Tensor Optimization Algorithms and Libraries for Quantum Simulation

Edgar Solomonik

L P N A @CS@Illinois

Department of Computer Science
University of Illinois at Urbana-Champaign

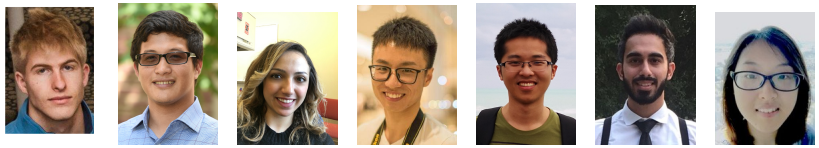
Quantum Information for Mathematics, Economics, and Statistics
IMSI

Laboratory for Parallel Numerical Algorithms

Recent/ongoing research topics
(*-covered today)

- parallel matrix computations
  - matrix factorizations
  - eigenvalue problems
  - preconditioners
- tensor computations
  - tensor decomposition*
  - sparse tensor kernels*
  - tensor completion*
- simulation of quantum systems
  - tensor networks*
  - quantum chemistry
  - quantum circuits*
- fast bilinear algorithms
  - convolution algorithms
  - tensor symmetry*

L P N A @CS@Illinois



DOE CSGF

NSF

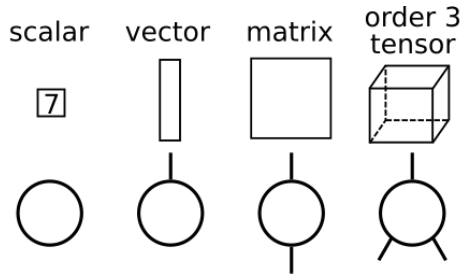
http://lpna.cs.illinois.edu

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 multiplies elements of two tensors and computes partial sums to produce a third, in a fashion expressible by pairing up modes of different tensors, which may be expressed in Einstein summation notation (einsum)

tensor contraction	*einsum*
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	$w_{ilk} = \sum_j u_{ijk} v_{lj}$

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 the tensors reduces the tensor contraction to matrix multiplication

- Combine consecutive indices in appropriate groups of size s ,t, or 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

Emulation of Quantum Gates and Quantum Circuits

- Consider an n-qubit quantum state

$$|\psi\rangle = \sum_{\boldsymbol{i}\in\{0,1\}^n} t^{\psi}_{\boldsymbol{i}} \, |i_1 \cdots i_n\rangle \text{ with } t_{\boldsymbol{i}} \in \mathbb{C}$$

- Quantum circuits generally consist of 1-qubit and 2-qubit gates

$$|\phi\rangle = \boldsymbol{U}^{(s)} |\psi\rangle \Rightarrow t^{\phi}_{i_1\cdots i_n} = \sum_{j_s=0}^{1} u^{(s)}_{i_s j_s} t^{\psi}_{i_1\cdots i_{s-1} j_s i_{s+1}\cdots i_n}$$

$$|\phi\rangle = \boldsymbol{U}^{(s,t)} |\psi\rangle \Rightarrow t^{\phi}_{i_1\cdots i_n} = \sum_{j_s=0}^{1}\sum_{j_t=0}^{1} u^{(s)}_{i_s i_t j_s j_t} t^{\psi}_{i_1\cdots i_{s-1} j_s i_{s+1}\cdots i_{t-1} j_t i_{t+1}\cdots i_n}$$

- A quantum gate can be emulated as an $O(2^n)$-cost tensor contraction
- An n-qubit quantum circuit with depth D and $O(nD)$ gates can be simulated classically with $O(nD2^n)$ cost and $O(2^n)$ storage

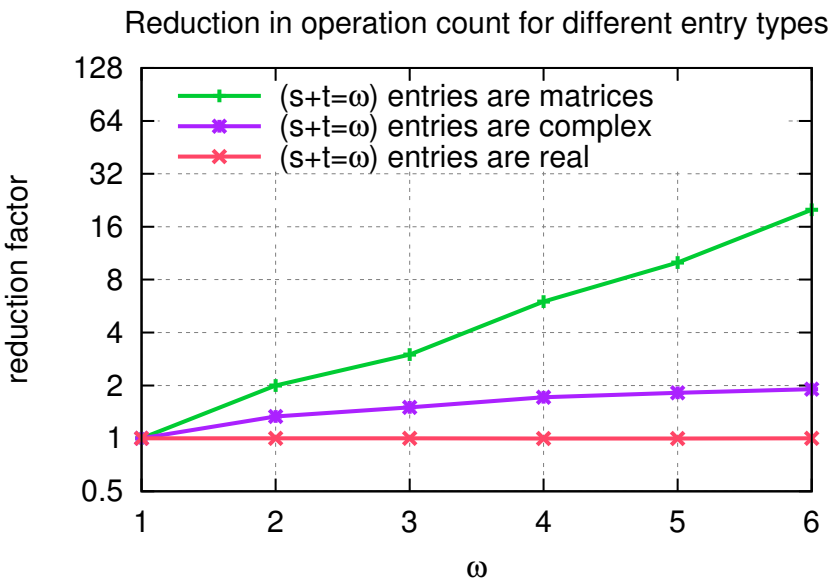
Symmetric Tensor Contractions

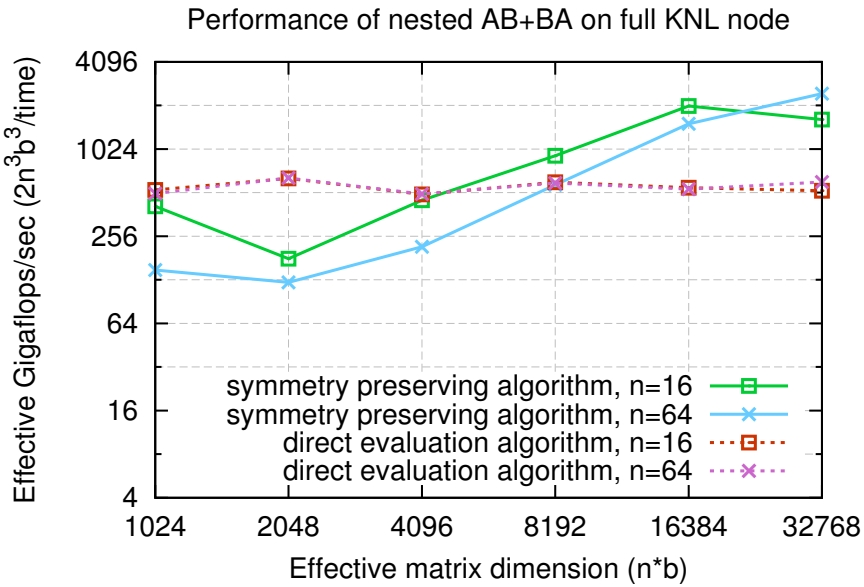
- Recall a symmetric tensor is defined by e.g.,
  $t_{ijk} = t_{ikj} = t_{kij} = t_{jki} = t_{jik} = t_{kji}$
- Tensors can also have skew-symmetry (also known as antisymmetry, permutations have $+/-$ signs), partial symmetry (only some modes are permutable), or group symmetry (blocks are zero if indices 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, CMAM 2020

Group Symmetry

- Tensors arising in physical simulations often have group structure that reflects conservation laws
  - Abelian group symmetries can be mapped to 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.,

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

  - We can write each of these tensors using a reduced form and an irrep map,

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

    where $m_{ABIJ} = 1$ if $A + B - I - J \equiv 0 \pmod{G}$ and $m_{ABIJ} = 0$ otherwise

Block Contraction Approach to Group Symmetry

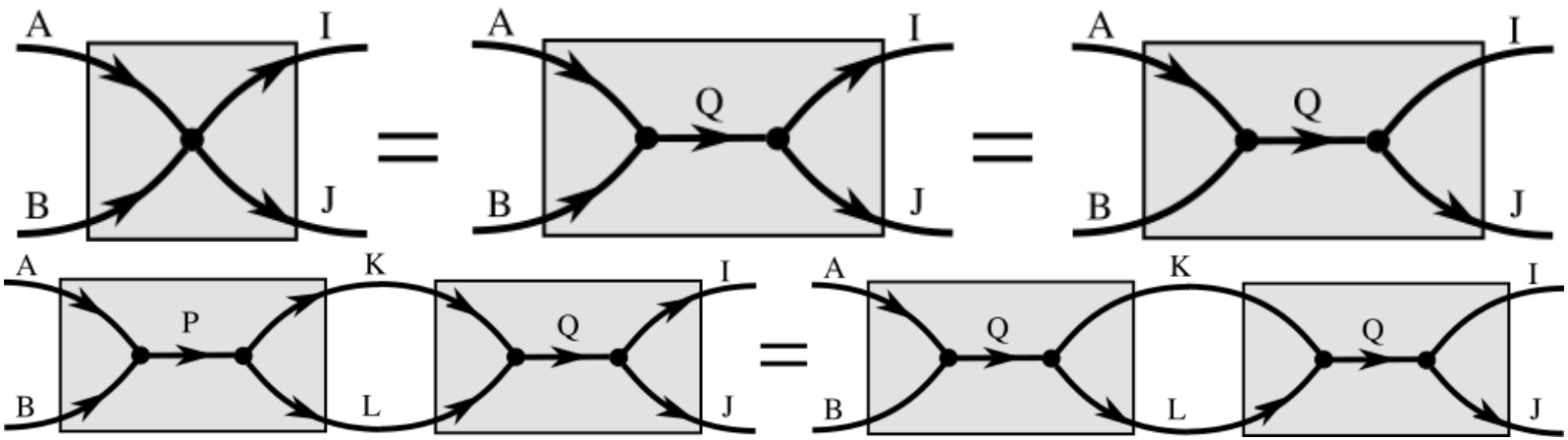
Algorithm 3.1 Loop nest to perform group symmetric contraction $w_{aA,bB,iI,jJ} = \sum_{k,K,l,L} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}$.

for $A = 1, \ldots, G$ **do**
  for $B = 1, \ldots, G$ **do**
    for $I = 1, \ldots, G$ **do**
      $J = A + B - I \bmod G$
      for $K = 1, \ldots, G$ **do**
        $L = -A - B - K \bmod G$
        $\forall a, b, i, j, \quad w_{aA,bB,iI,jJ} = \sum_{k,l} w_{aA,bB,iI,jJ} + u_{aA,bB,kK,lL}, v_{kK,lL,iI,jJ}$
      end for
    end for
  end for
end for

Group Symmetry in Tensor Contractions

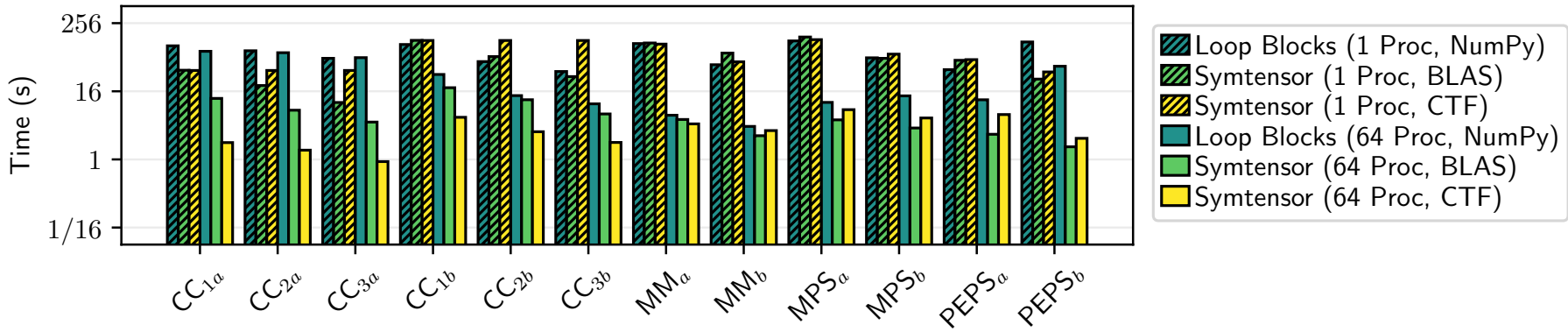
New contraction algorithm, *irreducible representation alignment* uses new reduced form to handle group symmetry (momentum conservation, spin, quantum numbers, etc.) without looping over blocks or sparsity[1]

$$\begin{aligned}\hat{w}_{aA,b,i,jJ,Q} &= w_{aA,b,Q-A \bmod G,i,-J-Q \bmod G,jJ}, \\ &= \sum_{L,k,l} \hat{u}_{aA,b,k,lL,Q} \hat{v}_{k,lL,i,jJ,Q}.\end{aligned}$$



[1] collaboration with Yang Gao, Phillip Helms, and Garnet Chan at Caltech, to appear on arxiv, July 2020

Automation of Group Symmetric Contractions



- Group symmetric tensors represented programmatically by
  - a dense reduced tensor (containing unique data)
  - an implicit sparse tensor (irrep map) describing the group symmetry
- At contraction time reduced form is transformed so as to align irrep maps (introduce Q index)
- 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)[2], quantum circuit simulation (IBM/LLNL)[3], and graph analysis (betweenness centrality)[4]
- 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

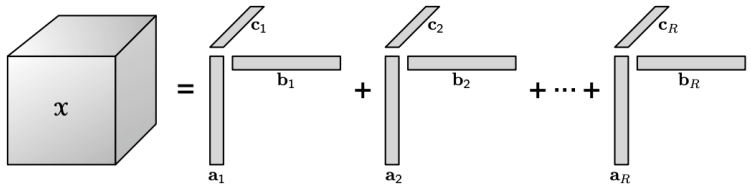
[1] https://github.com/cyclops-community/ctf

[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

CP Decomposition



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

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

- For an order N tensor, the decomposition generalizes as follows,

$$t_{i_1 \ldots i_d} = \sum_{r=1}^{R} \prod_{j=1}^{d} u_{i_j r}^{(j)}$$

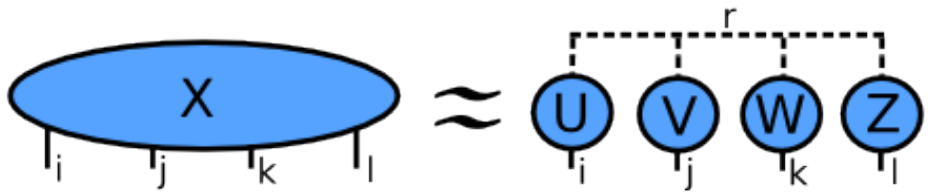
- Its rank is generally bounded by $R \leq n^{d-1}$

[1] Kolda and Bader, SIAM Review 2009

Alternating Least Squares for CP Decomposition

Consider rank R CP decomposition of an $s \times s \times s \times s$ tensor

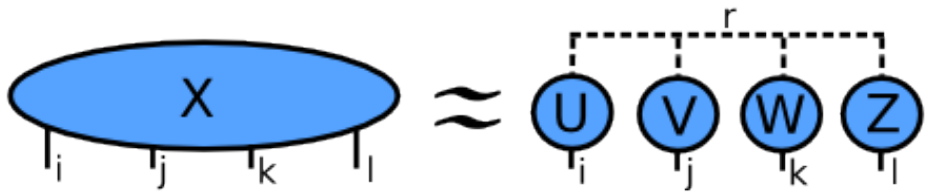
$$x_{ijkl} \approx \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} z_{lr}$$



Alternating Least Squares for CP Decomposition

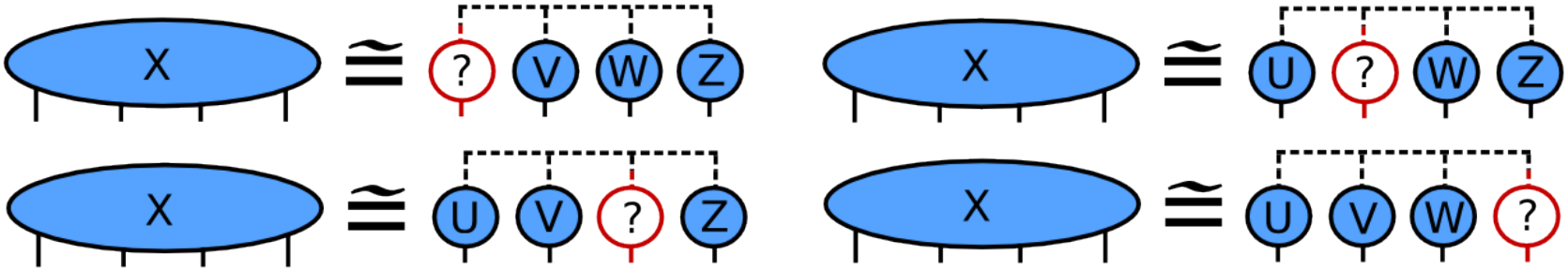
Consider rank R CP decomposition of an $s \times s \times s \times s$ tensor

$$x_{ijkl} \approx \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} z_{lr}$$



ALS updates factor matrices in an alternating manner

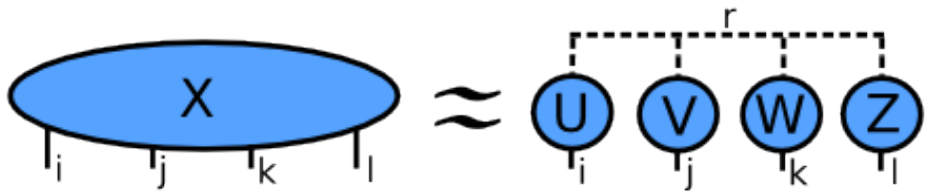
$$\min_{\boldsymbol{A}^{(n)}} f(\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}) = \frac{1}{2} ||\boldsymbol{\mathcal{X}} - [\![\boldsymbol{A}^{(1)}, \cdots, \boldsymbol{A}^{(n)}, \cdots, \boldsymbol{A}^{(N)}]\!]||_F^2,$$



Alternating Least Squares for CP Decomposition

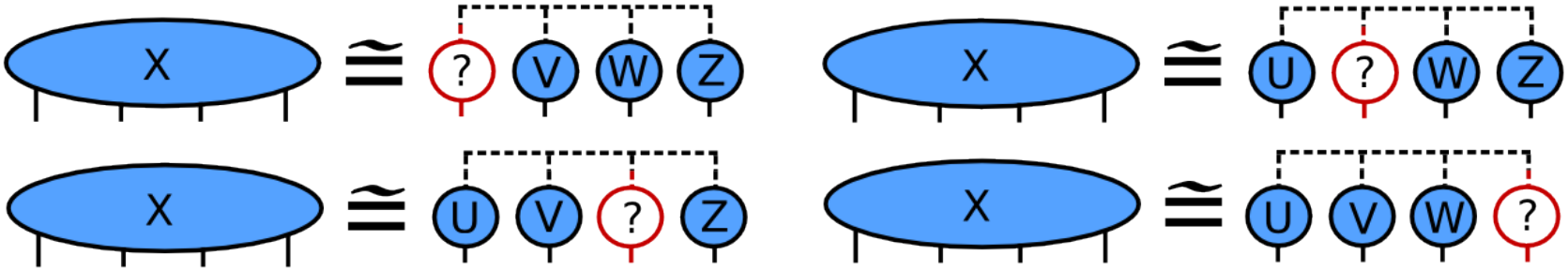
Consider rank R CP decomposition of an $s \times s \times s \times s$ tensor

$$x_{ijkl} \approx \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} z_{lr}$$

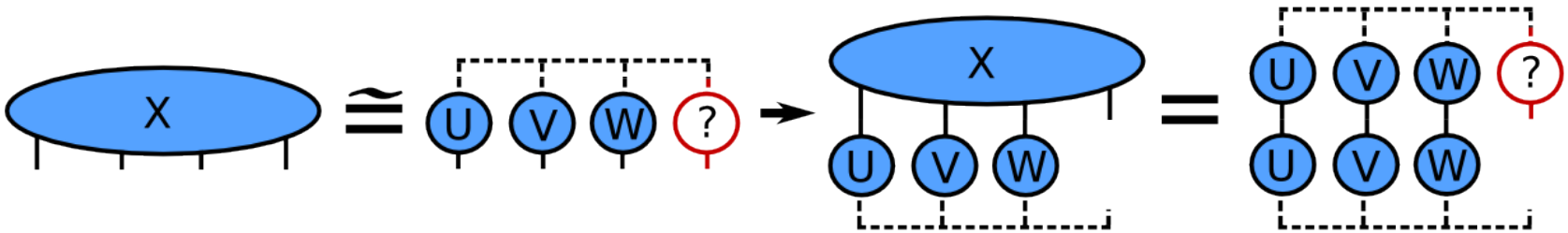


ALS updates factor matrices in an alternating manner

$$\min_{\boldsymbol{A}^{(n)}} f(\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}) = \frac{1}{2} ||\boldsymbol{\mathcal{X}} - [\![\boldsymbol{A}^{(1)}, \cdots, \boldsymbol{A}^{(n)}, \cdots, \boldsymbol{A}^{(N)}]\!]||_F^2,$$

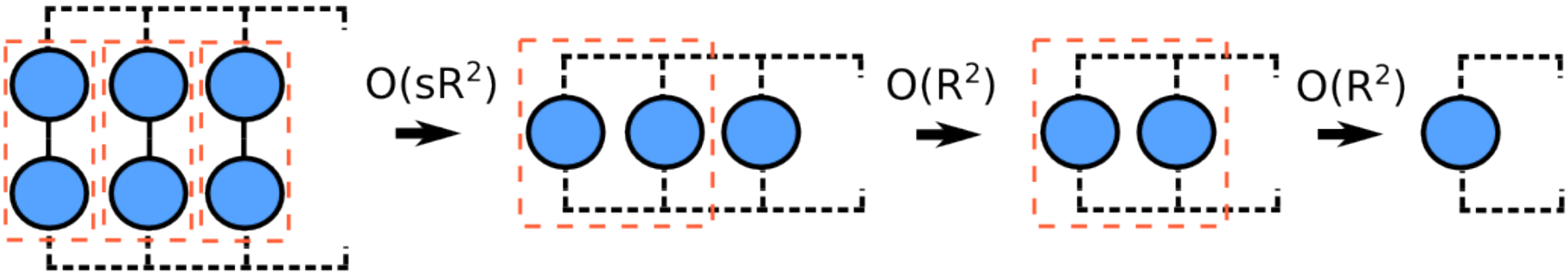


Each quadratic subproblem is typically solved via normal equations



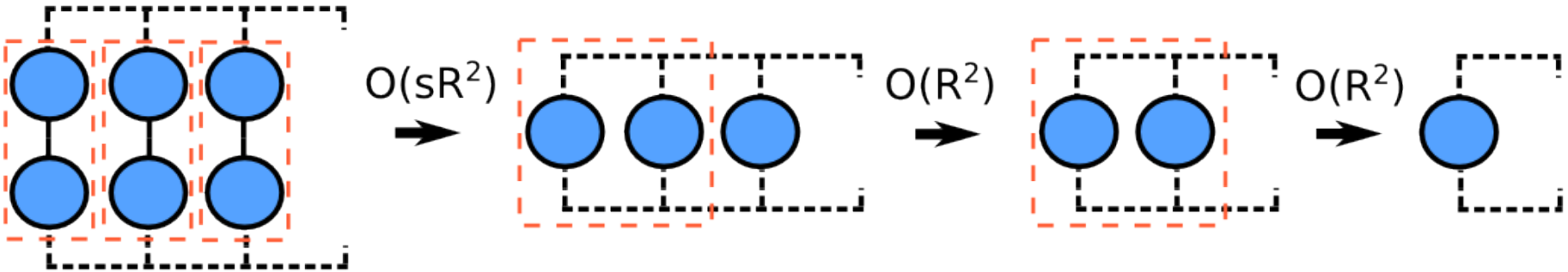
Tensor Contractions in CP ALS

The normal equations are cheap to compute

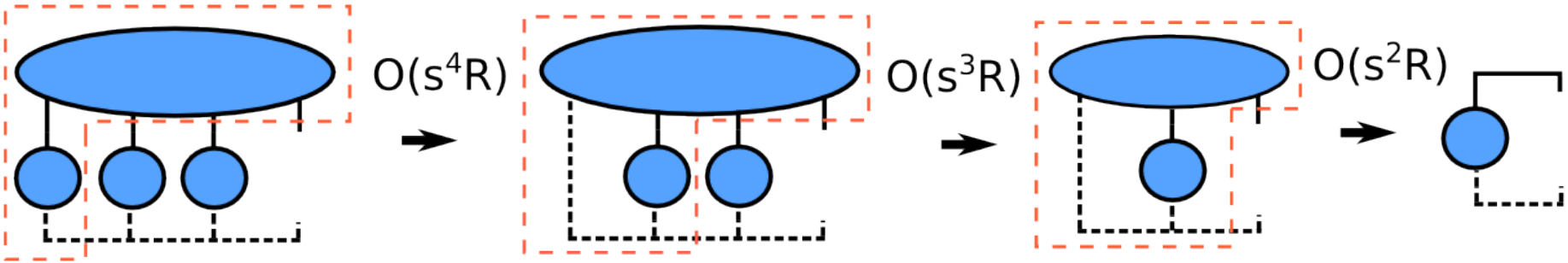


Tensor Contractions in CP ALS

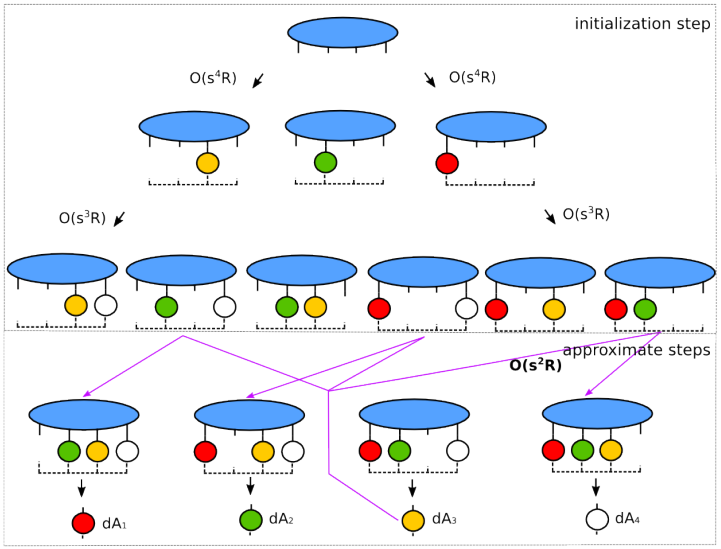
The normal equations are cheap to compute

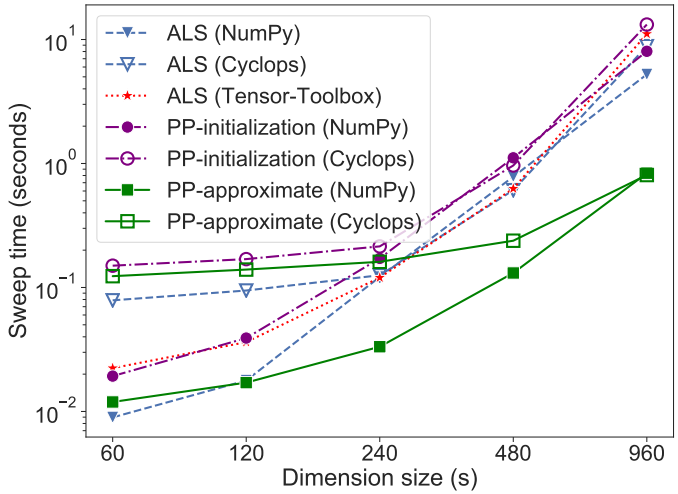


But forming the right-hand sides ($\boldsymbol{M}^{(n)}$) requires expensive MTTKRP (matricized tensor-times Khatri-Rao product)



Pairwise Perturbation Algorithm





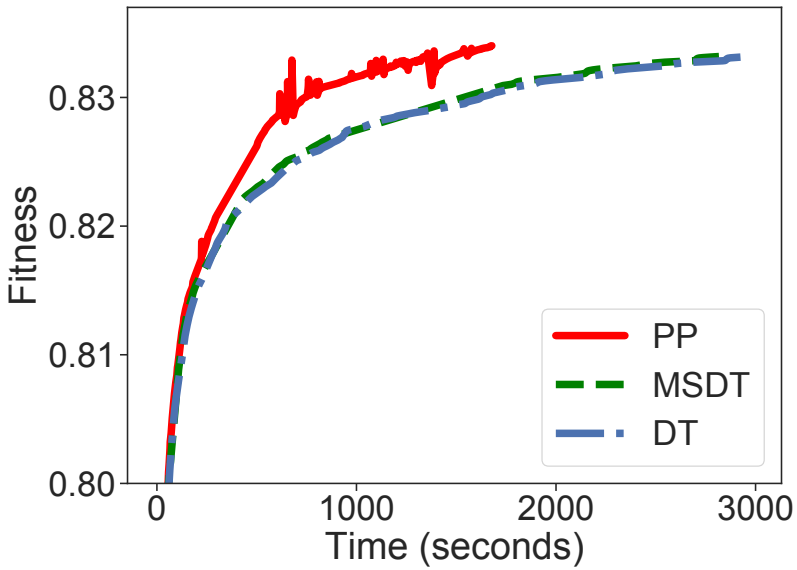
New algorithm: pairwise perturbation (PP)[1] approximates ALS

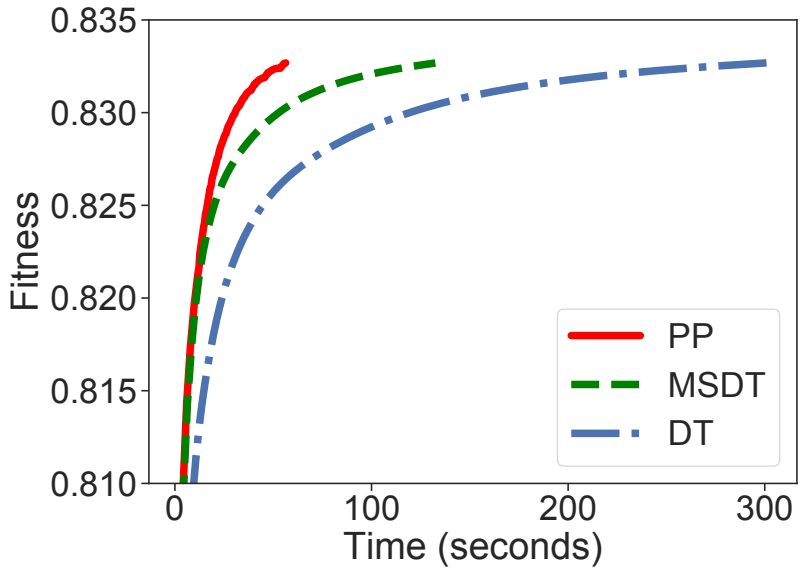
- based on perturbative expansion of ALS update to approximate MTTKRP
- approximation is accurate when ALS updates stagnate
- rank $R < s^{N-1}$ CP decomposition:
  - ALS sweep cost $O(s^N R) \Rightarrow O(s^2 R)$, up to 33x speed-up

Linjian Ma

[1] Linjian Ma, E.S. arXiv:1811.10573

Parallel Pairwise Perturbation Algorithm





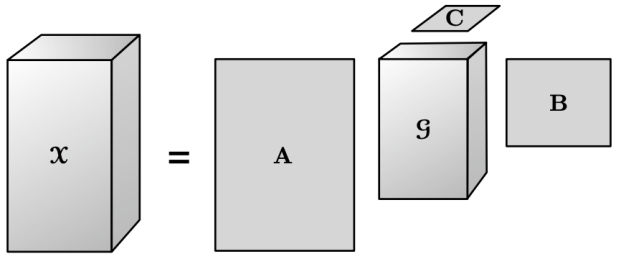
Effective parallelization by decomposing MTTKRP into local MTTKRPs [1]

$$\boldsymbol{U} = \text{MTTKRP}(\boldsymbol{\mathcal{T}}, \boldsymbol{V}, \boldsymbol{W}) \Rightarrow \boldsymbol{U}_i = \sum_{j,k} \text{MTTKRP}(\boldsymbol{\mathcal{T}}_{ijk}, \boldsymbol{V}_j, \boldsymbol{W}_k)$$

- processor (i, j, k) owns $\boldsymbol{\mathcal{T}}_{ijk}$, $\boldsymbol{V}_j$, and $\boldsymbol{W}_k$
- pairwise perturbation can be used to approximate local MTTKRPs
- multi-sweep dimension-tree (MSDT) amortizes terms across sweeps

[1] Linjian Ma, E.S. IPDPS 2021

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] Kolda and Bader, SIAM Review 2009

Randomized Methods for Sparse Tensor Decomposition

- When seeking a low-rank $R = O(1)$ decomposition for a sparse tensor, sketching schemes have been shown to be efficient
- In this regime, Tucker can be used to construct a CP decomposition
- Leverage score sampling on the rank-constrained least squares problem $\min_{\boldsymbol{X},\mathrm{rank}(\boldsymbol{X})\leq R} \|\boldsymbol{AX}-\boldsymbol{B}\|_F$ leads to a state-of-the-art cost-accuracy trade-off[1] to effectively approximate HOOI (ALS)

Algorithm for Tucker	LS solve cost	Sample size (m)
ALS	$O(\mathrm{nnz}(\boldsymbol{\mathcal{T}})R^{N-1})$	/
ALS + TensorSketch[2]	$\tilde{O}(mR^N + msR)$	$O(R^{2(N-1)} \cdot 3^{N-1}/(\epsilon^2\delta))$
ALS + TTMTS[2]	$\tilde{O}(msR^{N-1})$	$O(R^{2(N-1)} \cdot 3^{N-1}/(\epsilon^2\delta))$
ALS + TensorSketch[1]	$\tilde{O}(mR^{2N-2} + sR^{N-1})$	$O\big((R^{(N-1)} + 1/\epsilon^2) \cdot (3R)^{(N-1)}/\delta\big)$
ALS + leverage scores[1]	$\tilde{O}(mR^{2N-2} + sR^{N-1})$	$O(R^{(N-1)}/(\epsilon^2\delta))$

[1] Linjian Ma and E.S., arXiv:2104.01101

[2] O. Malik and S. Becker, 2018 (assuming unconstrained LSQ)

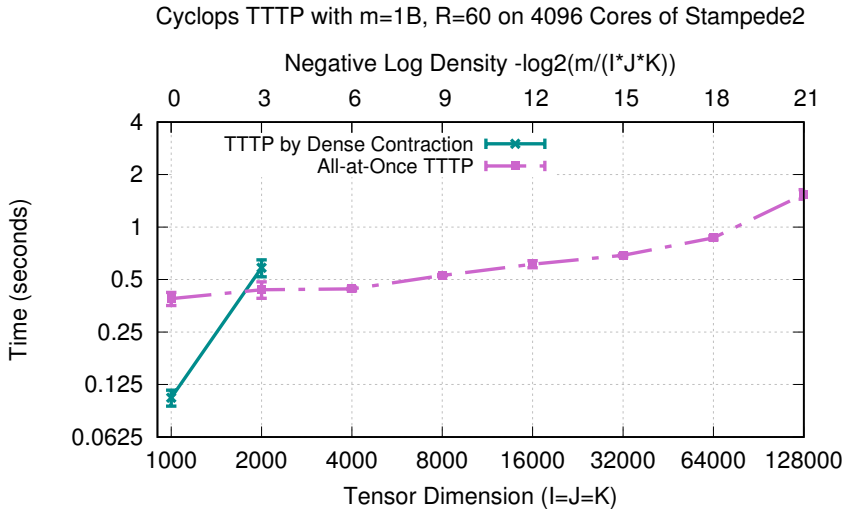
Tensor Completion

- 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] Navjot Singh, Zecheng Zhang, Xiaoxiao Wu, Naijing Zhang, Siyuan Zhang, and Edgar Solomonik arXiv:1910.02371

[2] Shaden Smith, Jongsoo Park, and George Karypis, 2016

Hamiltonians as Tensor Network Operators

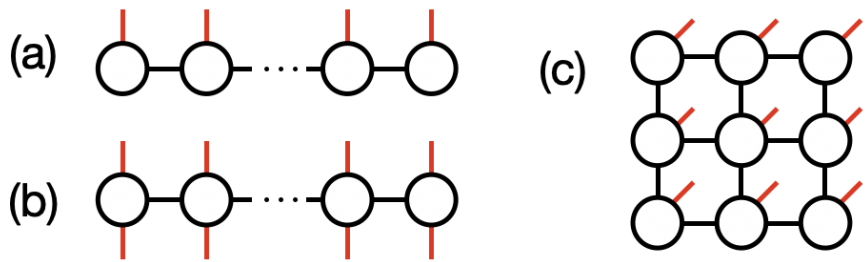
- Tensor network methods use tensor decompositions to represent quantum systems
- These methods are most natural for lattice spin systems such as the Heisenberg model and the simpler transverse field Ising model

$$H = \sum_{\langle i\, j\rangle} J^z Z_i Z_j + \sum_i h_x X_i$$

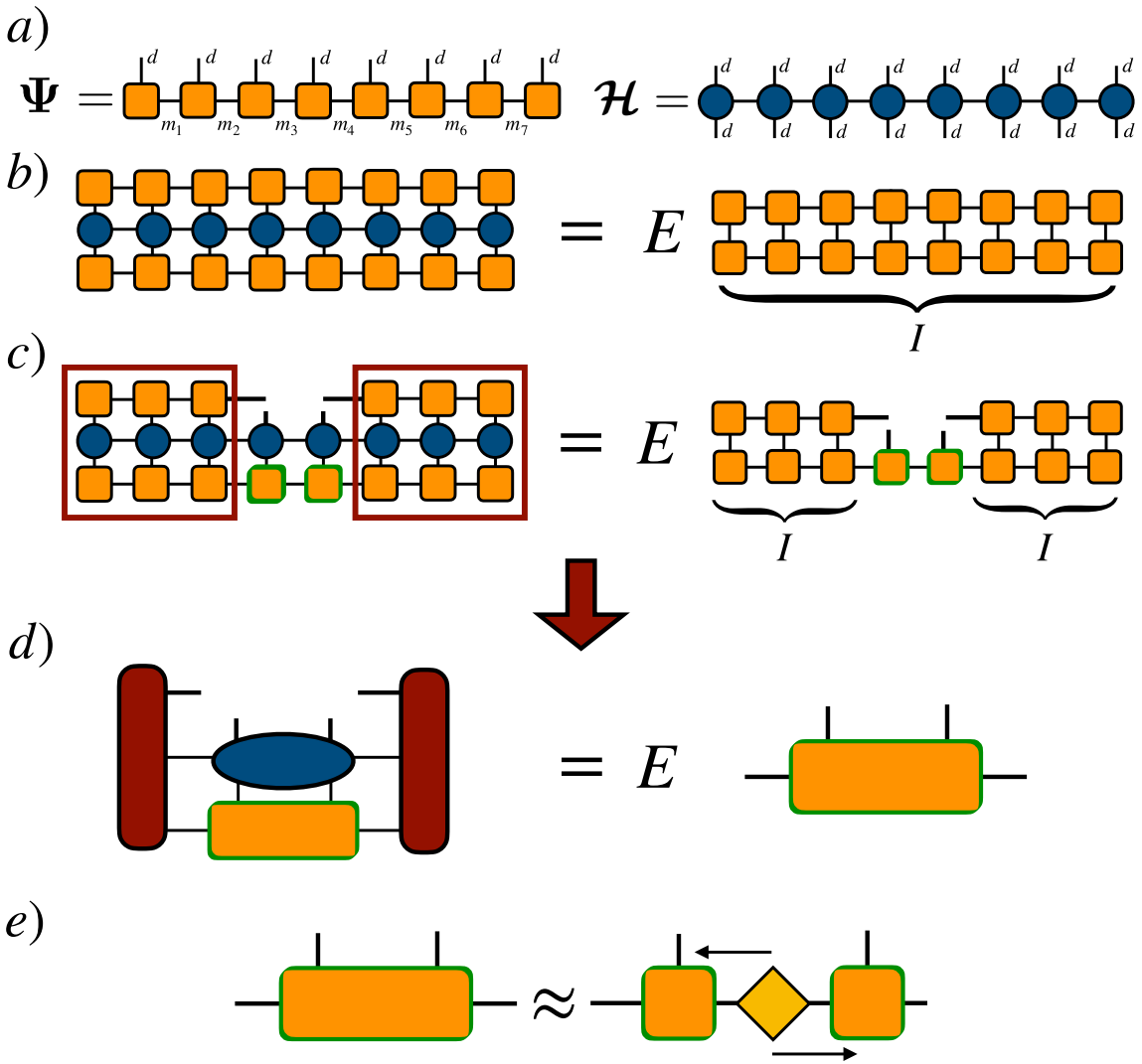
  where $\langle i\, j\rangle$ denote neighboring sites on a 2D lattice
- In the 1D case, 2-qubit operators such as $Z_i Z_{i+1}$ can be written as

$$\boldsymbol{H} = \boldsymbol{Z} \otimes \boldsymbol{Z} \otimes \boldsymbol{I} \otimes \cdots \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \boldsymbol{Z} \otimes \boldsymbol{Z} \otimes \boldsymbol{I} \otimes \cdots \otimes \boldsymbol{I} + \cdots$$

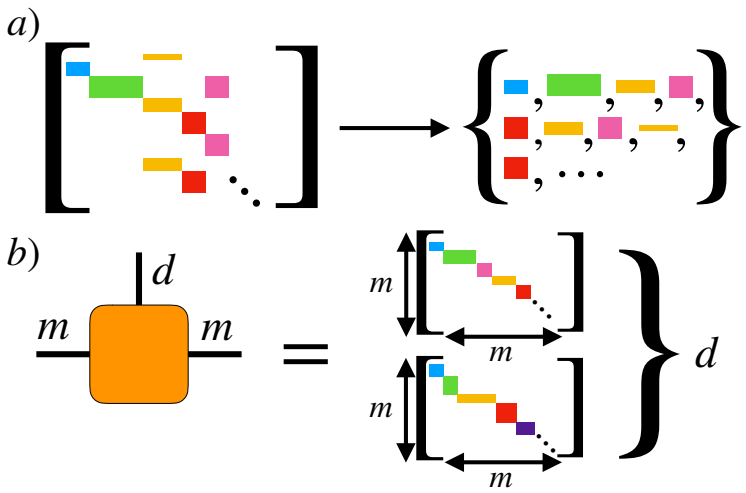
- In the 1D case, H can be represented as a matrix-product operator (MPO) with constant *bond dimension* (rank)

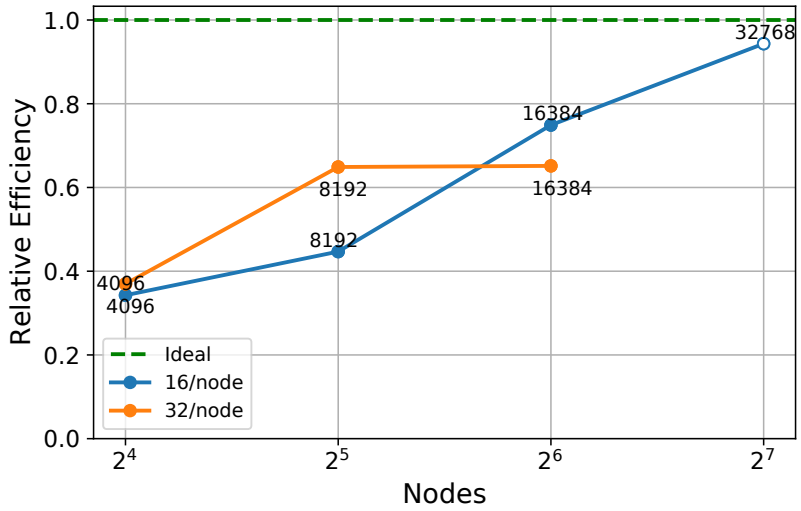


Density Matrix Renormalization Group (DMRG)



Parallel DMRG with Group Symmetry





We have recently developed a parallel DMRG code using Cyclops[1]

- compare two approaches to group symmetry
  - iterate over block-wise contractions
  - use CTF's sparse tensor representation
- match ITensor efficiency at scale for spin-system, but significantly lower efficiency for fermionic system with large number of blocks

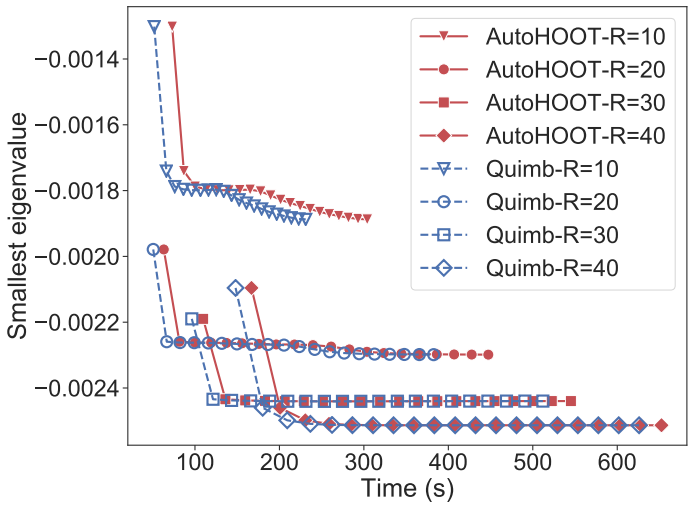
[1] R. Levy, B. Clark, E.S. arXiv:2007.05540

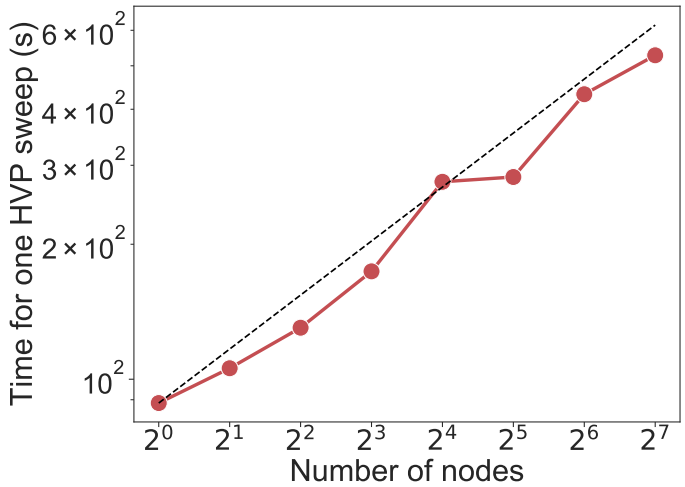
Automatic Generation of Tensor Network Methods

- 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

AutoHOOT: 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, 2020

Tensor Network State Evolution

- We can evolve a tensor network state by Trotterization of a Hamiltonian with m local terms

$$e^{-iH\tau} = \prod_{j=1}^{m} e^{-iH_j\tau} + O(\tau^2)$$

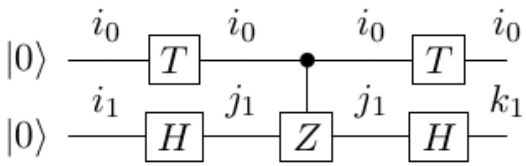
- Dynamics may be simulated by time-evolution $|\psi^{t+\tau}\rangle = e^{-iH\tau}|\psi^t\rangle$
- Ground state calculation can be done via imaginary time evolution, $|\psi^{i(t+\tau)}\rangle = e^{-H\tau}|\psi^{it}\rangle$, maximizing as follows

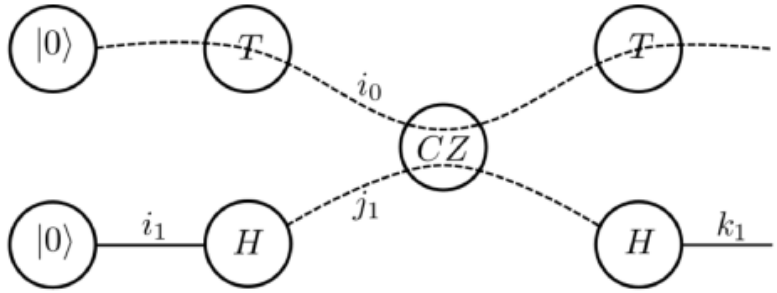
$$e^{-E\tau} = \max_{\|\psi\|_2} \langle\psi|e^{-H\tau}|\psi\rangle$$

  which is equivalent to minimizing E and leads to the same maximizer/minimizer ψ
- If H_j is a local (e.g., one/two-site) operator, so is $e^{-iH_j\tau}$

Quantum Circuit Simulation with Tensor Networks

- Evolution of tensor network states can also simulate quantum circuits
- In fact, a quantum circuit is a direct description of a tensor network[1]





- Why use HPC to (approximately) simulate quantum circuits?
  - enable development/testing/tuning of larger quantum circuits
  - understand approximability of different quantum algorithms
  - quantify sensitivity of algorithms to noise/error
  - potentially enable new hybrid quantum-classical algorithms
- Tensor network states (e.g., MPS/PEPS) provide robust tools for approximate simulation

[1] Markov and Shi SIAM JC 2007

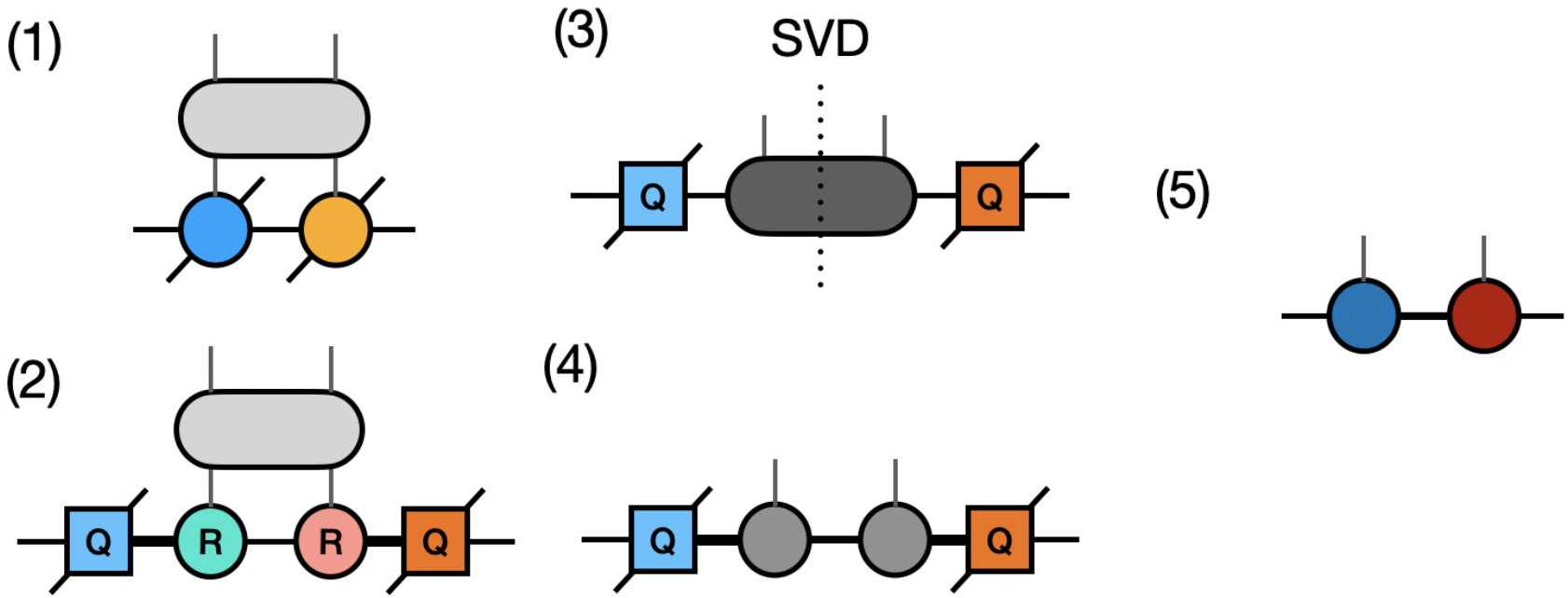
Quantum Circuit Simulation using PEPS[1]

- Near-term quantum architectures mostly connect qubits in a 2D fashion
- Non-local gates can be applied via the use of swap gates (with corresponding overhead)
- 2D tensor networks (projected entangles pair states (PEPS)) provide a natural way to simulate 2D quantum circuits
- Same software/algorithms infrastructure is also effective for (imaginary) time evolution with many Hamiltonians of interest
- Gate application and contraction of PEPS can both have exponential cost in the size of the circuit, so desire effective approximation

[1] Yuchen Pang, Tianyi Hao, Annika Dugad, Yiqing Zhou, and E.S., arXiv:2006.15234.

Approximate Application of Two-Site Operators

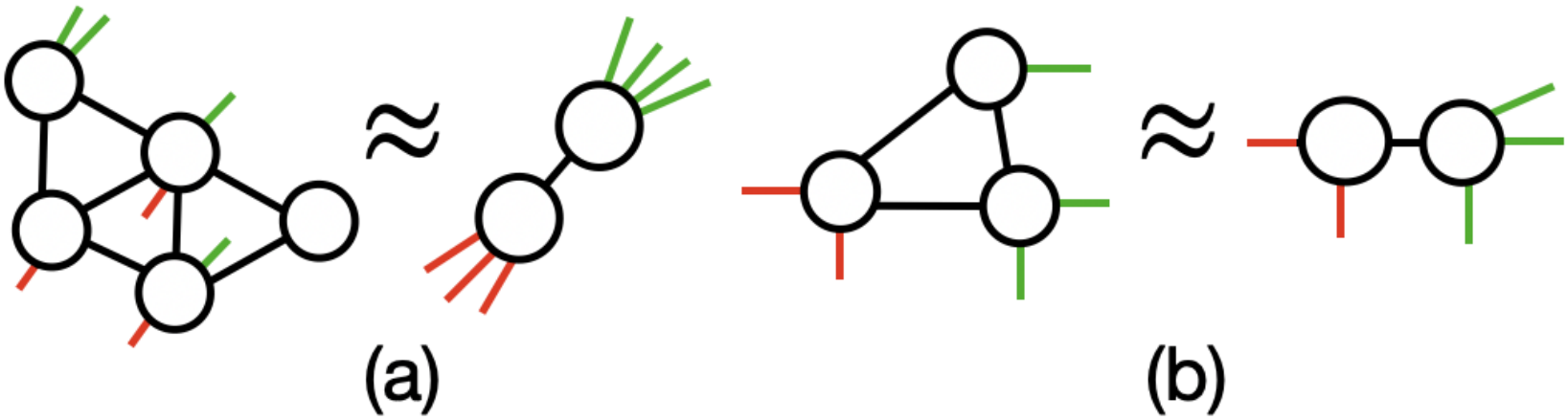
- Consider application of a two-site operator on neighboring PEPS sites
- *Simple update (QR-SVD)* algorithm:



- We provide an efficient distributed implementation of QR-SVD
- This operation is an instance of what we'll refer to as `einsumsvd` and QR-SVD is one algorithm/implementation

Implicit Randomized `einsumsvd`

- The `einsumsvd` primitive will also enable effective algorithms for PEPS contraction

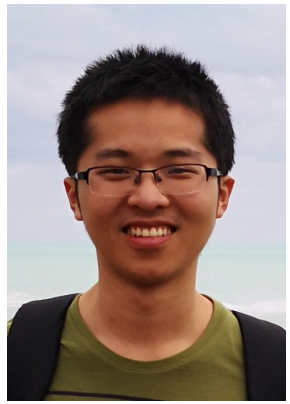


- An efficient general implementation is to leverage randomized SVD / orthogonal iteration, which iteratively computes a low-rank SVD by a matrix–matrix product that can be done implicitly via tensor contractions
- This approach improves complexity for PEPS boundary contraction method

Koala

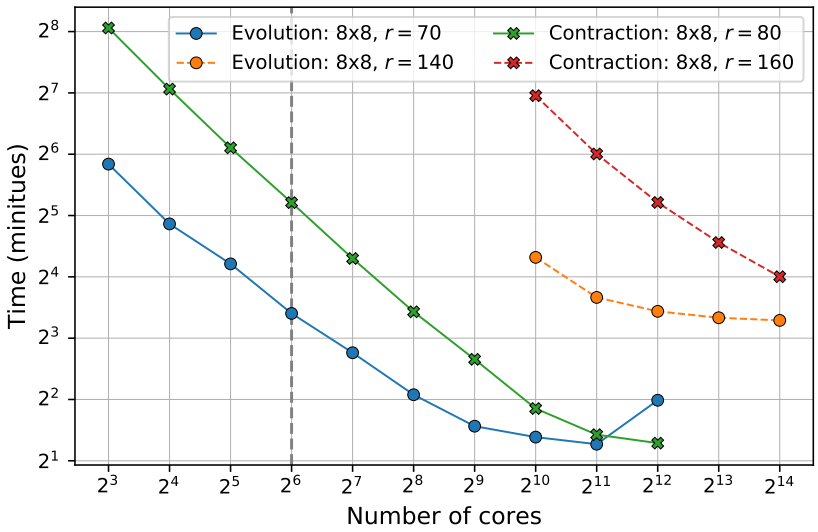
- We introduce a new library, Koala[1], for high-performance simulation of quantum circuits and time evolution with PEPS

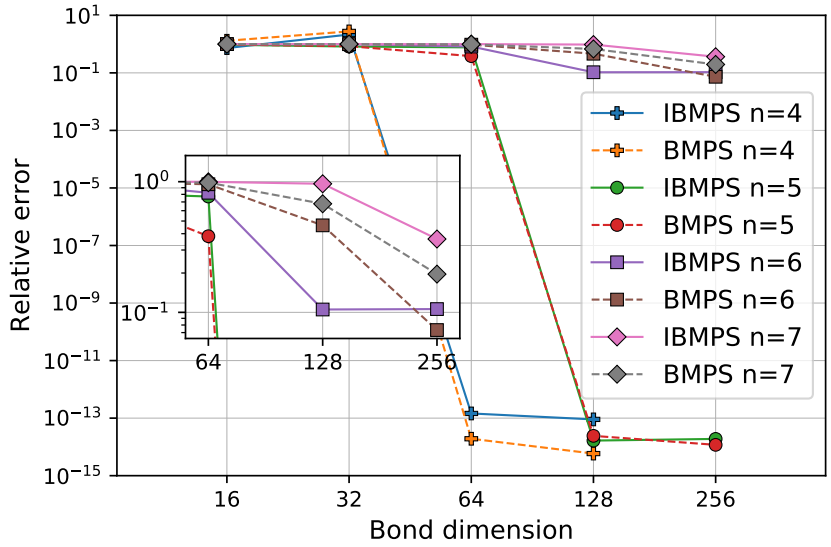
```python
from koala import peps, Observable
from tensorbackends.interface import ImplicitRandomizedSVD

# Create a 2-by-3 PEPS in distributed memory using CTF
qstate = peps.computational_zeros(nrow=2, ncol=3, backend='ctf')

# Construct operators and apply them to the quantum state
Y = qstate.backend.astensor([0,-1j,1j,0]).reshape(2,2)
CX = qstate.backend.astensor([1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,])
CX = CX.reshape(2,2,2,2)

qstate.apply_operator(Y, [1])
qstate.apply_operator(CX, [1,4], update_option=peps.QRUpdate(rank=2))

# Construct an observable and calculate the expectation with IBMPS
observable = Observable.ZZ(3, 4) + 0.2 * Observable.X(1)
result = qstate.expectation(
    observable, use_cache=True,
    contract_option=peps.BMPS(ImplicitRandomizedSVD(rank=4)),
)
```



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

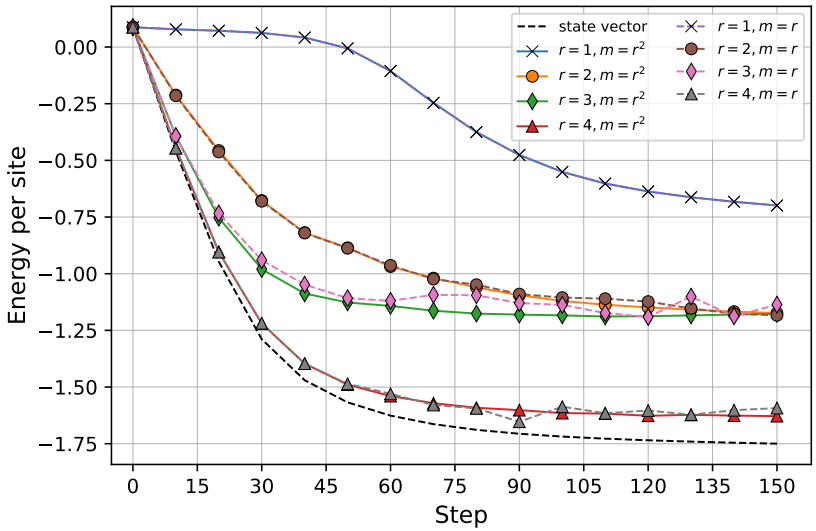
PEPS Benchmark Performance

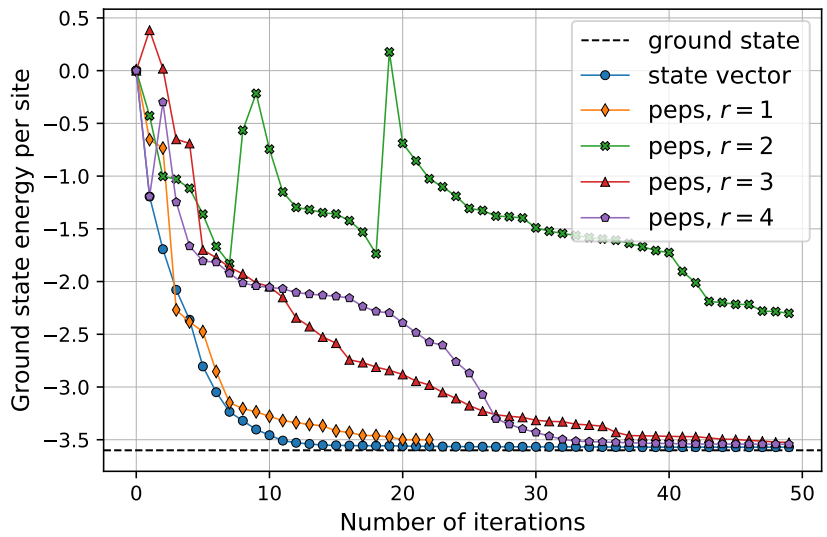




- Koala achieves good parallel scalability for approximate gate application (evolution) and contraction
- Approximation can be effective even for adversarially-designed circuits such as Google's random quantum circuit model (figure on right)

PEPS Accuracy for Quantum Simulation





- ITE code achieves improvable accuracy with increased PEPS bond dimension, but approximation in PEPS contraction is not variational
- Variational quantum eigensolver (VQE), which represents a wavefunction using a parameterized circuit $U(\theta)$ and minimizes

$$\langle U(\theta)| H |U(\theta)\rangle ,$$

also achieves improvable accuracy with higher PEPS bond dimension

Conclusion

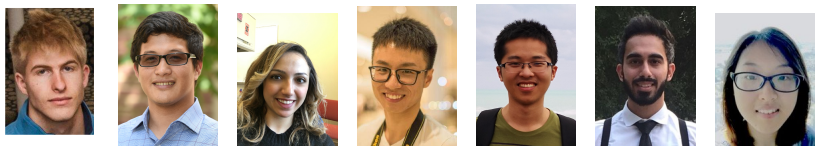
- We have presented innovations to numerical algorithms and software libraries for tensor contractions, decompositions, and tensor networks
- Our research group is developing an ecosystem of algorithms and software for simulation of quantum systems
- This work is relevant to both classical methods for quantum chemistry and physics, as well as quantum computation
- See our website[1] and github pages for access to high performance software for tensor methods

[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`