparison of accuracy and scalability of Gauss-Newton and alternating least squares for CP decomposition*, arXiv:1910.12331 (SISC 2021).
- Linjian Ma and ES. *Accelerating alternating least squares for tensor decomposition by pairwise perturbation*, arXiv:1811.10573 (NLAA 2022).
- Linjian Ma and ES. *Efficient parallel CP decomposition with pairwise perturbation and multi-sweep dimension tree*, arXiv:2010.12056 (IPDPS 2021).
- Linjian Ma and ES. *Fast and accurate randomized algorithms for low-rank tensor decompositions*, arxiv.org:2104.0110 (NeurIPS 2021).

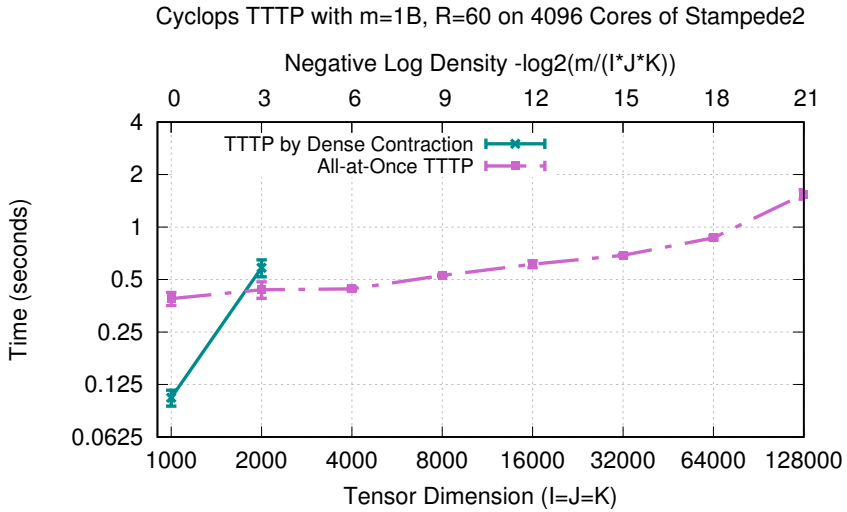
Tensor Completion

- The tensor completion problem seeks to build a model (e.g., CP decomposition) for a partially-observed tensor
- For an order three tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n \times n \times n}$, given a set of observed entries t_{ijk} for $(i,j,k) \in \Omega$, we seek to minimize

$$\sum_{(i,j,k)\in\Omega} \underbrace{(t_{ijk} - \sum_{r} u_{ir} v_{jr} w_{kr})^2}_{\text{loss function}} + \lambda^2(\|\boldsymbol{U}\|_2^2 + \|\boldsymbol{V}\|_2^2 + \|\boldsymbol{W}\|_2^2)$$

- Completion objective differs from decomposition of a sparse tensor, as it excludes unobserved entries
- Other loss functions than quadratic loss are often interest for different tensor data

Tensor Completion



- Via the Cyclops Python interface, we have implemented parallel (over MPI) completion with SGD, CCD, ALS (with iterative and direct solves), and Gauss-Newton, with support for generalized loss[1]
- Tensor times tensor product (TTTP) routine enables CP tensor completion

$$r_{ijk} = \sum_r t_{ijk} u_{ir} v_{jr} w_{kr}$$

- For ALS, explicit parallel direct solves[2] are fastest

[1] N. Singh, Z. Zhang, X. Wu, N. Zhang, S. Zhang, and E.S., arXiv:1910.02371

[2] S. Smith, J. Park, and G. Karypis, SC 2016

All-at-once Contraction

- Contraction of more 3 or more tensors may be performed by contracting two tensors at a time
- This approach is often suboptimal in the presence of sparsity
- Customized routines have been developed as a result for various specific multi-tensor contractions: MTTKRP, SDDMM, TTTP, TTMc
- Further challenges are posed by needing to form and solve linear least squares problems on the fly, as needed by tensor completion and quasi-robust density fitting[1]
- We are working toward general all-at-once multi-tensor contraction routines and on-the-fly linear solvers as part of Cyclops

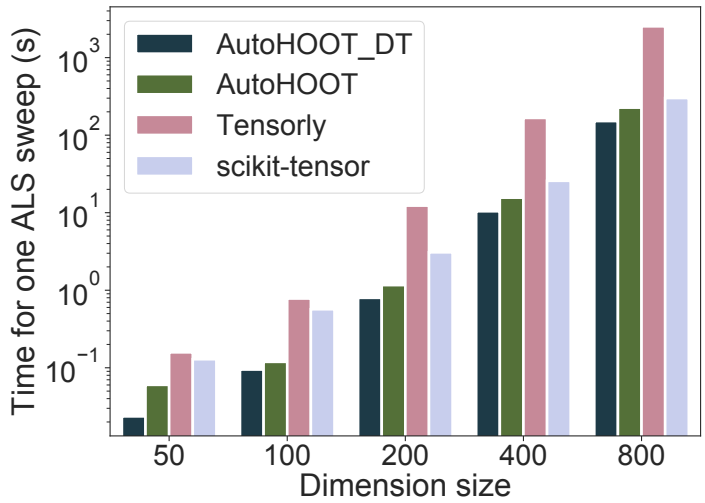
[1] D.P. Tew, The Journal of Chemical Physics 2018

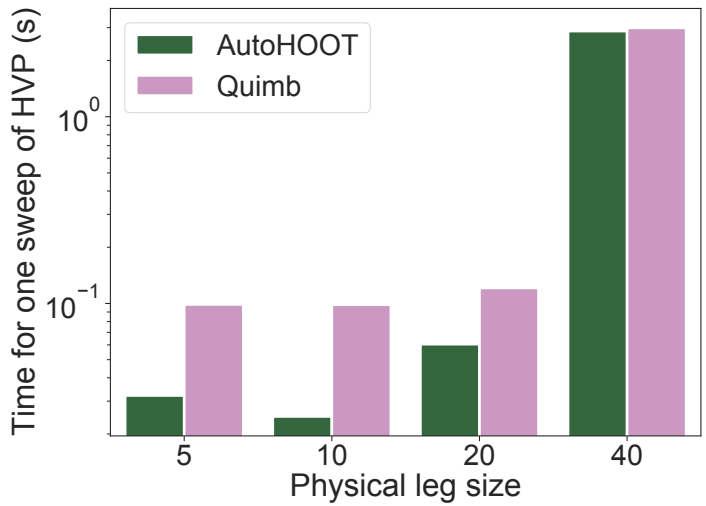
Automatic Generation of Tensor Optimization Methods

- Automatic differentiation (AD) in principle enables automatic generation of methods such as ALS and DMRG
- Both apply Newton's method on a sequence of subsets of variables
- However, existing AD tools such as Jax (used by TensorFlow) are designed for deep learning and are ineffective for more complex tensor computations
  - focus purely on first order optimization via Jacobian-vector products
  - unable to propagate tensor algebra identities such as $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$ to generate efficient code

Automatic High-Order Optimization for Tensors

- AutoHOOT[1] provides a tensor-algebra centric AD engine
- Designed for `einsum` expressions and alternating minimization common in tensor decomposition and tensor network methods
- Python-level AD is coupled with optimization of contraction order and caching of intermediates
- Generates code for CPU/GPU/supercomputers using high-level back-end interface to tensor contractions





[1] Linjian Ma, Jiayu Ye, and E.S., arXiv:2005.04540, PACT 2020

Further References

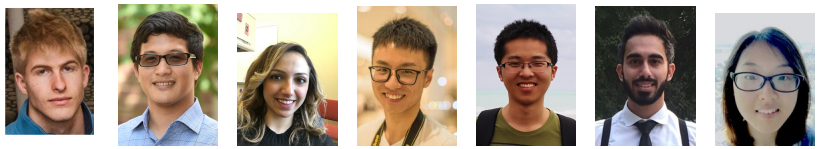
- We have presented innovations to numerical algorithms and software libraries for tensor contractions, decompositions, and tensor networks
- All software libraries and results discussed in this presentation are available in open source via `https://github.com/cyclops-community/` and `https://github.com/LinjianMa/AutoHOOT`
- Our research group is developing an ecosystem of algorithms and software for quantum chemistry calculations and other applications in quantum simulation
- See our group website[1] for further details/references

[1] `http://lpna.cs.illinois.edu`

Acknowledgements

- Laboratory for Parallel Numerical Algorithms (LPNA) at University of Illinois, `lpna.cs.illinois.edu` and collaborators
- Funding from NSF awards: #1839204 (RAISE-TAQS), #1931258 (CSSI), #1942995 (CAREER)
- Stampede2 resources at TACC via XSEDE

L·P·N·A @CS@Illinois



DOE CSGF

NSF

`http://lpna.cs.illinois.edu`