BERKELEY LAB

U.S. DEPARTMENT OF ENERGY

Office of Science

Exploiting Sparsity and Randomness to Accelerate Linear Least-Squares at Scale

Vivek Bharadwaj, Qualifying Examination

Exam Committee: Katherine Yelick (Chair), James Demmel, Aydın Buluç, Michael Lindsey

Date & Time: January 17, 2024, 12pm PST

Location: Room 510, Soda Hall

Introduction

Bedrock: Dense Linear Algebra

- Many algorithms in machine learning and scientific computing rely on linear algebraic (LA) primitives.
- BLAS [Law+79] & LAPACK [And+92]: Enabled non-specialists to use optimized dense LA kernels.
- SCALAPACK [Bla+97], MAGMA [TDB10], SLATE [Gat+20] & many others: Extended capabilities to massively parallel architectures and accelerators.

Figure 1: Frontier, the first US exascale machine. Credit: OLCF at ORNL, Wikimedia Commons CC2.0.

Sparsity Introduces New Challenges

Consider a linear least-squares problem of the form $\min_X \|AX - B\|_F$. What happens if:

- A or B has **structural sparsity** (most entries are zero)?
- A is **data-sparse** (written as a Kronecker product, or related structure)?
- B is only available *on-demand*, and we must **induce sparsity** to reduce the sampling complexity?

Dense LA methods are often intractable! Need tailored algorithms and custom computational kernels to achieve high performance.

Randomization in Numerical Linear Algebra

- Randomized computing has a long history. Randomized linear algebra has seen significant progress since 2008 (e.g. [RT08], Blendenpik [AMT10]) [Woo14].

- Key ingredient: **sketching**. Example for overdetermined LSTSQ: apply random matrix S with far fewer rows than columns, solve cheaper problem

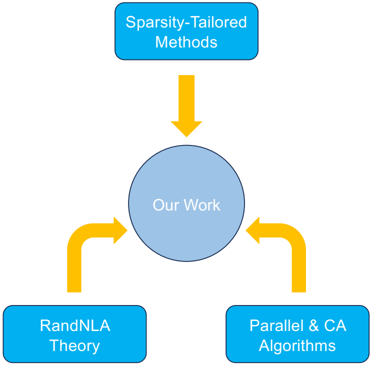
$$\min_X \|SAX - SB\|_F$$

- Efforts are underway to standardize randomized linear algebra primitives [Mur+23]. Intersection of sparsity and RandNLA: open frontier.

High-Level Contributions of this Work

Our goal: accelerate sparse LSTSQ problems using randomization and communication avoiding (CA) parallel algorithms. We will:

- **Identify** applications involving sparse linear least-squares problems.
- **Design** novel randomized techniques and parallelization strategies tailored to that sparsity.
- **Deploy** our methods on parallel architectures at scale.



Topics We Will Cover

1. Sketching linear least-squares problems in sparse Candecomp / PARAFAC (Neurips'23; [**Bh**MMGBD23]).

2. Distributed-memory randomized CP methods (Preprint, Arxiv; [**Bh**MMBD23]).

3. CA-algorithms for sparse matrix *completion* (IPDPS'22; [**Bh**BuDe22]).

4. Future work: sketching to accelerate the marginalized graph kernel, emerging extensions of sketching for tensor trains.

Fast Exact Leverage Score Sampling from Khatri-Rao Products

The Khatri-Rao Product

- The Khatri-Rao product (KRP, denoted $\odot$) is the column-wise Kronecker product of two matrices:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \odot \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw & bx \\ cw & dx \\ ay & bz \\ cy & dz \end{bmatrix}$$

- Our goal: efficiently solve an overdetermined linear least-squares problem

$$\min_X \|AX - B\|_F$$

where $A = U_1 \odot ... \odot U_N$ with $U_j \in \mathbb{R}^{I_j \times R}$.

Motivating Application

This least-squares problem is the computational bottleneck in alternating least-squares Candecomp / PARAFAC (CP) decomposition [KB09].

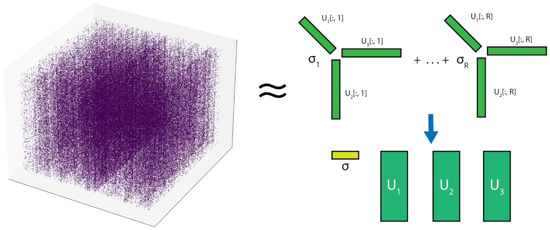


Figure 2: Subregion of Amazon sparse tensor and illustrated CP decomposition.

Focus on large sparse tensors (mode sizes in the millions) and moderate decomposition rank $R \approx 10^2$. Assume $I_j = I$ for all j and $I \geq R$.

Randomized Linear Least-Squares

- **Sketch & Solve**: Apply short-wide sketching matrix S to both A and B, solve reduced problem

$$\min_{\tilde{X}} \left\| S A \tilde{X} - S B \right\|_F$$

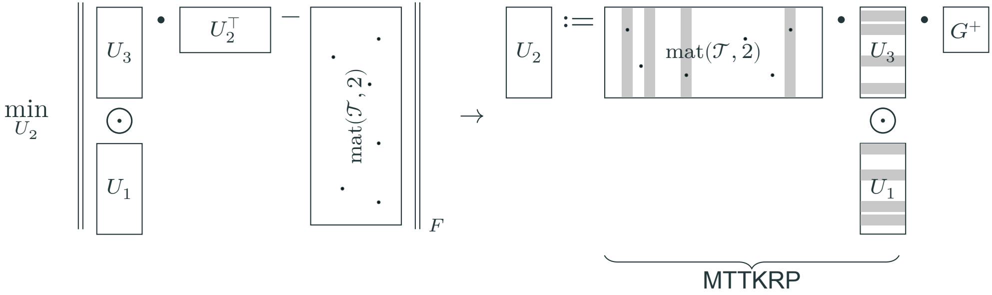
- Want an (ε, δ) guarantee on solution quality: with high probability $(1 - \delta)$,

$$\left\| A \tilde{X} - B \right\|_F \leq (1 + \varepsilon) \min_X \| AX - B \|$$

- Restrict S to be a *sampling* matrix: selects and reweights rows from A and B. **How do we downsample a Khatri-Rao product accurately AND efficently?**

Effect of Sampling Operator, Sparse Tensor Decomposition

$$\min_{U_j} \left\| \left[\bigodot_{k \neq j} U_k \right] \cdot U_j^\top - \mathrm{mat}(\mathcal{T}, j)^\top \right\|_F$$



Our Contributions [BhMMGBD23]

Method	Source	Round Complexity ($\tilde{O}$ notation)
CP-ALS	[KB09]	$N(N+I)I^{N-1}R$
CP-ARLS-LEV	[LK22]	$N(R+I)R^N/(\varepsilon\delta)$
TNS-CP	[Mal22]	$N^3IR^3/(\varepsilon\delta)$
GTNE	[MS22]	$N^2(N^{1.5}R^{3.5}/\varepsilon^3+IR^2)/\varepsilon^2$
STS-CP	Ours	$N(NR^3\log I+IR^2)/(\varepsilon\delta)$

- We build a data structure with runtime **logarithmic** in the height of the KRP and quadratic in R to sample from *leverage scores* of A.
- Yields the **STS-CP** algorithm: lower asymptotic runtime for randomized dense CP decomposition than recent SOTA methods (and even greater advantages for sparse tensors).

Leverage Score Sampling

We will sample rows i.i.d. from A according to the *leverage score distribution* on its rows. Given **reduced SVD** $A = U\Sigma V^\top$, the leverage score ℓ_i of row i is

$$\ell_i = \|U[i,:]\|^2 .$$

Theorem (Leverage Score Sampling Guarantees, [Mal22])

Suppose $S \in \mathbb{R}^{J\times I}$ is a leverage-score sampling matrix for $A \in \mathbb{R}^{I\times R}$, and define

$$\tilde{X} := \arg\min_{\tilde{X}} \left\|SA\tilde{X} - SB\right\|_F$$

If $J \gtrsim R\max(\log(R/\delta), 1/(\varepsilon\delta))$, then with probability at least $1-\delta$,

$$\left\|A\tilde{X} - B\right\|_F \leq (1+\varepsilon)\min_X \|AX - B\|_F .$$

Interpretation of Leverage Scores

When A has 1 column, leverage scores are proportional to squared distance from origin.

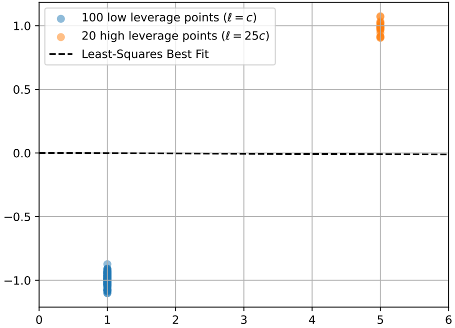
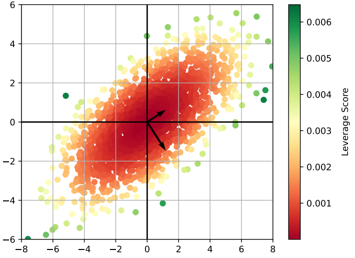


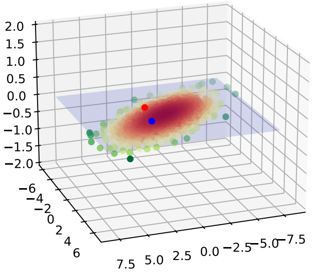
Figure 3: A univariate regression problem with low and high leverage points (intercept constrained to be 0).

Interpretation of Leverage Scores

In general, leverage scores of A quantify *influence* that each row has on the solution, capture correlation of rows of A with rows of $\Sigma^{-1}V^\top$.



(a) Projection onto xy-plane



(b) (x, y, z) data

Figure 4: Leverage scores of $(x, y, 0)$ triples from a multivariate normal distribution. Left: components of $\Sigma^{-1}V^\top$ shown. Right: the red point has greater influence than the blue point (both equidistant from $(0, 0)$).

Interpretation of Leverage Scores

- Leverage score sampling captures the geometry of the column space of A.
- Rigorously: sampling i.i.d. with leverage score probabilities leads to an **optimal** [DM20] sample complexity to construct an ℓ_2-subspace embedding matrix S. W.h.p simultaneously for ALL vectors $x \in \mathbb{R}^R$,

$$(1-\tilde{\varepsilon})\left\|Ax\right\|_2 \leq \left\|SAx\right\|_2 \leq (1+\tilde{\varepsilon})\left\|Ax\right\|_2$$

- In turn, an ℓ_2-S.E. guarantees that our sketched solution has close-to-optimal residual with respect to the original problem.

Prior Work

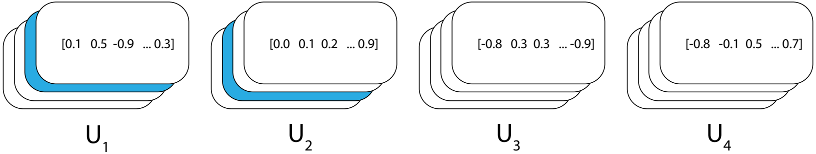
Problem: Cost to compute all leverage scores exactly is identical to runtime of QR decomposition. Defeats the purpose of sampling!

- (SPALS [Che+16]): Sample rows according to *approximate* leverage scores of A. Worst-case **exponential** in N to achieve (ε, δ) guarantee.
- (CP-ARLS-LEV [LK22]): Similar approximation, hybrid random-deterministic sampling strategy and practical improvements.
- (TNS-CP [Mal22]): Samples implicitly from exact leverage distribution with **polynomial** complexity to achieve (ε, δ) guarantee, but linear dependence on I for each sample. **We want to accelerate this algorithm.**

Implicit Leverage Score Sampling

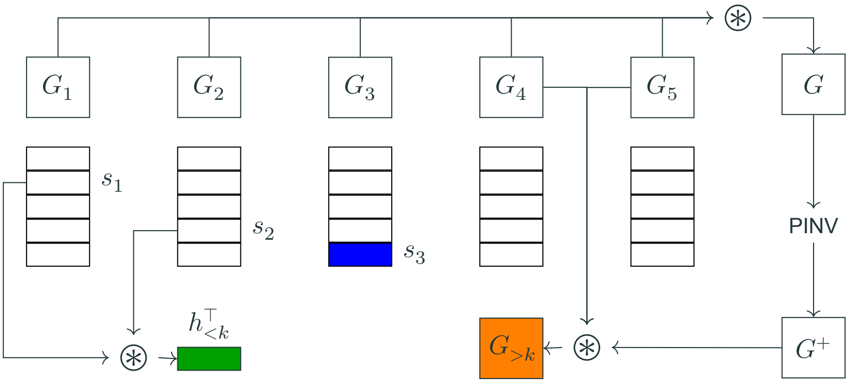
- For $I = 10^7, N = 3$, matrix A has 10^{21} rows. Can't even index rows with 64-bit integers. Instead: use identity $\ell_i = A[i,:] \, (A^\top A)^+ A[i,:]^\top$.

- Draw a row from each of $U_1, ..., U_N$, return their Hadamard product.



- Let $\hat{s}_j$ be a random variable for the row index drawn from U_j. Assume $(\hat{s}_1, ..., \hat{s}_N)$ jointly follows the leverage score distribution on $U_1 \odot ... \odot U_N$.

The Conditional Distribution of $\hat{s}_k$



Theorem

$$p(\hat{s}_k = s_k \mid \hat{s}_{<k} = s_{<k}) \propto \langle h_{<k} h_{<k}^\top, U_k[s_k, :]^\top U_k[s_k, :], G_{>k} \rangle$$

Key Sampling Primitive

$$q[i] := C^{-1}\langle h_{<k} h_{<k}^\top, U_k[i,:]^\top U_k[i,:], G_{>k}\rangle$$

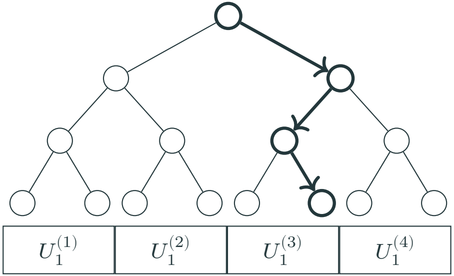
- Can't compute q entirely - would cost $O(IR^2)$ per sample per mode.
- Imagine we magically had all entries of q - how to sample? Initialize I bins, j'th has width $q[j]$.
- Choose random real r in $[0,1]$, find "containing bin" i such that

$$\sum_{j=0}^{i-1} q[j] < r < \sum_{j=0}^{i} q[j]$$

Binary Tree Inversion Sampling

- Locate bin via binary search (truncated to $\log(I/R)$ levels)
- Root: branch right iff $\sum_{j=0}^{I/2} q[j] < r$
- Level 2: branch right iff

$$\sum_{j=0}^{I/2} q[j] + \sum_{j=I/2}^{3I/4} q[j] < r$$



Key: Can compute summations quickly if we cache information at each node!

Caching Partial Gram Matrices

Let an internal node v correspond to an interval of rows $[S(v)...E(v)]$.

$$
\begin{aligned}
\sum_{j=S(v)}^{E(v)} q[j] &= \sum_{j=S(v)}^{E(v)} C^{-1} \langle h_{<k} h_{<k}^\top, U_k[j,:]^\top U_k[j,:], G_{>k} \rangle \\
&= C^{-1} \langle h_{<k} h_{<k}^\top, \sum_{j=S(v)}^{E(v)} U_k[j,:]^\top U_k[j,:], G_{>k} \rangle \\
&= C^{-1} \langle h_{<k} h_{<k}^\top, U_k[S(v):E(v),:]^\top U_k[S(v):E(v),:], G_{>k} \rangle \\
&:= C^{-1} \langle h_{<k} h_{<k}^\top, G^v, G_{>k} \rangle
\end{aligned}
\tag{1}
$$

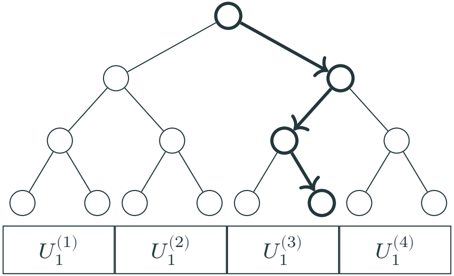
Can compute and store G^v for ALL nodes v in time $O(IR^2)$, storage space $O(IR)$. Use BLAS-3 **syrk** calls in parallel to efficiently construct the tree.

Efficient Sampling after Caching

- At internal nodes, compute $C^{-1}\langle h_{<k}h_{<k}^{\top}, G^{v}, G_{>k}\rangle$ in $O(R^2)$ time (read normalization constant from root)

- At leaves, spend $O(R^3)$ time to compute remaining values of q. Can reduce to $O(R^2 \log R)$, see paper.

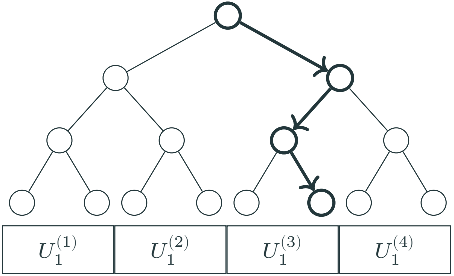
- Complexity per sample: $O(NR^2 \log I)$ (summed over all tensor modes).



High-Performance Parallel Sampling, Approach 1

- We want to execute $J \sim 50,000$ independent random walks down a full, complete tree. At each node, execute a matrix-vector multiplication to decide which direction to branch.

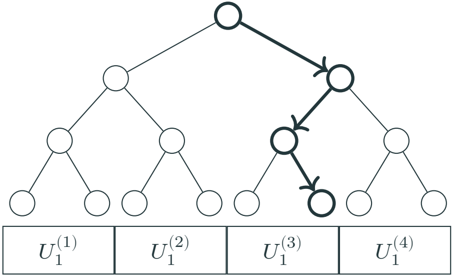
- **Approach 1:** Assign a thread team to execute random walks independently. **Proudly parallel, no data races.**



High-Performance Parallel Sampling, Approach 2

- Issues: irregular memory access pattern on CPU, not optimal for a single GPU thread to execute a BLAS call.

- **Approach 2:** March down the tree one level at a time, computing the branches of ALL random walks with a **batched GEMV / GEMM**.



Runtime Benchmarks (LBNL Perlmutter CPU)

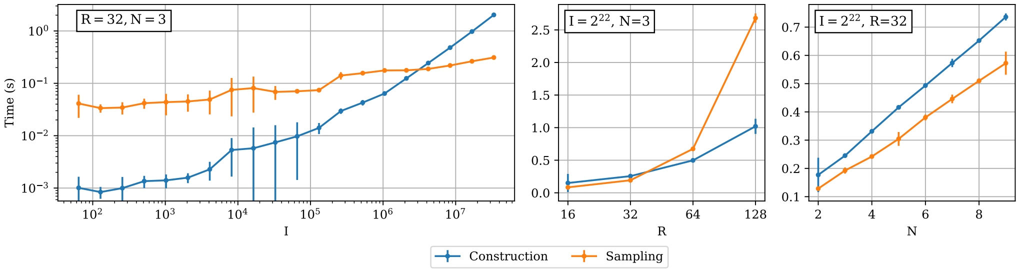


Figure 5: Time to construct sampler and draw $J = 65,536$ samples. C++ Implementation Linked to OpenBLAS. 1 Node, 128 OpenMP Threads, BLAS3 Construction, BLAS2 Sampling.

Distortion, Ours vs. Approximate Leverage Score Sampling

Define the distortion $D(S, A)$ of sketch S with respect to matrix A by

$$D(S, A) = \kappa(SQ) - 1$$

where Q is any orthonormal basis for the column space of A [Mur+23]. Distortion quantifies the distance preservation property of a sketch.

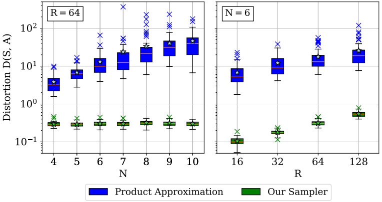


Figure 6: Sketch distortion as a function of KRP matrix count N and column count R, $J = 65,536$. Green: our sampler. Blue: product approximation by [LK22].

Accuracy Comparison for Fixed Sample Count

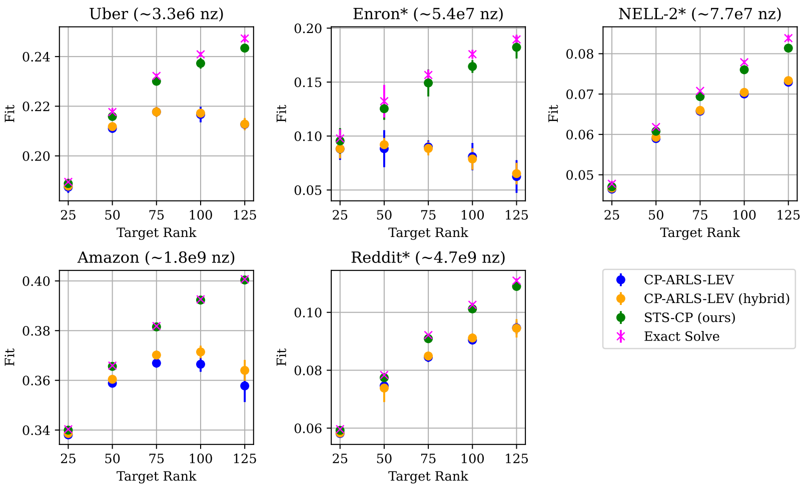


Figure 7: Sparse tensor ALS accuracy comparison for $J = 2^{16}$ samples, varied target ranks.

STS-CP Makes Faster Progress to Solution

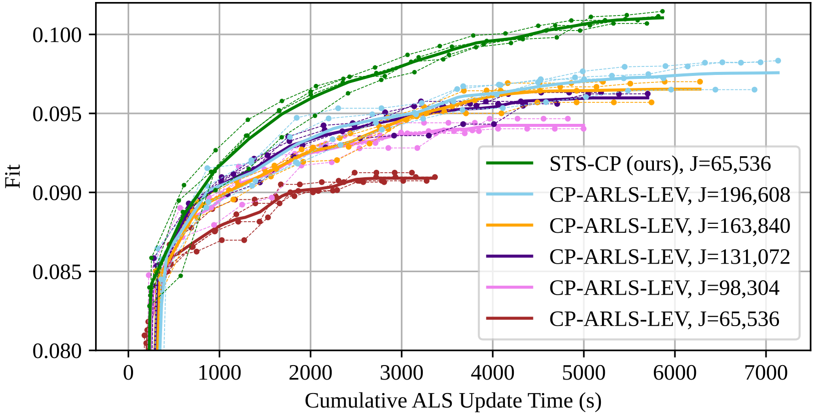


Figure 8: Fit vs. ALS update time, Reddit tensor, $R = 100$.

Takeaways: Faster Leverage Score Sampling from Khatri-Rao Products

- We accelerated sampling from the Khatri-Rao product by devising a novel data structure and a high-performance implementation.
- We demonstrated convincing speedups and accuracy benefits over CP-ARLS-LEV [LK22], an algorithm that approximates the leverage scores.
- **Up next**: distributed-memory formulations of both our algorithm and CP-ARLS-LEV.

High-Performance Randomized CP Decomposition at Scale

Sparse Tensors from FROSTT

Tensor	Dimensions	NNZ
Uber	$183 \times 24 \times 1.1K \times 1.7K$	$3.3M$
Amazon	$4.8M \times 1.8M \times 1.8M$	$1.7B$
Patents	$46 \times 239K \times 239K$	$3.6B$
Reddit	$8.2M \times 177K \times 8.1M$	$4.7B$

- Sparse tensors may have **billions** of nonzero entries, mode sizes in the **tens of millions** [Smi+17].

- Randomized algorithms okay in shared-memory, but **existing codes cannot compete** with classic distributed-memory implementations [Smi+15; Kan+12].

Our Contributions [BhMMBD23]

- We give high-performance implementations of STS-CP and CP-ARLS-LEV scaling to thousands of CPU cores.
- Up to 11x speedup over SPLATT.
- Several communication / computation optimizations unique to randomized CP decomposition.

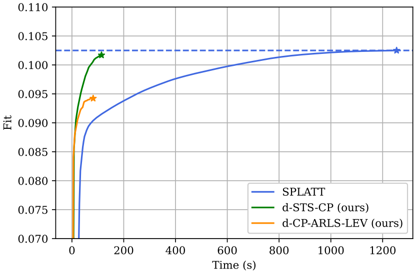


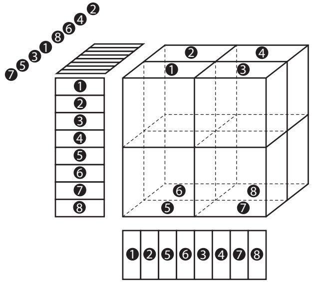
Figure 9: Accuracy vs. time, Reddit tensor, $R = 100$, 512 cores / 4 Perlmutter CPU nodes, 4.7 billion nonzeros.

Methodology

- We distribute STS-CP and CP-ARLS-LEV [LK22] with very distinct communication / computation patterns, each with varying time / accuracy tradeoffs.
- We tailor the communication schedule to randomized CP decomposition to eliminate Reduce-scatter collectives, achieving better load balance in the process.
- We use a hybrid of CSC format for nonzero lookups and CSR format to enable race-free thread parallelism. Key primitive: sparse transpose.

Factor and Sparse Matrix Layout

- Processors arranged in a hypercube.
- Factor matrices $U_1, ..., U_N$ distributed by block rows. Assume that all processors redundantly compute $U_j^\top U_j$ for all j (product is gram matrix G of A).
- Each processor owns a block of the sparse tensor. Randomly permute modes to load-balance.



Bulk-Synchronous Randomized ALS Update

1. **Sampling and All-gather:** Sample rows of $U_{\neq j}$, Allgather the rows to processors who require them.

2. **Local Computation:** Extract the corresponding nonzeros from the local tensor, execute the downsampled MTTKRP.

3. **Reduction and Postprocessing:** Reduce the accumulator of the sparse-dense matrix multiplication across processors, if necessary, and post-process the factor.

Distribution of Distinct Sampling Algorithms

Sampler	Compute	Messages	Words Communicated
d-CP-ARLS-LEV	JN/P	P	P
d-STS-CP	$JR^2 \log I/P$	$P \log P$	$JR \log P/P$

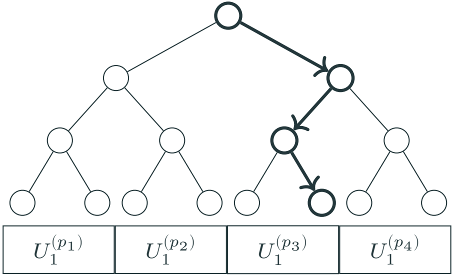
- CP-ARLS-LEV *approximates* the leverage scores with lower computation / communication overhead. Accuracy degrades at high rank.
- STS-CP samples from the *exact* leverage score distribution, requiring higher sampling time.
- **Problem:** How to sample when factors distributed by block rows?

d-CP-ARLS-LEV

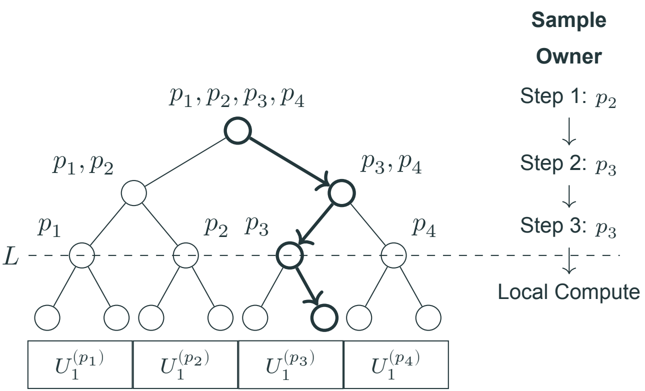
- **Key Idea:** Approximate the leverage scores of A by the *product* of leverage scores on each factor matrix U_i.
- Let $U_i^{(p_j)}$ be the block row of U_i owned by processor p_j. Leverages scores of this block given by
$$\mathrm{diag}\left(U_i^{(p_j)} G^{+} U_i^{(p_j)\top}\right)$$
- Computed locally on each processor without communication. Sampling requires (in expectation) only a small constant number of words communicated.
- **Drawback:** Accuracy degrades for high N or R.

d-STS-CP

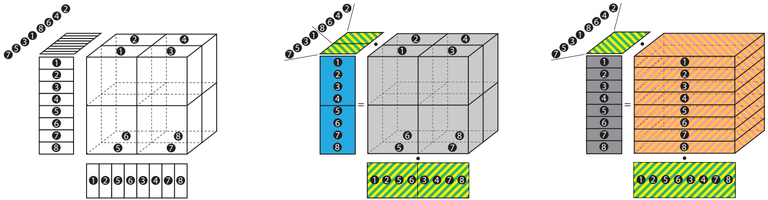
- Samples from **exact** leverage score distribution by sampling from each of $U_1, ..., U_N$ in sequence (excluding U_j).
- Execute random walk on binary tree to find the row index for each U_j. Node v caches "partial Gram matrix" G^v.
- At each node, compute $h^\top G^v h$ (where h is unique to each sample) to decide whether to branch left / right.



d-STS-CP Parallelization Scheme



Randomization-Tailored Communication Schedule

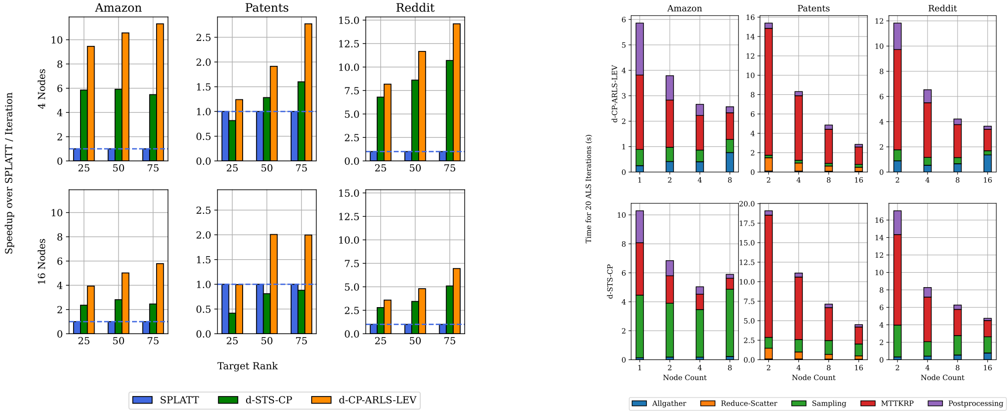


Initial Data Distribution | Tensor Stationary MTTKRP | Accumulator Stationary MTTKRP

All-gather Downsampled Matrix | Reduce-Scatter | Redistribute Downsampled Tensor | Stationary

Schedule	Words Communicated / Round
Non-Randomized TS	$2NR\left(\prod_{k=1}^{N} I_k/P\right)^{1/N}$
Sampled TS	$NR\left(\prod_{k=1}^{N} I_k/P\right)^{1/N}$
Sampled AS	$JRN(N-1)$

Speedup and Scaling on Large Sparse Tensors



(a) Speedup

(b) Strong Scaling, $R = 25$

Figure 10: Speedup over SPLATT and strong scaling for our randomized algorithms.

Comparison of Communication Schedules

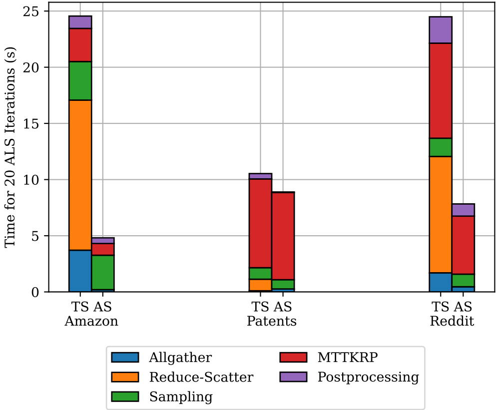


Figure 11: Runtime breakdown for tensor-stationary vs. accumulator-stationary communication schedules.

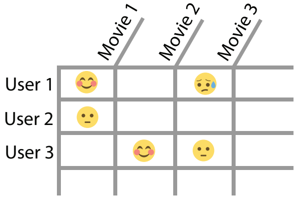
Takeaways: Distributed-Memory Randomized CP Algorithms

- We proposed the first distributed-memory implementation of two sampling-based, sparse CP algorithms.
- We optimized our algorithms to avoid communication in both the sampling and MTTKRP phases.
- Our method scales to thousands of CPU cores with significant speedups over existing SOTA sparse tensor decomposition software.

Communication-Avoiding Algorithms for Matrix Completion

Matrix / Tensor Completion vs. Factorization

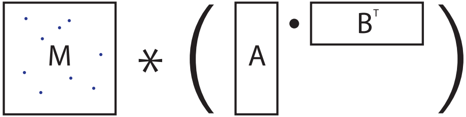
- **Sparse Tensor Completion:** Zero indicates *unobserved data*. Want to fit model only to observed entries, expect it to generalize. Distinct from factorization, where zeros are "true zeros".



- For simplicity, we will focus on the matrix case. Let $S \in \mathbb{R}^{m \times n}$ be the matrix we want to factor and $A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{n \times r}$ be embedding matrices.

Collaborative Filtering

- **Factorization:** Solve $\min_{A,B} \left\| S - AB^\top \right\|_F$.
- **Completion:** Solve $\min_{A,B} \left\| S - M \circledast (AB^\top) \right\|_F$, where M is a binary mask with the same sparsity pattern as S.
- $M \circledast (AB^\top)$ is called **sampled dense-dense matrix multiplication**.



ALS Formulation

Cheap, first-pass algorithm (can be modified to impose nonnegativity): alternating least-squares.

- Keep B fixed, solve for A:

$$A := \min_X \left\| M \circledast (XB^\top) - S \right\|_F$$

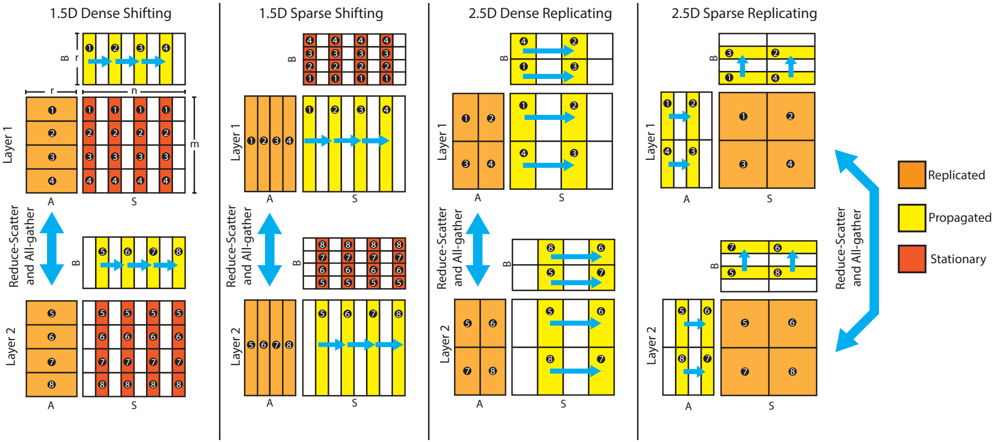
- In standard form, we get an independent LSTSQ problem for each output row of A (each E_j selects nonzero indices of $M[j,:]$):

$$\mathrm{vec}(A) := \mathrm{argmin}_x \left\| \begin{bmatrix} E_1 B & & & \\ & E_2 B & & \\ & & \ddots & \\ & & & E_m B \end{bmatrix} x - \mathrm{vec}_{nz}(S) \right\|_2$$

Our Contributions [BhBuDe22]

- **Prior work:** The ALS algorithm for matrix completion can be reformulated to depend entirely on the SDDMM and SpMM kernels (running in a conjugate-gradient loop) [CZ13; Nis+18].
- We build communication-avoiding, distributed-memory implementations of the SDDMM and SpMM kernels. We use them in sparse matrix factorization on **hundreds** of Cori CPU nodes.
- These can also be used for specific graph neural network architectures with self-attention (similar to CAGNET [TYB20]).

Existing Algorithms for SpMM



SpMM / SDDMM Duality

SDDMM and SpMM have **identical** data access patterns: every nonzero $(i, j) \in \mathrm{nz}(S)$ requires an interaction between row i of A and row j of B.

procedure SpMM(S, B)
  Initialize $A := 0$;
  for $(i, j) \in \mathrm{nz}(S)$ **do**
    $A[i, :] \mathrel{+}= S[i, j]\, B[j, :]$
  return A

procedure SDDMM(S, A, B)
  Initialize $R := 0$;
  for $(i, j) \in \mathrm{nz}(S)$ **do**
    $R[i, j] = S[i, j]\, (A[i, :] \cdot B[j, :])$
  return R

Observation: Every distributed algorithm for SpMM can be converted into an algorithm for SDDMM, and vice-versa.

Converting SpMM Algorithms to SDDMM Algorithms

Consider any distributed algorithm for SpMM that performs no replication. For all $k \in [1, r]$, the algorithm must (at some point)

- Co-locate $S[i,j]$, $A[i,k]$, $B[j,k]$ on a single processor
- Perform the update $A[i,k] \mathrel{+}= S[i,j]\, B[j,k]$

Transform the algorithm as follows:

1. Change the input sparse matrix S to an output initialized to 0.
2. Change A from an output to an input.
3. Have each processor execute the local update $S[i,j] \mathrel{+}= A[i,k]\, B[j,k]$

Dealing with Replication

Inputs are typically replicated via broadcasts, outputs via reduction. To handle this:

- Replace initial braodacasts of inputs with terminal reductions.
- Replace terminal reductions of outputs with initial broadcasts.

The resulting algorithm performs SDDMM up to multiplication with the original values in S.

We performed a communication analysis for several variants of SpMM / SDDMM, as well as optimizations that fuse the two kernels back-to-back.

Strong Scaling on LBNL Cori

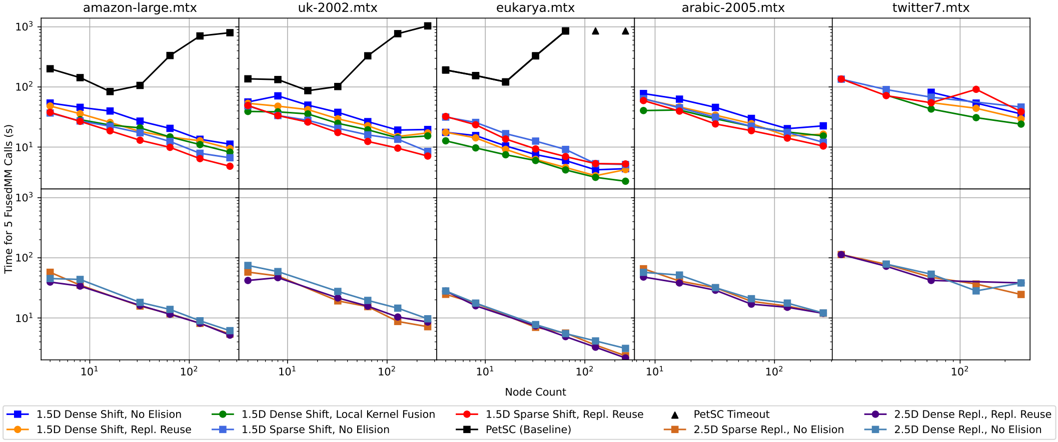


Figure 12: Strong scaling Experiments for 5 back-to-back SDDMM and SpMM calls on Cori CPU nodes.

Applications: Collaborative Filtering and GATs

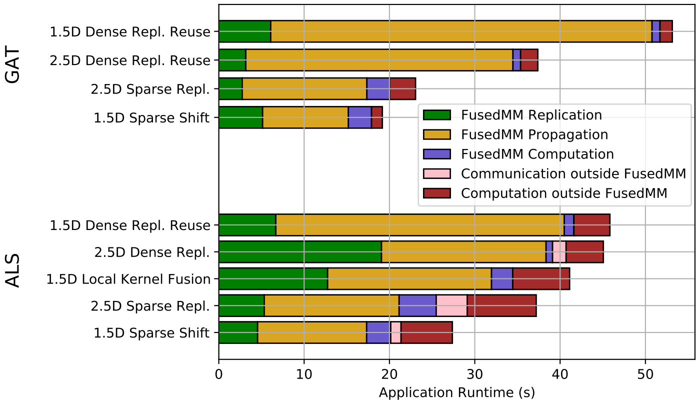


Figure 13: Application benchmarks for our distributed SDDMM / SpMM implementations

Takeaways: Communication Avoiding SDDMM / FusedMM Kernels

- We devised a procedure to convert well-analyzed SpMM algorithms into SDDMM algorithms.
- We analyzed the communication costs of a pair of back-to-back SpMM / SDDMM calls and demonstrated significant speedups at scale over the implementation in PETSc.
- We used our methods to accelerate ALS matrix completion on some of the largest matrices in the Suitesparse collection.

Future Work

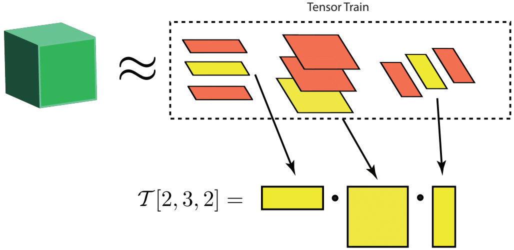
Work in Progress

Currently exploring three thrusts:

- Extensions of our CP sampling strategy to other tensor formats (mainly tensor trains).
- Application of sketching to domain science problems, such as Electrical Impedence Tomography (EIT).
- Accelerating other problems that involve tensor product structure, such as the marginalized graph kernel.

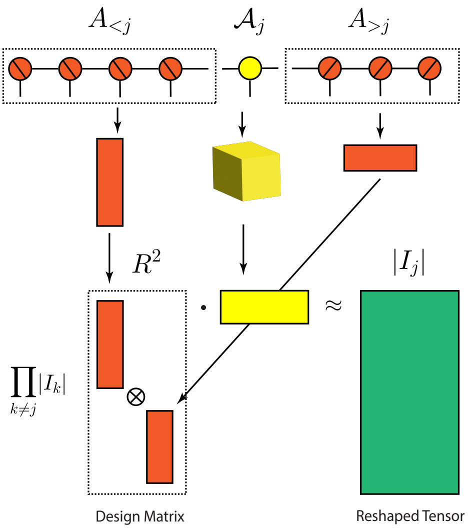
Extension of Implicit Sampling to Tensor Train Decomposition

The tensor-train decomposition represents a tensor $\mathcal{T}$ as a contraction between order-3 "tensor-cores".



j'th core has dimensions $R_j \times I_j \times R_{j+1}$. Represents a tensor with I^N elements using $O(NIR^2)$ space when all rank are equal.

Iterative TT Optimization Problems



Sampling from $A_{<j}$

Theorem (Orthonormal Subchain Leverage Sampling)

There exists a data structure that costs $O(IR^3)$ per tensor train core to build / update. For any $1 < j \leq N$, the structure can sample a row from $A_{<j}$ proportional to its squared row norm in time

$$O((j-1)R^2 \log I)$$

Apply same binary tree trick to the left matricizations of each core $\mathcal{A}_j$, exploit orthonormality to reduce complexity. Accelerates TT-ALS, potentially useful in other contexts.

Tensor Structure in PDE-Inverse Problems

Consider a 2D slice of conducting tissue. A source voltage is applied and the potential is measured at several pairs of boundary points.

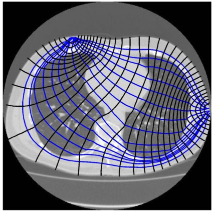
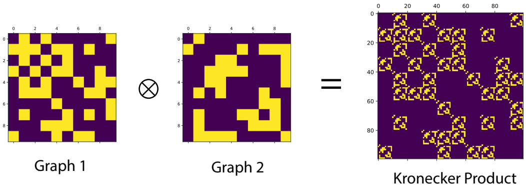


Figure 14: CT Image of thorax with EIT equipotential lines. Image credit Andy Adler, CC3.0 unported, Wikimedia Commons.

Goal: determine conductivity in **interior** of tissue. Solve $(U_{11} \odot U_{12} + U_{21} \odot U_{22})x = b$ where $U_{11}, U_{12}, U_{21}, U_{22}$ depend on the geometry of the tissue / boundary, b is a measurement taken for every source / sink pair [Che+20].

Sketching for the Marginalized Graph Kernel

The marginalized graph kernel computes a similarity measure between two (labeled, weighted) graphs G_1, G_2 by finding the stationary distribution of a random walk on their **Kronecker Product Graph** [Vis+10].



For the **inner product edge kernel**: solve $\left(\sum_{i=1}^{N} A_i \otimes B_i\right) x = p$ where A_i, B_i have the sparsity structures of adjacency matrices of A_1, A_2. Potentially a ripe application for Tensorsketch, low-rank approximation.

Conclusions and Acknowledgements

Summary

- We exhibited algorithms for ALS CP decomposition that have **lower asymptotic complexity and faster time-to-solution** compared to SOTA competitors.
- We showed that randomized methods are practical on **thousands of CPU cores** and **billion-scale sparse tensors**, offering up to 11x speedup over carefully-engineered deterministic algorithms
- We optimized the SDDMM kernel involved in sparse matrix factorization based on proven algorithms for SpMM, exploiting **algorithmic duality** between the two kernels.
- Planned work this year: investigate other regression problems that involve tensor structure, particularly scientific applications.

Acknowledgements

This slide will expand to a whole section in my dissertation talk! For now, my appreciation goes to:

- Friends and past / present members of BeBOP and PASSION.
- Collaborators Laura Grigori, Osman Asif Malik, Riley Murray.
- Committee members Katherine Yelick and Michael Lindsey.
- My fantastic advisors, Aydın Buluç and James Demmel.

This work was funded by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0022158.

[**Bh**BuDe22] Vivek Bharadwaj, Aydın Buluç, and James Demmel. **"Distributed-Memory Sparse Kernels for Machine Learning".** In: *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2022, pp. 47–58.

[**Bh**MMBD23] Vivek Bharadwaj, Osman Asif Malik, Riley Murray, Aydin Buluç, and James Demmel. ***Distributed-Memory Randomized Algorithms for Sparse Tensor CP Decomposition*.** 2023. arXiv: `2210.05105 [math.NA]`.

[**Bh**MMGBD23] Vivek Bharadwaj, Osman Asif Malik, Riley Murray, Laura Grigori, Aydın Buluç, and James Demmel. **"Fast Exact Leverage Score Sampling from Khatri-Rao Products with Applications to Tensor Decomposition".** In: *Thirty-seventh Conference on Neural Information Processing Systems*. Dec. 2023.

[AMT10] Haim Avron, Petar Maymounkov, and Sivan Toledo. **"Blendenpik: Supercharging LAPACK's Least-Squares Solver".** In: *SIAM Journal on Scientific Computing* 32.3 (2010), pp. 1217–1236.

References ii

[And+92] E Anderson, Z Bai, C Bischof, J Demmel, J Dongarra, J Du Croz, A Greenbaum, S Hammarling, A McKenney, S Ostrouchov, and D Sorensen. ***LAPACK users‘ guide: Release 1.0*.** Jan. 1992.

[Bla+97] L. Blackford, J. Choi, A. Cleary, J. Demmel, Inderjit S. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ***ScaLAPACK Users' Guide*.** SIAM, 1997.

[Che+16] Dehua Cheng, Richard Peng, Yan Liu, and Ioakeim Perros. **"SPALS: Fast Alternating Least Squares via Implicit Leverage Scores Sampling".** In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016.

[Che+20] Ke Chen, Qin Li, Kit Newton, and Stephen J. Wright. **"Structured Random Sketching for PDE Inverse Problems".** In: *SIAM Journal on Matrix Analysis and Applications* 41.4 (2020), pp. 1742–1770. doi: `10.1137/20M1310497`.

[CZ13] John Canny and Huasha Zhao. **"Big Data Analytics with Small Footprint: Squaring the Cloud".** en. In: *KDD*. 2013.

References iii

[DKM06] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. **"Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication".** In: *SIAM Journal on Computing* 36.1 (2006), pp. 132–157. doi: `10.1137/S0097539704442684`.

[DM20] Michal Derezinski and Michael W. Mahoney. **"Determinantal Point Processes in Randomized Numerical Linear Algebra".** In: *CoRR* abs/2005.03185 (2020). arXiv: `2005.03185`.

[Gat+20] Mark Gates, Ali Charara, Jakub Kurzak, Asim YarKhan, Mohammed Al Farhan, Dalal Sukkari, and Jack Dongarra. ***SLATE Users' Guide*.** SLATE Working Notes 10, ICL-UT-19-01. July 2020.

[Kan+12] U. Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. **"GigaTensor: scaling tensor analysis up by 100 times - algorithms and discoveries".** In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '12. New York, NY, USA: Association for Computing Machinery, Aug. 2012, pp. 316–324. isbn: 978-1-4503-1462-6.

[KB09] Tamara G. Kolda and Brett W. Bader. **"Tensor Decompositions and Applications".** In: *SIAM Review* 51.3 (Aug. 2009). Publisher: Society for Industrial and Applied Mathematics, pp. 455–500. issn: 0036-1445. doi: `10.1137/07070111X`.

[Law+79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. **"Basic Linear Algebra Subprograms for Fortran Usage".** In: *ACM Trans. Math. Softw.* 5.3 (Sept. 1979), pp. 308–323. issn: 0098-3500.

[LK22] Brett W. Larsen and Tamara G. Kolda. **"Practical Leverage-Based Sampling for Low-Rank Tensor Decomposition".** In: *SIAM Journal on Matrix Analysis and Applications* 43.3 (2022), pp. 1488–1517.

[Mal22] Osman Asif Malik. **"More Efficient Sampling for Tensor Decomposition With Worst-Case Guarantees".** In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 14887–14917.

References v

[Min+23] Rachel Minster, Irina Viviano, Xiaotian Liu, and Grey Ballard. **“CP decomposition for tensors via alternating least squares with QR decomposition”.** In: *Numerical Linear Algebra with Applications* 30.6 (2023), e2511. doi: `https://doi.org/10.1002/nla.2511`.

[MS22] Linjian Ma and Edgar Solomonik. **“Cost-efficient Gaussian tensor network embeddings for tensor-structured inputs”.** In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, pp. 38980–38993.

[Mur+23] Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michal Derezinski, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. ***Randomized Numerical Linear Algebra: A Perspective on the Field With an Eye to Software.*** Tech. rep. UCB/EECS-2023-19. EECS Department, University of California, Berkeley, Feb. 2023.

[Nis+18] Israt Nisa, Aravind Sukumaran-Rajam, Sureyya Emre Kurt, Changwan Hong, and P. Sadayappan. **“Sampled Dense Matrix Multiplication for High-Performance Machine Learning”.** In: *HiPC*. Dec. 2018, pp. 32–41.

[RT08] Vladimir Rokhlin and Mark Tygert. **"A fast randomized algorithm for overdetermined linear least-squares regression".** In: *Proceedings of the National Academy of Sciences* 105.36 (2008), pp. 13212–13217. doi: `10.1073/pnas.0804869105`.

[Smi+15] Shaden Smith, Niranjay Ravindran, Nicholas D. Sidiropoulos, and George Karypis. **"SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication".** In: *2015 IEEE International Parallel and Distributed Processing Symposium*. ISSN: 1530-2075. May 2015, pp. 61–70.

[Smi+17] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. ***FROSTT: The Formidable Repository of Open Sparse Tensors and Tools*.** 2017. url: `http://frostt.io/`.

[TDB10] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. **"Towards dense linear algebra for hybrid GPU accelerated manycore systems".** In: *Parallel Computing* 36.5-6 (June 2010), pp. 232–240. issn: 0167-8191.

[TYB20] Alok Tripathy, Katherine Yelick, and Aydın Buluç. **"Reducing Communication in Graph Neural Network Training".** In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2020.

[Vis+10] S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. **“Graph Kernels”.** In: *Journal of Machine Learning Research* 11.40 (2010), pp. 1201–1242. url: `http://jmlr.org/papers/v11/vishwanathan10a.html`.

[Woo14] David P. Woodruff. **“Sketching as a Tool for Numerical Linear Algebra”.** In: *Found. Trends Theor. Comput. Sci.* 10.1–2 (Oct. 2014), pp. 1–157. issn: 1551-305X.

Backup Slides, Deck 1: Fast Khatri-Rao Product Leverage Score Sampling

Leverage Score Sampling Proof Sketch

Theorem (Structural Conditions for LSTSQ, [DKM06])

Let Q be a basis for the column-space of A. Suppose that a sketching matrix S satisfies the following two ***deterministic*** *structural conditions:*

- ***(S1) Approximate Isometry***: $\sigma_{min}(SQ) \geq 1/\sqrt{2}$
- ***(S2) Minimal Junk***: $\left\|Q^\top S^\top S B^\perp\right\|_F^2 \leq \varepsilon \left\|B^\perp\right\|_F^2 /2$

Then the sketched solution $\tilde{X}$ satisfies

$$\left\|A\tilde{X} - B\right\|_F \leq (1+\varepsilon) \min_X \|AX - B\|_F .$$

Main Proof Idea: Show, with probability $\geq (1-\delta)$, that a leverage score sketch satisfies these two conditions.

Leverage Score Sampling Proof Sketch

- (S1) holds by the well-known ℓ_2-subspace embedding property of leverage score sketches [Woo14], with probability $\geq 1 - \delta/2$ for high enough sample count.

- (S2) holds by an approximate matrix-multiplication argument [DKM06] (with one-sided information) with probability $\geq 1 - \delta/2$.

$$\begin{aligned} \left\|Q^\top S^\top S B^\perp\right\|_F^2 &= \left\|\mathbf{0} - Q^\top S^\top S B^\perp\right\|_F^2 \\ &= \left\|Q^\top B^\perp - Q^\top S^\top S B^\perp\right\|_F^2 \end{aligned}$$

- Use a union bound to guarantee that both hold with probability $\geq 1 - \delta$. Will sketch the proof of (S1).

Leverage Score Sampling Gives an ℓ_2-SE Proof Sketch

Proof follows a version by David Woodruff (we adapt it to our notation and drop the β parameter). Let $A = Q\Sigma V^\top$; we need a matrix Chernoff result.

Theorem (Matrix Chernoff, [Woo14])

Let $X_1, \ldots, X_J$ be independent copies of a symmetric random matrix $X \in \mathbb{R}^{R\times R}$ satisfying

1. $E[X] = 0$,
2. $\|X\|_2 \leq \gamma$,
3. $\left\|E\left[X^\top X\right]\right\|_2 \leq T \leq J^2$.

Let $W = \frac{1}{J}\sum_{i=1}^{J} X_i$. Then for any $\tilde{\varepsilon} > 0$,

$$Pr\left[\|W\|_2 > \tilde{\varepsilon}\right] \leq 2R\exp\left(-J\tilde{\varepsilon}^2/(2T + 2\gamma\tilde{\varepsilon}/3)\right)$$

Leverage Score Sampling Gives an ℓ_2-SE Proof Sketch

Want to show, for appropriate parameters J, γ, ε, that $\frac{1}{\sqrt{2}} \leq \sigma_i^2(SU)$ w.h.p. $(1-\delta)$. Let $z_i = (SU)_{i:}^\top$, $q_j = Q_{j:}^\top$ and choose

$$p_j := \ell_j / R \;\; \forall j$$

$$X_i := I - z_i z_i^\top / p_i$$

$$\gamma := 1 + R$$

$$\tilde{\epsilon} := 1 - 1/\sqrt{2}$$

$$T := R - 1$$

Easy to verify that $\mathrm{E}\left[X\right] = 0$, need to check conditions (2) and (3) of the Chernoff bound.

Leverage Score Sampling Gives an ℓ_2-SE

Condition 2: $\|X\|_2 \leq \gamma$ implies $\max_{j \in [I]} \left\| I - \frac{q_j q_j^\top}{p_j} \right\| \leq \gamma$. For any j, we have

$$\begin{aligned} \left\| I - \frac{q_j q_j^\top}{p_j} \right\|_2 &\leq \|I\|_2 + \left\| \frac{q_j q_j^\top}{p_j} \right\|_2 \\ &= 1 + \frac{R \left\| q_j q_j^\top \right\|_2}{\left\| q_j \right\|_2^2} \\ &= 1 + R \\ &= \gamma \end{aligned}$$

Crucially, this choice for p_j allows the **minimal** choice $1 + R$ for γ.

Leverage Score Sampling Gives an ℓ_2-SE

Condition 3: We derive

$$\begin{aligned}
\mathrm{E}\left[X^\top X\right] &= \sum_{j=1}^{I} p_j (I - q_j q_j^\top / p_j)(I - q_j q_j^\top / p_j) \\
&= \sum_{j=1}^{I} p_j I - 2 \sum_{j=1}^{I} p_j q_j q_j^\top / p_j + \sum_{j=1}^{I} \frac{p_j q_j q_j^\top q_j q_j^\top}{p_j^2} \\
&= I - 2I + \sum_{j=1}^{I} \frac{q_j q_j^\top q_j q_j^\top}{p_j} \\
&= I - 2I + \sum_{j=1}^{I} R q_j q_j^\top \\
&= (R-1)I
\end{aligned}$$

So $\left\|\mathrm{E}\left[X^\top X\right]\right\|_2 = R - 1 \le J^2$.

Leverage Score Sampling Gives an ℓ_2-SE

Evaluating the Chernoff guarnatee, we ignore $\tilde{\epsilon}$ since it is a constant. We **want**

$$\exp\left(-J\tilde{\varepsilon}^2/(2T + 2\gamma\tilde{\varepsilon}/3)\right) \leq \delta$$

$$J/(2R + 2R/3)) \geq \Omega\left(\log\frac{R}{\delta}\right)$$

Setting $J = \Omega\left(R\log\frac{R}{\delta}\right)$ causes the failure probability to fall below the threshold.

The Normal Equations in Tensor Decomposition

- The normal equations are widely used for ALS CP decomposition [KB09] despite squaring the condition number.

- QR decomposition of a KRP is more difficult to compute (but only slightly) [Min+23]:

$$
\begin{aligned}
A &:= U_1 \odot \ldots \odot U_N \\
&= (Q_1 R_1) \odot \ldots \odot (Q_N R_N) \\
&= (Q_1 \otimes \ldots \otimes Q_N) \cdot (R_1 \odot \ldots \odot R_N) \\
&= (Q_1 \otimes \ldots \otimes Q_N) \cdot Q_{\text{tail}} \cdot R_{\text{tail}}
\end{aligned} \tag{2}
$$

- QR formulation useful for lower-precision decomposition, adversarial tensors [Min+23], e.g. $\sin(x_1 + \ldots + x_N)$.

Why Don't We Use the QR Formulation?

- QR Decomposition not useful for leverage score sampling. R^N samples required to sketch $Q_1 \otimes ... \otimes Q_N$, computation of Q_{tail} introduces exponential cost in N.
- Leverage score computation robust to numerical error (just take slightly more samples).
- For our applications, we can sacrifice some accuracy.

Backup Slides, Deck 2: Randomized Distributed CP Decomposition

d-STS-CP Parallelization Scheme

- Matrices G^v replicated $\log P$ times. Each processor stores data on path from leaf to root.
- **Initialization:** Each sample assigned arbitrarily to a processor (along with corresponding sample vectors h).
- **At Each Node:** Branching decision made for each sample, Alltoallv computed to reorganize sample vectors.
- **Drawback:** Repeated Alltoallv calls are expensive!

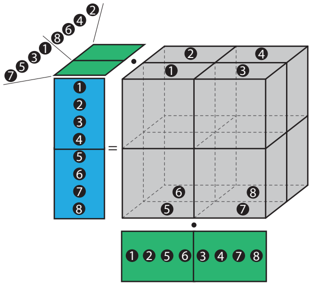
Non-Randomized Communication Analysis

- Let processor grid dimensions be $P_1 \times \ldots \times P_N$.
- All-gather + Reduce-Scatter Costs:

$$2\sum_{k=1}^{N} IR/P_k$$

- Cost Under Optimal Grid:

$$\frac{2NRI}{P^{1/N}}$$



Tensor Stationary MTTKRP

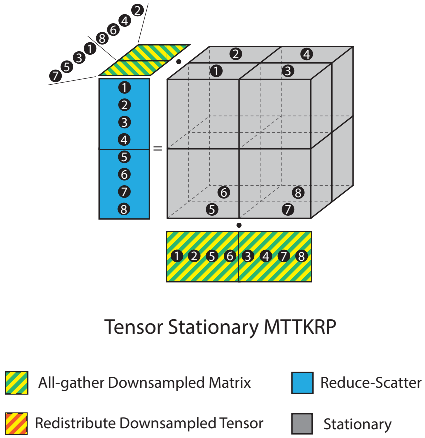
All-gather Matrix
Reduce-Scatter
Redistribute Downsampled Tensor
Stationary

Downsampled Tensor-Stationary MTTKRP

- Reduce-scatter cost is unchanged by sampling.
- Minimum communication:

$$\frac{NRI}{P^{1/N}}$$

- Drops at most a constant factor compared to non-randomized ALS



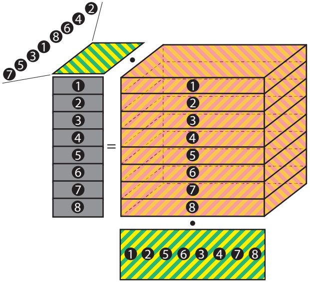
Downsampled Accumulator-Stationary MTTKRP

- Eliminate reduce-scatter by gathering sampled rows to all processors, redistributing sampled nonzeros.

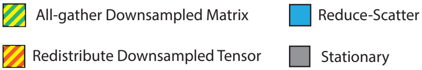
- Communication Cost:

$$JRN(N-1) + \frac{3}{P}\sum_{j=1}^{N} \mathrm{nnz}(\mathrm{mat}(\mathcal{T}, j)S_j^\top).$$

- Avoid retransmitting nonzeros by storing N different matricizations of the tensor.

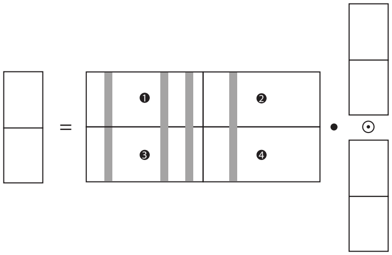


Accumulator Stationary MTTKRP



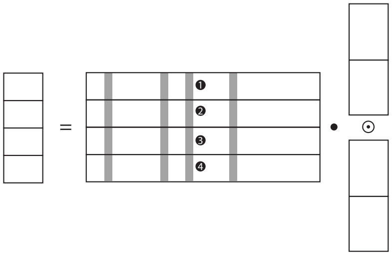
Tensor-Stationary MTTKRP Load Balance

- We use random permutations of each tensor mode to evenly distribute nonzeros & samples to processors.
- Theoretical model: each sampled column has q nonzeros with row i.i.d. uniform.
- TS Load Balance: J balls into $P^{1-1/N}$ bins (each ball here is a column).



Accumulator-Stationary MTTKRP Load Balance

- AS Load Balance: Jq balls into P bins.
- Here, each ball is a nonzero entry. This distibution has better load balance when q is high.



Load Balance

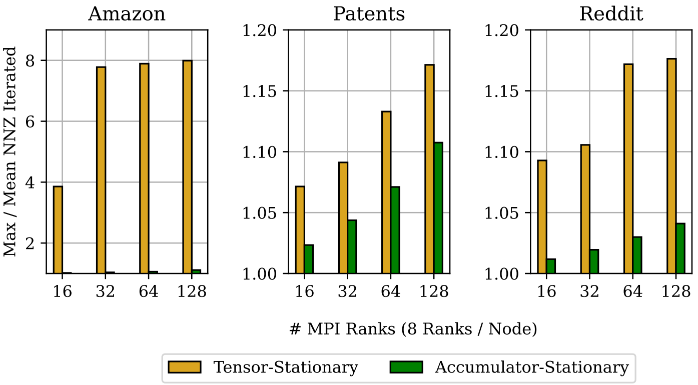
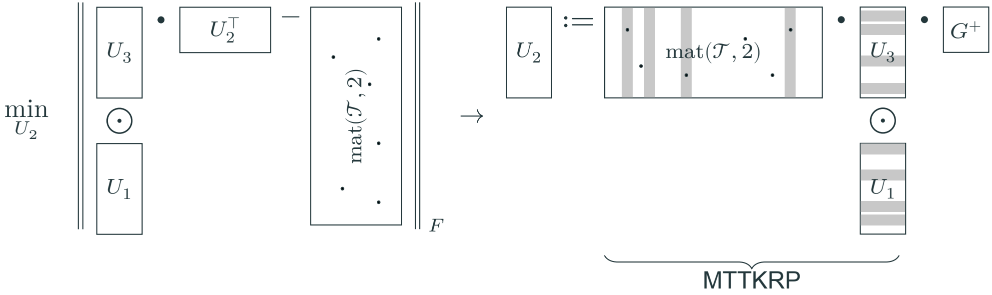


Figure 15: Load imbalance for tensor-stationary vs. accumulator stationary schedules as a function of MPI rank count.

Local Computation: SpMTTKRP is SpMM

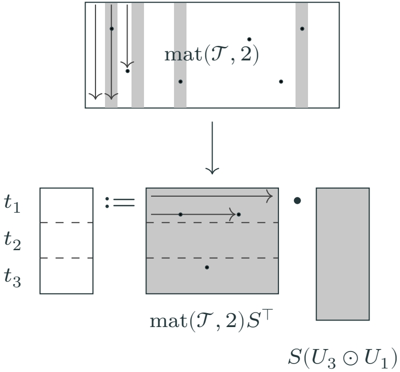
$$\min_{U_j} \left\| \left[\bigodot_{k\neq j} U_k \right] \cdot U_j^\top - \mathrm{mat}(\mathcal{T}, j)^\top \right\|_F$$



Matricized Tensor Storage Format

- **CSC**: Easy to look up nonzeros, but need atomics when accumulating to output buffer (with multiple threads)
- **CSR:** No data races, but difficult to select nonzeros.
- **Solution:** Use CSC for lookup, sparse transpose into CSR.

Sampling and Sparse Transpose Operation



Drawback: Need to store N copies of the sparse tensor, but we do this anyway to avoid communication.

Weak Scaling

- **Weak scaling for non-randomized CP**: increase target rank R and processor count proportionally, measure runtime.

- **Problems for Randomized CP:**
  - Nonzero count selected from sparse tensor varies.
  - Need higher sample counts at higher ranks to maintain accuracy.

- **Solution:** Benchmark STS-CP with fixed sample count to maintain accuracy (as much as possible) for a fixed sample count, measure throughput instead:

$$\text{Throughput} = \frac{\text{nnz selected in MTTKRP}}{\text{Runtime}}$$

Experimental Platform

- Experiments conducted on up to 16 nodes / 2048 CPU cores on NERSC Perlmutter at LBNL.
- Hybrid OpenMP / MPI implementation in C++, Python wrappers using Pybind11.
- Baseline : SPLATT, a highly-optimized CP decomposition library.

Figure 16: LBNL Perlmutter, an HPE Cray Supercomputer (#12 on the Nov’23 Top500).

Weak Scaling

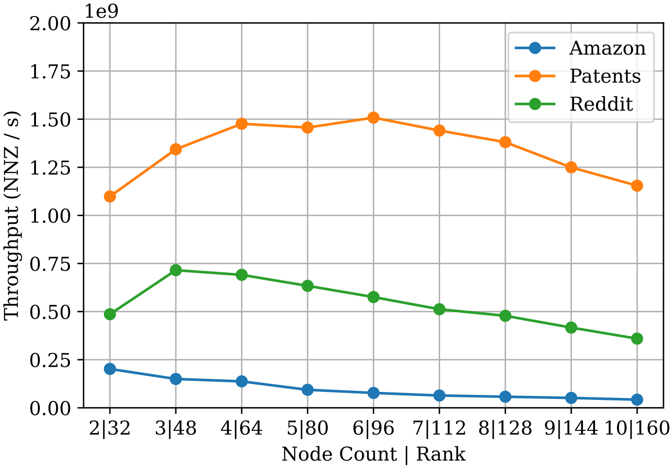


Figure 17: Throughput as a function of increasing target rank and node count.