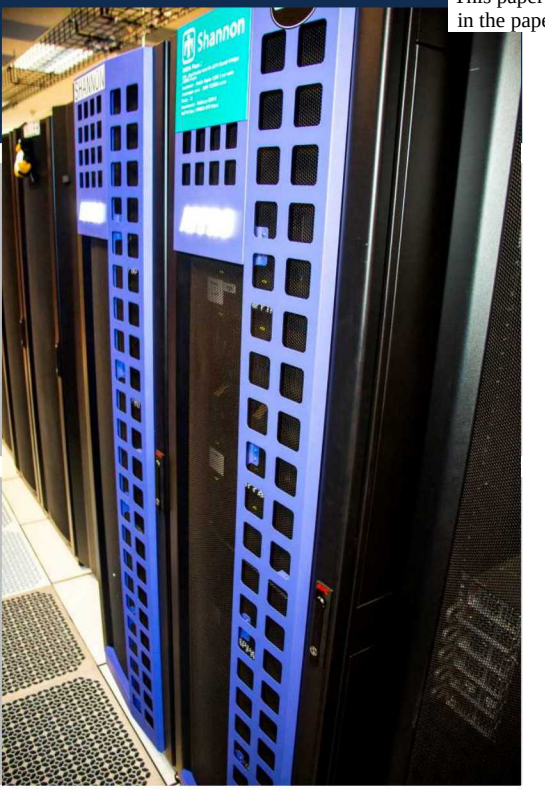


SAND2020-1377C

CCR
Center for Computing Research

Computing Generalized CP Decompositions on Emerging Parallel Architectures

Eric Phipps (etphipp@sandia.gov)
and Tammy Kolda,
Sandia National Laboratories
Albuquerque New Mexico and Livermore California, USA

SIAM Conference on Parallel Processing for Scientific Computing (PP20)

February 12-15, 2020

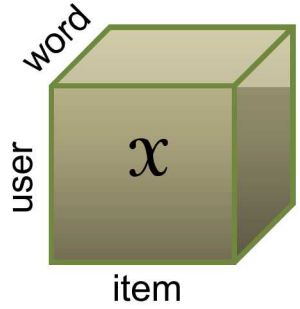
U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

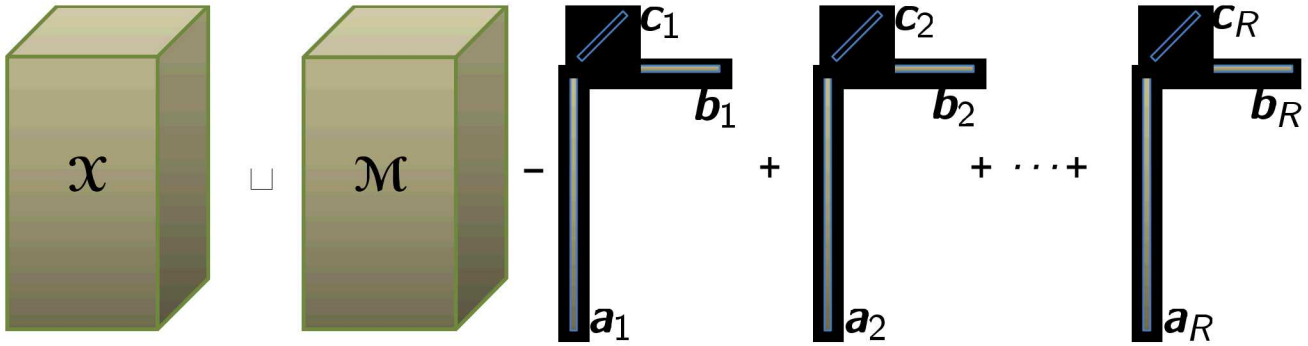
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2020-xxxx C

Tensors

- N-way array used to represent multi-relationship data
  - E.g., word frequencies in Amazon product rankings:



- Canonical Polyadic (CP) tensor decomposition
  - Approximate tensor as a sum of rank-1 tensors
  - Discovers dominant relationships in data



$$\mathcal{X} \approx \mathcal{M} = \boldsymbol{a}_1 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_1 + \boldsymbol{a}_2 \circ \boldsymbol{b}_2 \circ \boldsymbol{c}_2 \cdots + \boldsymbol{a}_R \circ \boldsymbol{b}_R \circ \boldsymbol{c}_R$$

$$x(i,j,k) \approx m(i,j,k) = \sum_{l=1}^{R} a(i,l)b(j,l)c(k,l)$$

CP via Alternating Least-Squares (ALS)

- CP minimization problem
  - This assumes *Gaussian model* for tensor, resulting in least-squares

$$\min_{\mathcal{M}} \ \|\mathcal{X} - \mathcal{M}\|_F^2 \ \text{ s.t. } \ \mathcal{M} = \boldsymbol{a}_1 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_1 + \cdots + \boldsymbol{a}_R \circ \boldsymbol{b}_R \circ \boldsymbol{c}_R$$

- Solve via alternating linear least squares
  - Fix all but one term, solve linear least squares problem, iterate

Using…

$$\mathcal{M}_{(1)} = \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T$$
$$\mathcal{M}_{(2)} = \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})^T$$
$$\mathcal{M}_{(3)} = \boldsymbol{C}(\boldsymbol{B} \odot \boldsymbol{A})^T$$

Repeat until convergence…

$$\min_{\boldsymbol{A}} \left\|\mathcal{X}_{(1)} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\right\|_F^2$$
$$\min_{\boldsymbol{B}} \left\|\mathcal{X}_{(2)} - \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})^T\right\|_F^2$$
$$\min_{\boldsymbol{C}} \left\|\mathcal{X}_{(3)} - \boldsymbol{C}(\boldsymbol{B} \odot \boldsymbol{A})^T\right\|_F^2$$

Repeat until convergence…

$$\boldsymbol{A} = \mathcal{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B})(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{B}^T\boldsymbol{B})^\dagger$$
$$\boldsymbol{B} = \mathcal{X}_{(2)}(\boldsymbol{C} \odot \boldsymbol{A})(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{A}^T\boldsymbol{A})^\dagger$$
$$\boldsymbol{C} = \underbrace{\mathcal{X}_{(3)}(\boldsymbol{B} \odot \boldsymbol{A})}_{\text{MTTKRP}}(\boldsymbol{B}^T\boldsymbol{B} * \boldsymbol{A}^T\boldsymbol{A})^\dagger$$

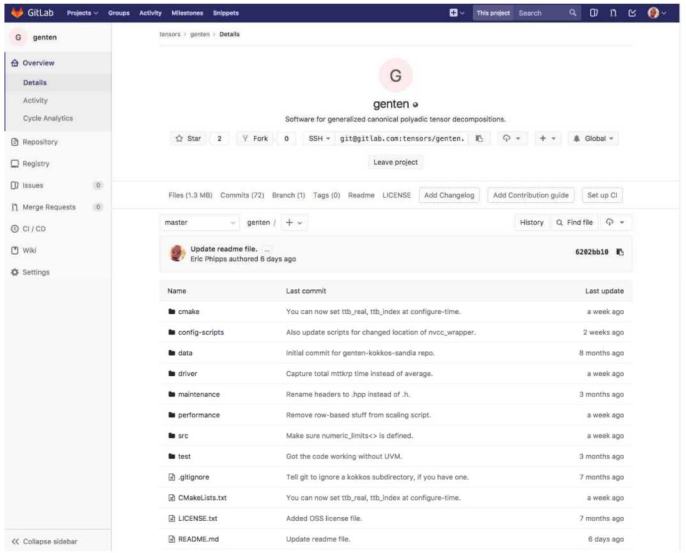
- Matricized Tensor Times Khatri-Rao Product (MTTKRP):
  - Most expensive part of CP-ALS

$$\boldsymbol{V} = \mathcal{X}_{(3)}(\boldsymbol{A} \odot \boldsymbol{B}) \implies v(i,l) = \sum_{(i,j,k)\in\mathcal{N}(\mathcal{X})} x(i,j,k)a(j,l)b(k,l), \quad l = 1,\ldots,R$$

GenTen : Software for Generalized Canonical Polyadic Tensor Decompositions

Sandia National Laboratories

- New software package Genten developed at SNL
  - E. Phipps, T. Kolda, D. Dunlavy, G. Ballard, T. Plantenga
  - Based on C++ port of Matlab Tensor Toolbox
  - Publicly available at https://gitlab.com/tensors/genten
  - Implements full CP-ALS algorithm for sparse (and dense) tensors, as well as GCP algorithm for sparse tensors



- Incorporates shared memory parallelism for emerging manycore hardware using **Kokkos**
  - Multicore CPUs via OpenMP, pThreads
  - GPUs via Nvidia Cuda (Intel and AMD coming soon)
  - Intel Xeon Phi (a.k.a. KNC/KNL) via OpenMP

- Implements parallelism for all performance-critical operations
  - **MTTKRP**, tensor inner product, norms, ...
  - Can use optimized third-party libraries (MKL, cuBLAS, ...)
  - Natively handles data transfers between CPU, GPU memory

- Callable from Matlab Tensor Toolbox!

What is Kokkos?

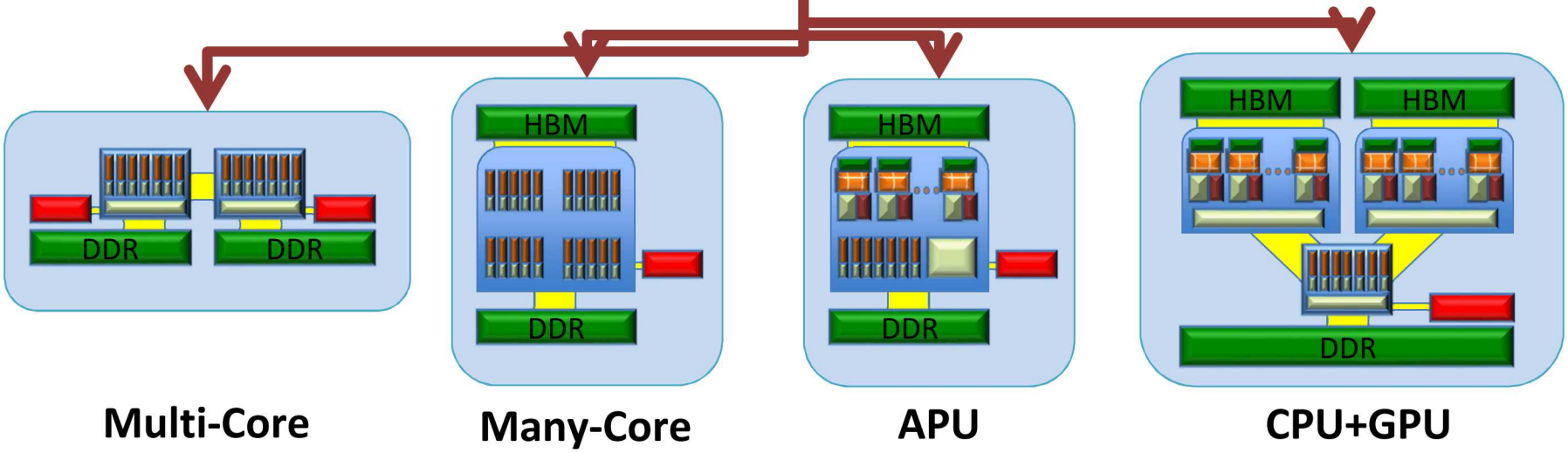
Sandia National Laboratories

LAMMPS
Albany
Drekar
EMPIRE
SPARC
SIERRA
Trilinos
etc...

Applications & Libraries

Kokkos

performance portability for C++ applications



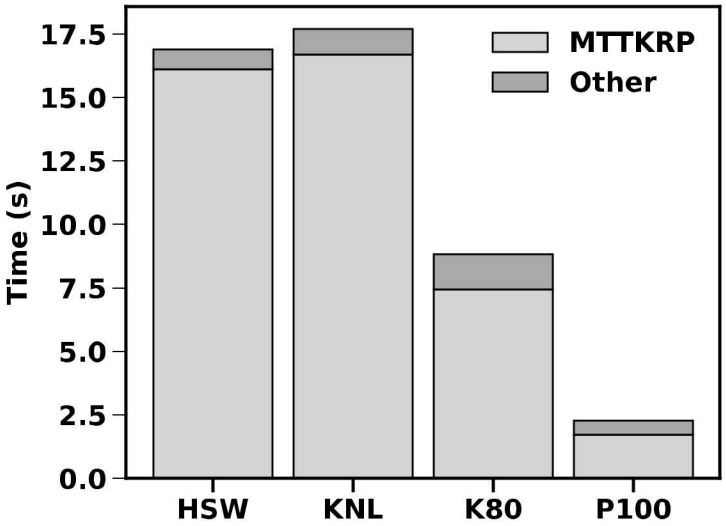
Multi-Core **Many-Core** **APU** **CPU+GPU**

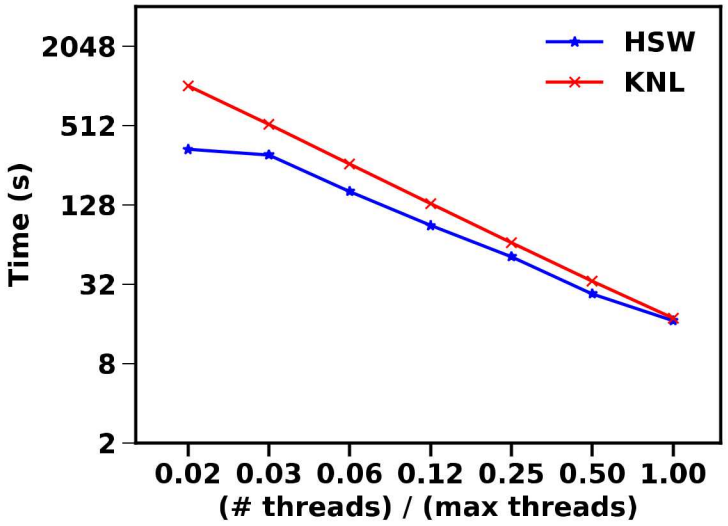
H.C. Edwards, C. Trott, *et al.*, https://github.com/kokkos/kokkos

Genten CP-ALS Performance*

- Synthetic data tensor (double precision)
  - 30K x 40K x 50K, 10M non-zeros, R = 128 (perf_CpAlsRandomKtensor performance test in Genten)
  - 10 CP-ALS iterations

- Architectures
  - HSW (OpenMP): Intel Xeon E5-2698v3 CPU, 2.3 GHz, 2 sockets, 16 cores/socket, 2 threads/core
  - KNL (OpenMP): Intel Xeon Phi 7250, 68 cores (using up to 64), 4 threads/core, 16 GB HBM in cache mode
  - K80 (Cuda): Nvidia Kepler K80 GPU, 12 GB GDDR5
  - P100 (Cuda): Nvidia Pascal P100 GPU, 16 GB HBM





*E. Phipps and T. Kolda, *Software for Sparse Tensor Decomposition on Emerging Computing Architectures,* SIAM SISC, vol. 41 (3), 2019, (https://doi.org/10.1137/18M1210691).

Performance on Real Tensors

Sandia National Laboratories

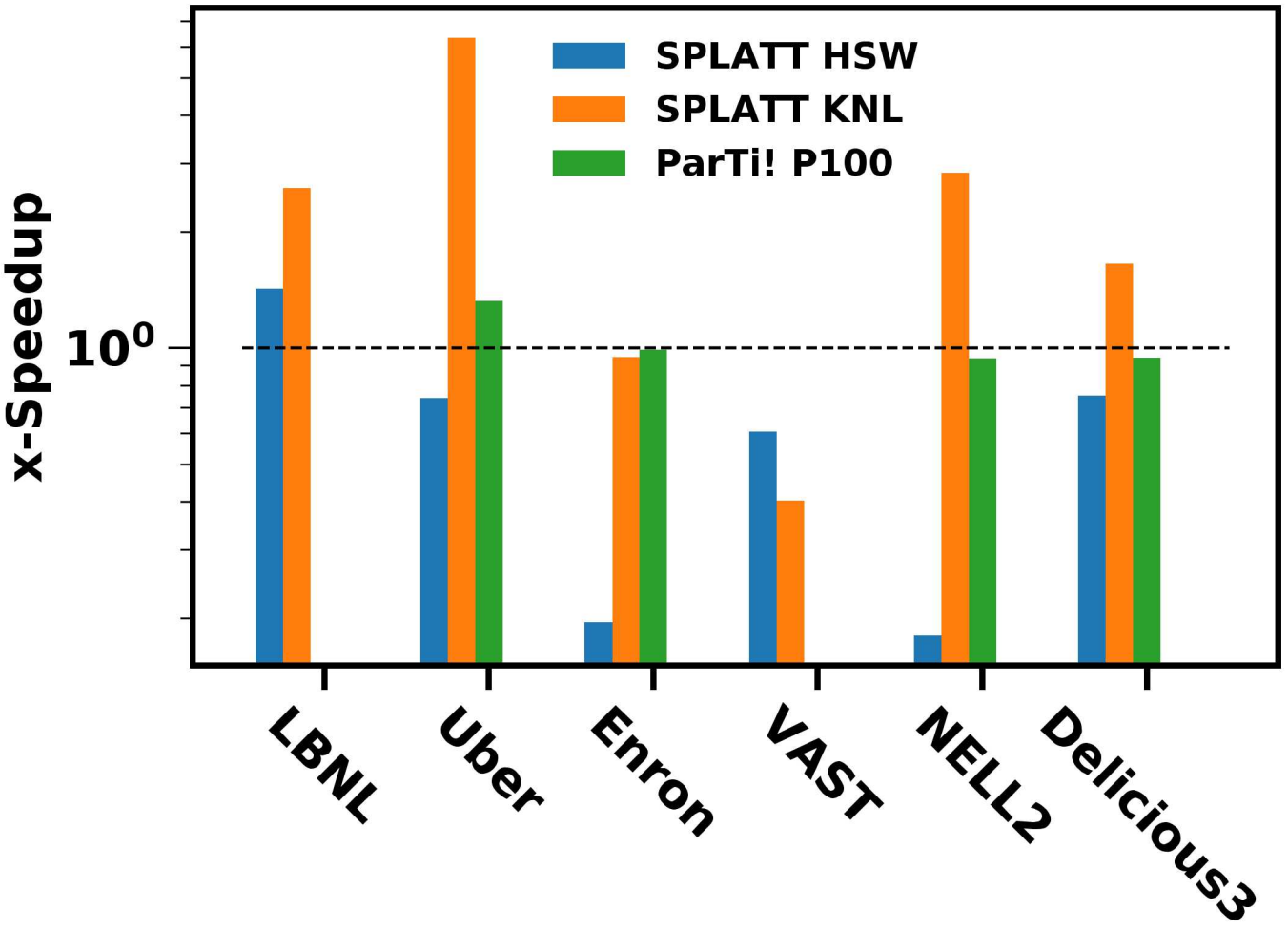
- Look at Genten MTTKRP performance on several tensors from the FROSTT collection
  - Shaden Smith et al, https://frost.io
  - Fixed 10 iterations of CP-ALS
  - R =16 for all tests

Name	Order	Dimensions	Nonzeros
LBNL	5	1.6K □ 4.2K □ 1.6K □ 4.2K □ 868K	1.7M
Uber	4	183 □ 24 □ 1.1K □ 1.7K	13M
Enron	4	6.0K □ 5.7K □ 244K □ 1.2K	54M
VAST	5	165K □ 11K □ 2 □ 100 □ 89	26M
NELL2	3	12K □ 9.1K □ 29K	77M
Delicious3	3	532K □ 17M □ 2.5M	140M

- Compare GenTen MTTKRP to SPLATT on HSW/KNL and ParTI! on GPU
  - SPLATT: Shaden Smith et al, http://glaros.dtc.umn.edu/gkhome/splatt/overview
    - Compressed Sparse Fiber (CSF) storage format (Smith and Karypis, 2015)
    - Use default 2 CSF modes, with and without tiling
  - ParTI!: J. Li et al, https://github.com/hpcgarage/ParTI
    - Only works with order-3 and order-4 tensors
    - Requires double-precision atomics (no K80)

GenTen Speedup for Best MTTKRP Algorithm

Sandia National Laboratories

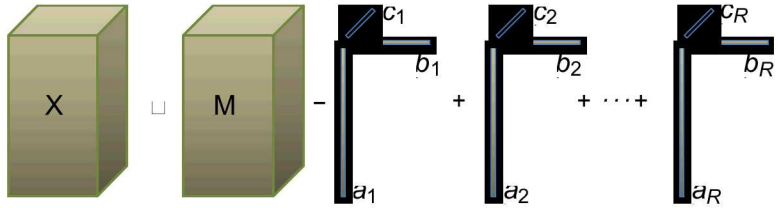


Review of Generalized CP (GCP)*

- Standard CP optimization problem:

$$\min_{\mathcal{M}} \quad F(\mathcal{X}, \mathcal{M}) = \|\mathcal{X} - \mathcal{M}\|_F^2 = \sum_i (x_i - m_i)^2$$

$$\text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!] = \boldsymbol{a}_1 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_1 + \cdots + \boldsymbol{a}_R \circ \boldsymbol{b}_R \circ \boldsymbol{c}_R$$



- Objective function derived through maximum likelihood estimation assuming the data tensor (X) entries are i.i.d. Gaussian:

$$x_i \sim N(x_i | m_i, \sigma) = \frac{1}{2\pi\sigma^2} e^{(x_i - m_i)^2/(2\sigma^2)} \implies$$

$$L(\mathcal{M}|\mathcal{X}) = \prod_i N(x_i | m_i, \sigma) \implies$$

$$-\log L(\mathcal{M}|\mathcal{X}) = \sum_i \left(\frac{(x_i - m_i)^2}{2\sigma^2} + \frac{1}{2}\log(2\pi\sigma^2) \right) \sim \sum_i (x_i - m_i)^2$$

*Hong, Kolda, Duersch. Generalized Canonical Polyadic Tensor Decomposition. SIAM Review, 2019.

Review of Generalized CP (GCP)*

- Generalize to other types of data through arbitrary loss function:

$$x_i \sim p(x_i|\theta_i)$$
$$f(x_i, m_i) \equiv p(x_i|\ell^{-1}(m_i)) \implies \min_{\mathcal{M}} \quad F(\mathcal{X}, \mathcal{M}) = \sum_i f(x_i, m_i)$$
$$\text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$$

loss function

link function

Distribution	Link function	Loss function	Constraints
$\mathcal{N}(\mu, \sigma)$	$m = \mu$	$(x-m)^2$	$x, m \in \mathbb{R}$
Gamma(k, σ)	$m = k\sigma$	$x/(m+\epsilon) + \log(m+\epsilon)$	$x > 0, m \geq 0$
Poisson(λ)	$m = \lambda$	$m - x\log(m+\epsilon)$	$x \in \mathbb{N}, m \geq 0$
	$m = \log\lambda$	$e^m - xm$	$x \in \mathbb{N}, m \in \mathbb{R}$
Bernoulli(ρ)	$m = \rho/(1-\rho)$	$\log(m+1) - x\log(m+\epsilon)$	$x \in \{0,1\}, m \geq 0$
	$m = \log(\rho/(1-\rho))$	$\log(1+e^m) - xm$	$x \in \{0,1\}, m \in \mathbb{R}$

*Hong, Kolda, Duersch. Generalized Canonical Polyadic Tensor Decomposition. SIAM Review, 2019.

Fitting GCP Model

$$\min_{\mathcal{M}} \; F(\mathcal{X}, \mathcal{M}) = \sum_i f(x_i, m_i) \quad \text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d]\!]$$

- Lose the least-squares structure underlying ALS-type algorithms. Instead pursue gradient-based optimization approach.

- Define tensor Y such that

$$y(i_1, \ldots, i_d) = y_i = \frac{\partial f}{\partial m}(x_i, m_i)$$

- Then gradient of objective function given by

$$\boldsymbol{G}_k = \frac{\partial F}{\partial \boldsymbol{A}_k} = \mathcal{Y}_{(k)} (\boldsymbol{A}_d \odot \cdots \odot \boldsymbol{A}_{k+1} \odot \boldsymbol{A}_{k-1} \odot \cdots \odot \boldsymbol{A}_1)$$

← MTTKRP!

- Unfortunately, Y is in general dense, even when X is sparse, making standard optimization infeasible. Instead, employ Stochastic Gradient Descent (SGD) where Y is only randomly sampled

Sampling for SGD*

- Given index set Ω, define $\tilde{\mathcal{Y}} : \tilde{y}_i = \begin{cases} \omega_i y_i, & i \in \Omega \\ 0, & i \notin \Omega \end{cases}$
- SGD theory only requires $\mathbb{E}[\tilde{\mathcal{Y}}] = \mathcal{Y}$

- Uniform sampling
  - Sample s indices uniformly with replacement: $\tilde{y}_i = \tilde{s}_i \cdot \frac{n^d}{s} \cdot y_i, \quad n^d \equiv \prod_k n_k$
  - Will miss nonzeros if X is very sparse

- Stratified sampling
  - Sample p nonzeros and q nonzeros uniformly with replacement
  - Requires searching X for each sample to determine if zero or nonzero

$$\tilde{y}_i = \begin{cases} \tilde{p}_i \cdot \frac{N}{p} \cdot y_i, & i \in \Omega_{nz} \\ \tilde{q}_i \cdot \frac{n^d - N}{q} \cdot y_i, & i \in \Omega_z \end{cases}, \quad N = \text{nnz}(\mathcal{X})$$

- Semi-stratified sampling
  - Sample p nonzeros and q "zeros" uniformly with replacement
  - Correct for sampled zeros that were really nonzeros

$$\tilde{y}_i = \begin{cases} \tilde{p}_i \cdot \frac{N}{p} \cdot (y_i - c_i) + \tilde{q}_i \cdot \frac{n^d}{q} \cdot c_i, & i \in \Omega_{nz} \\ \tilde{q}_i \cdot \frac{n^d}{q} \cdot y_i, & i \in \Omega_z \end{cases}, \quad c_i = \frac{\partial f}{\partial m}(0, m_i)$$

*Kolda, Hong, Duersch. Stochastic Gradients for Large-Scale Tensor Decomposition. arXiv 1906.01687, 2019.

Performance Considerations

Sandia National Laboratories

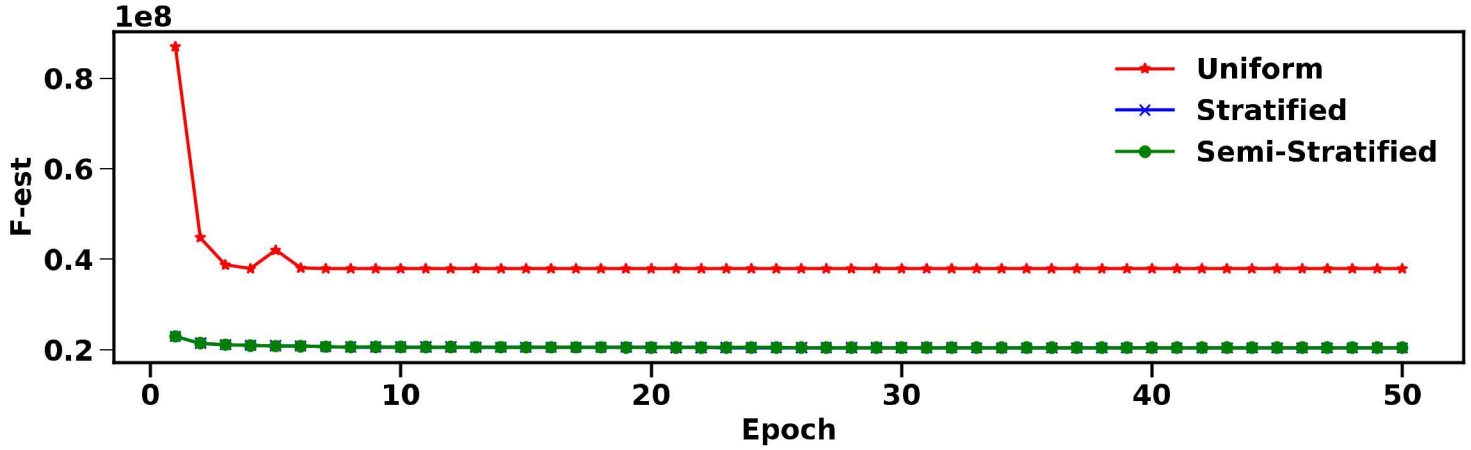
- Even for semi-stratified we use stratified for estimating objective function
- Stratified sampling requires efficient way of determining whether a given sample corresponds to a zero or nonzero
  - Sort tensor indices lexicographically and do binary search
    - O(log(N)) search, no increase in storage
    - Oversample Y tensor, sort, and do bulk search
  - Construct hash-map with tensor indices as keys
    - O(1) search but O(N) increase in storage
    - Use Kokkos::UnorderedMap for parallel, lock-free map construction
- For gradient evaluation, explicitly construct sampled Y tensor and pass to MTTKRP
  - Allows use of highly optimized MTTKRP kernels from CP-ALS

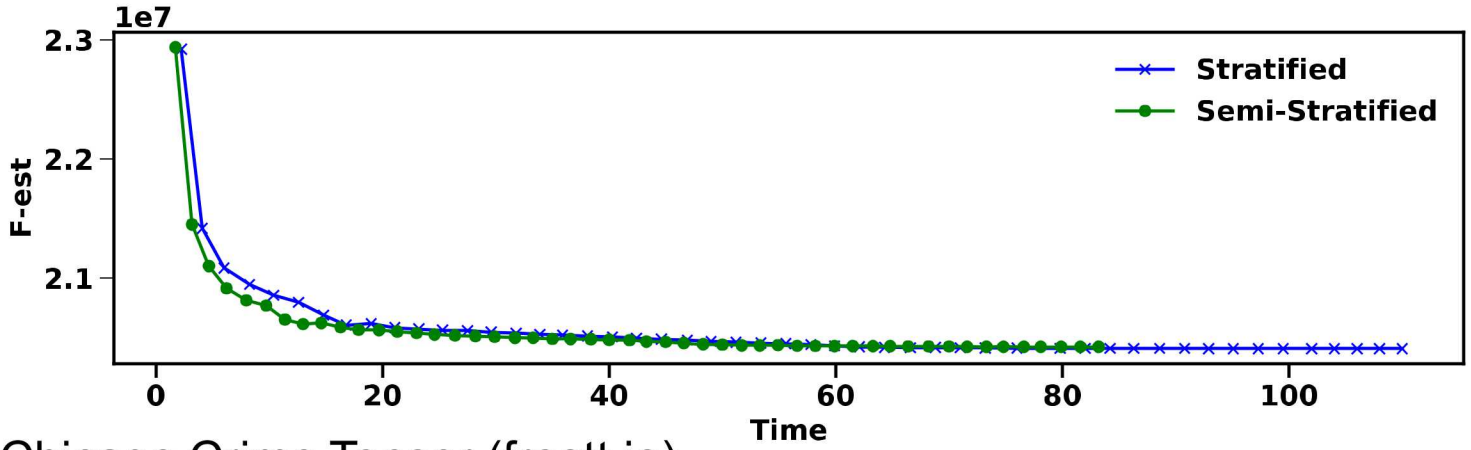
GCP-SGD (ADAM) Algorithm

Sandia National Laboratories

1: **function** GCPADAM($\mathcal{X}$, $\{\boldsymbol{A}_k\}_{k=1,\ldots,d}$, κ, τ)
2: **for** $k = 1, \ldots, d$ **do**
3: $\boldsymbol{G}_k, \boldsymbol{M}_k, \boldsymbol{V}_k \leftarrow$ zero matrix of size $n_k \times r$
4: $\mathcal{Y}_f \leftarrow$ SAMPLEFOROBJECTIVE($\mathcal{X}$)
5: $\hat{F} \leftarrow$ COMPUTEOBJECTIVE($\mathcal{Y}_f, \{\boldsymbol{A}_k\}$)
6: **while** $\hat{F} >$ TOLERANCE and at most κ iterations **do**
7: Save copies of $\{\boldsymbol{A}_k\}$, $\{\boldsymbol{M}_k\}$, $\{\boldsymbol{V}_k\}$, $\hat{F}$
8: **for** τ iterations **do**
9: $\mathcal{Y}_g \leftarrow$ SAMPLEFORGRADIENT($\mathcal{X}$)
10: $\{\tilde{\boldsymbol{G}}_k\} \leftarrow \mathrm{MTTKRP}(\mathcal{Y}_g, \{\boldsymbol{A}_k\})$
11: $\{\boldsymbol{A}_k\}, \{\boldsymbol{M}_k\}, \{\boldsymbol{V}_k\} \leftarrow$
12: ADAM($\{\boldsymbol{G}_k\}, \{\boldsymbol{A}_k\}, \{\boldsymbol{M}_k\}, \{\boldsymbol{V}_k\}$)
13: $\{\boldsymbol{A}_k\} \leftarrow$ CLIP($\{\boldsymbol{A}_k\}$)
14: $\hat{F} \leftarrow$ COMPUTOBJECTIVE($\mathcal{Y}_f, \{\boldsymbol{A}_k\}$)
15: **if** $\hat{F} > \hat{F}_{\text{old}}$ **then**
16: Restore saved copied of $\{\boldsymbol{A}_k\}$, $\{\boldsymbol{V}_k\}$, $\{\boldsymbol{V}_k\}$, $\hat{F}$
17: Increase learning rate

Comparing Sampling Approaches





- Chicago Crime Tensor (frostt.io)
  - 6186 x 24 x 77 x 32, ~533k nonzeros,
  - Poisson loss,
  - ~15k zero/nonzero samples

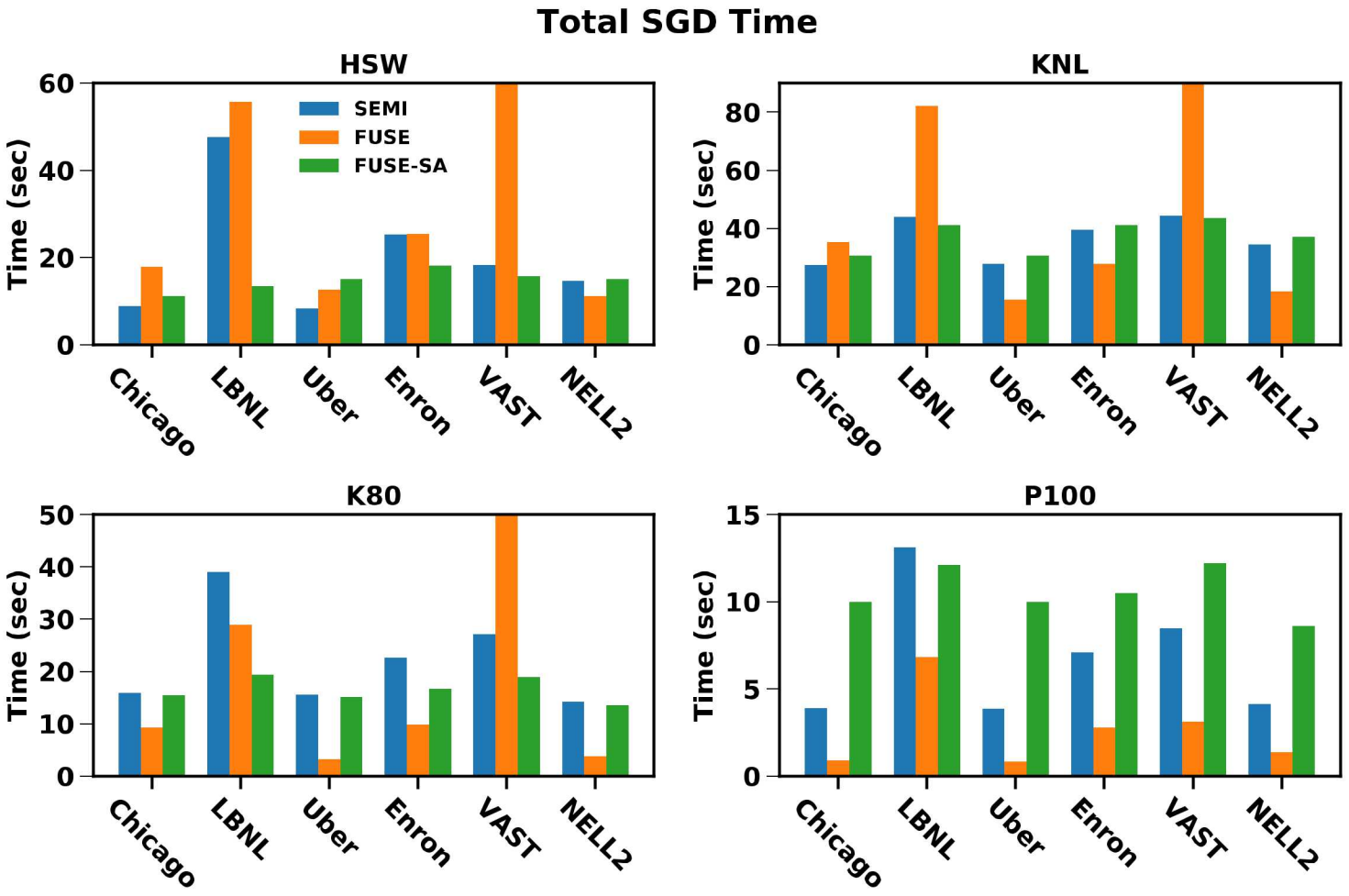
Improving Gradient Computation

- Idea: For semi-stratified, fuse sampling and MTTKRP kernels
  - Avoid explicit construction of sampled tensor
  - Compute MTTKRP in all modes simultaneously
  - Currently restricted to atomics for MTTKRP factor matrix update
  - Should work well for GPU
- What about architectures without fast atomics: ``Sparse Array"
  - Put MTTKRP contribution for each nonzero in Y in a separate row of temporary buffer along with row index (no thread contention)
  - Reduce entries with the same row index (parallel reduce-by-key)
  - Perform ADAM update and clip with only the nonzero rows
  - Should work well when # samples << mode length

function FusedUpdate($\mathcal{X}$, $\{\boldsymbol{A}_k\}$, $\{\boldsymbol{G}_k\}$, $\{\boldsymbol{M}_k\}$, $\{\boldsymbol{V}_k\}$)
  parallel for $j = 1, \ldots, s$ **do**
    $y, i_1, \ldots, i_d \leftarrow$ SemiStratifiedSample($\mathcal{X}, j$)
    for $k = 1, \ldots, d$ **do**
      $\boldsymbol{G}_k(i_k, :) \leftarrow_a \boldsymbol{G}_k(i_k, :) + y \cdot \prod_{j \neq k} \boldsymbol{A}_j(i_j, :)$
  $\{\boldsymbol{A}_k\}, \{\boldsymbol{M}_k\}, \{\boldsymbol{V}_k\} \leftarrow$
    Adam($\{\boldsymbol{G}_k\}, \{\boldsymbol{A}_k\}, \{\boldsymbol{M}_k\}, \{\boldsymbol{V}_k\}$)
  $\{\boldsymbol{A}_k\} \leftarrow$ Clip($\{\boldsymbol{A}_k\}$)

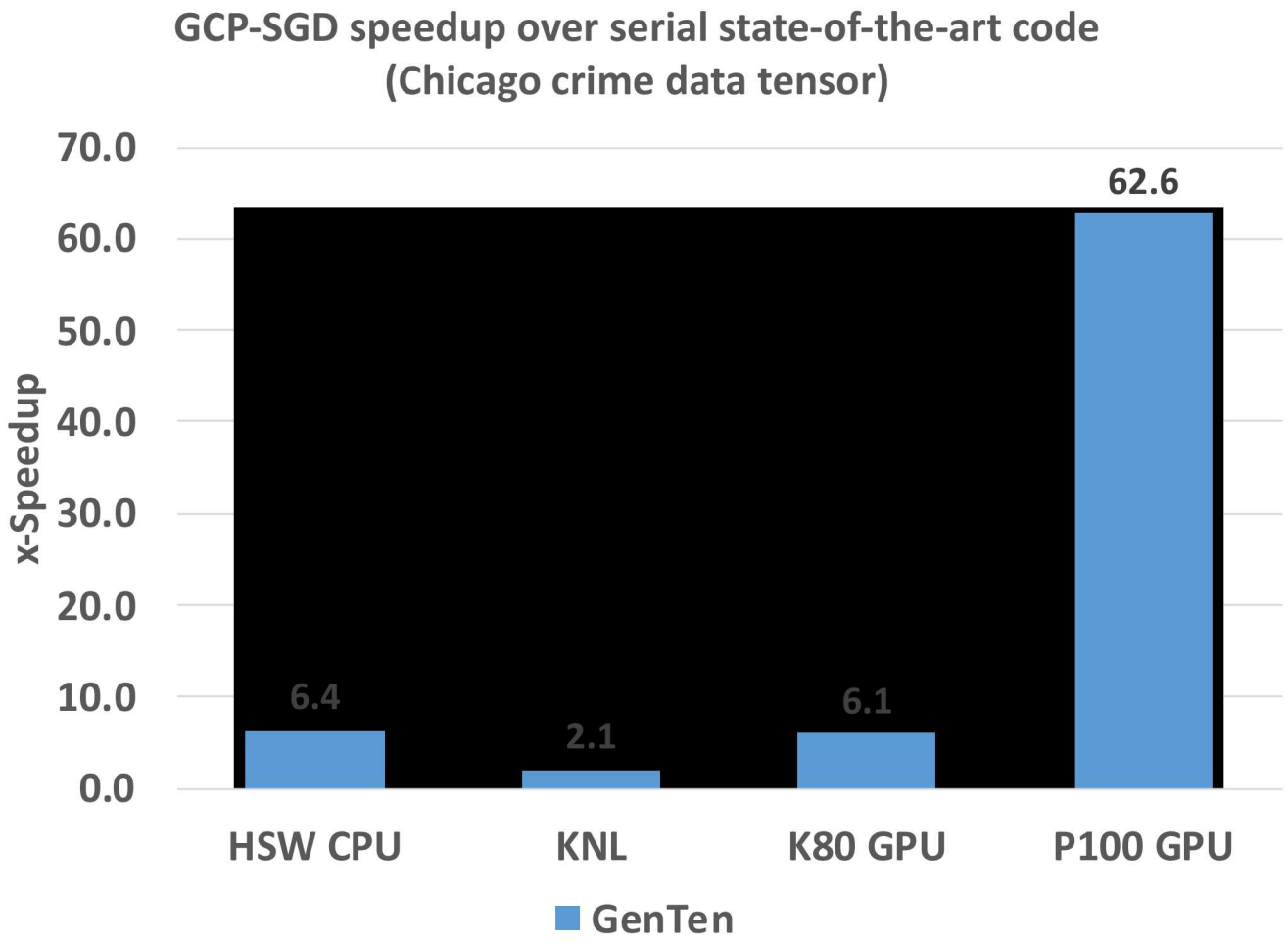
function SparseArrayUpdate($\mathcal{X}$, $\{\boldsymbol{A}_k\}$, $\{\boldsymbol{M}_k\}$, $\{\boldsymbol{V}_k\}$)
  for $k = 1, \ldots, d$ **do**
    $\tilde{\boldsymbol{G}}_k \leftarrow$ zero matrix of size $s \times r$ ▷ Sample MTTKRPs
    $\boldsymbol{R}_k \leftarrow$ zero vector of size s ▷ Row keys
  parallel for $j = 1, \ldots, s$ **do**
    $y, i_1, \ldots, i_d \leftarrow$ SemiStratifiedSample($\mathcal{X}, j$)
    for $k = 1, \ldots, d$ **do**
      $\boldsymbol{R}_k(j) \leftarrow i_k$
      $\tilde{\boldsymbol{G}}_k(j, :) \leftarrow y \cdot \prod_{j \neq k} \boldsymbol{A}_j(i_j, :)$ ▷ MTTKRP
  for $k = 1, \ldots, n$ **do**
    $\boldsymbol{R}_k, \tilde{\boldsymbol{G}}_k, t_k \leftarrow$ ParallelReduceByKey($\boldsymbol{R}_k, \tilde{\boldsymbol{G}}_k$)
    parallel for $j = 1, \ldots, t_k$ **do**
      $i_k \leftarrow \boldsymbol{R}_k(j)$
      $\boldsymbol{A}_k(i_k, :), \boldsymbol{M}_k(i_k, :), \boldsymbol{V}_k(i_k, :)) \leftarrow$
        Adam($\tilde{\boldsymbol{G}}_k(j, :), \boldsymbol{A}_k(i_k, :), \boldsymbol{M}_k(i_k, :), \boldsymbol{V}_k(i_k, :)$)
      $\boldsymbol{A}_k(i_k, :) \leftarrow$ Clip($\boldsymbol{A}_k(i_k, :)$)

GCP-SGD Performance Comparison



15K zero and nonzero samples for all tensors

GenTen Speedup over Matlab TTB



Summary & Conclusions

Sandia National Laboratories

- New software package Genten
  - Publicly available at https://gitlab.com/tensors/genten
  - POC: Eric Phipps, etphipp@sandia.gov
- Built on Kokkos for shared-memory parallelism and high performance, portable to emerging parallel architectures
- Supports traditional CP-ALS as well as new GCP-SGD method
- Callable directly from Matlab Tensor Toolbox through GenTen-Matlab MEX interface
  - Enables high productivity and analysis power of the Matlab Tensor Toolbox accelerated by fast architectures such as GPUs!
- Future Work
  - Investigate number of gradient samples vs. MTTKRP algorithms
  - Continue to improve sampling/gradient kernels
  - Sparse array approach motives asynchronous SGD (a la HogWild!)

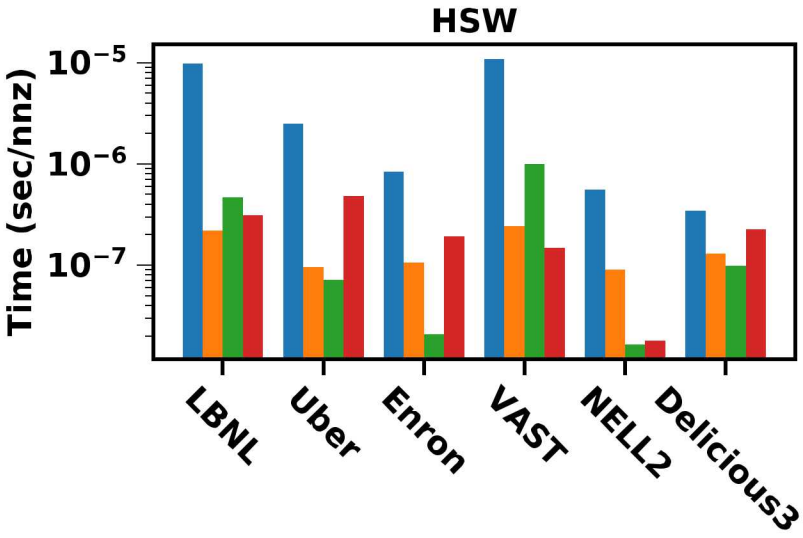
Acknowledgments

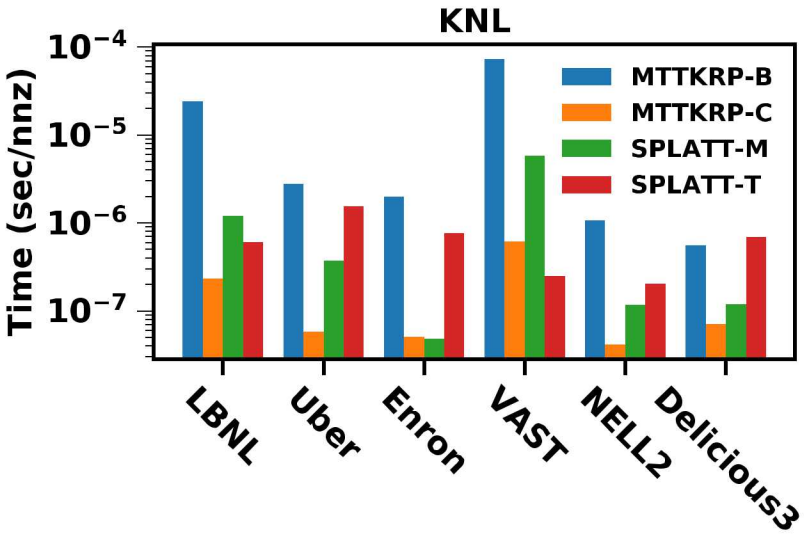
- This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

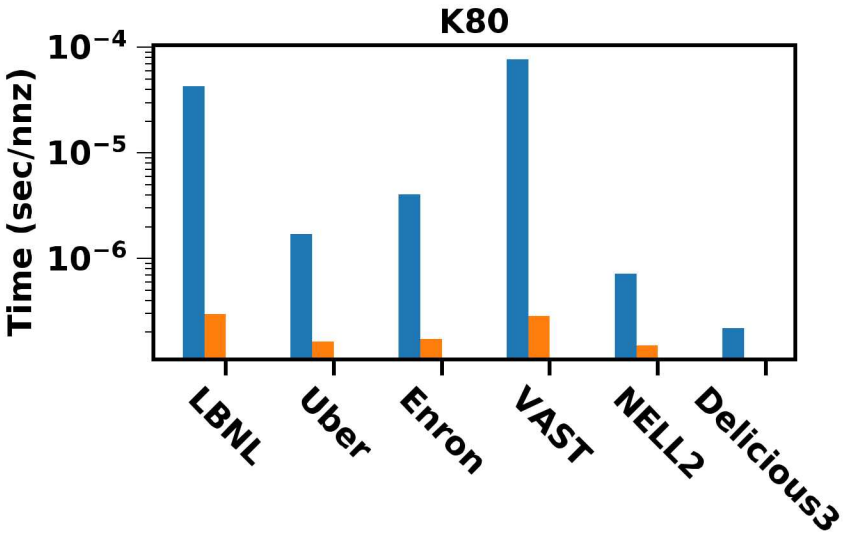
Sandia National Laboratories

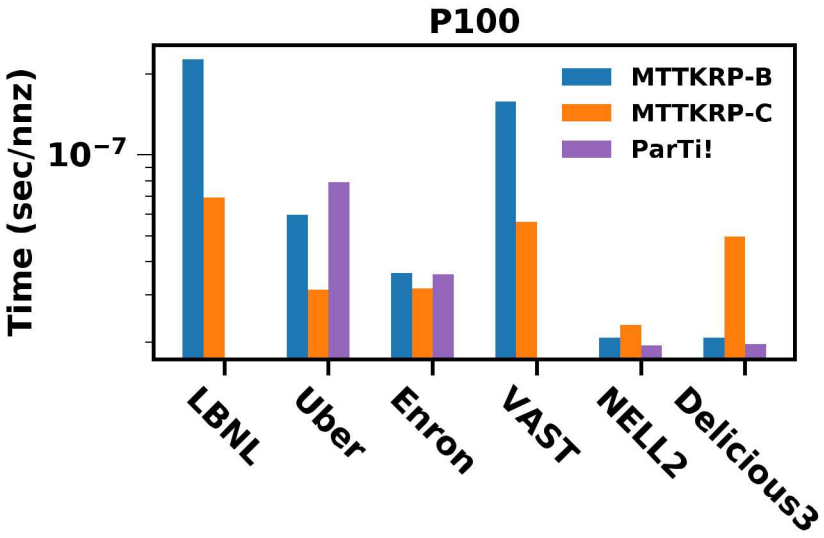
Backup Slides

Total MTTKRP Time per Nonzero









Tensor Sorting Cost

Sandia National Laboratories

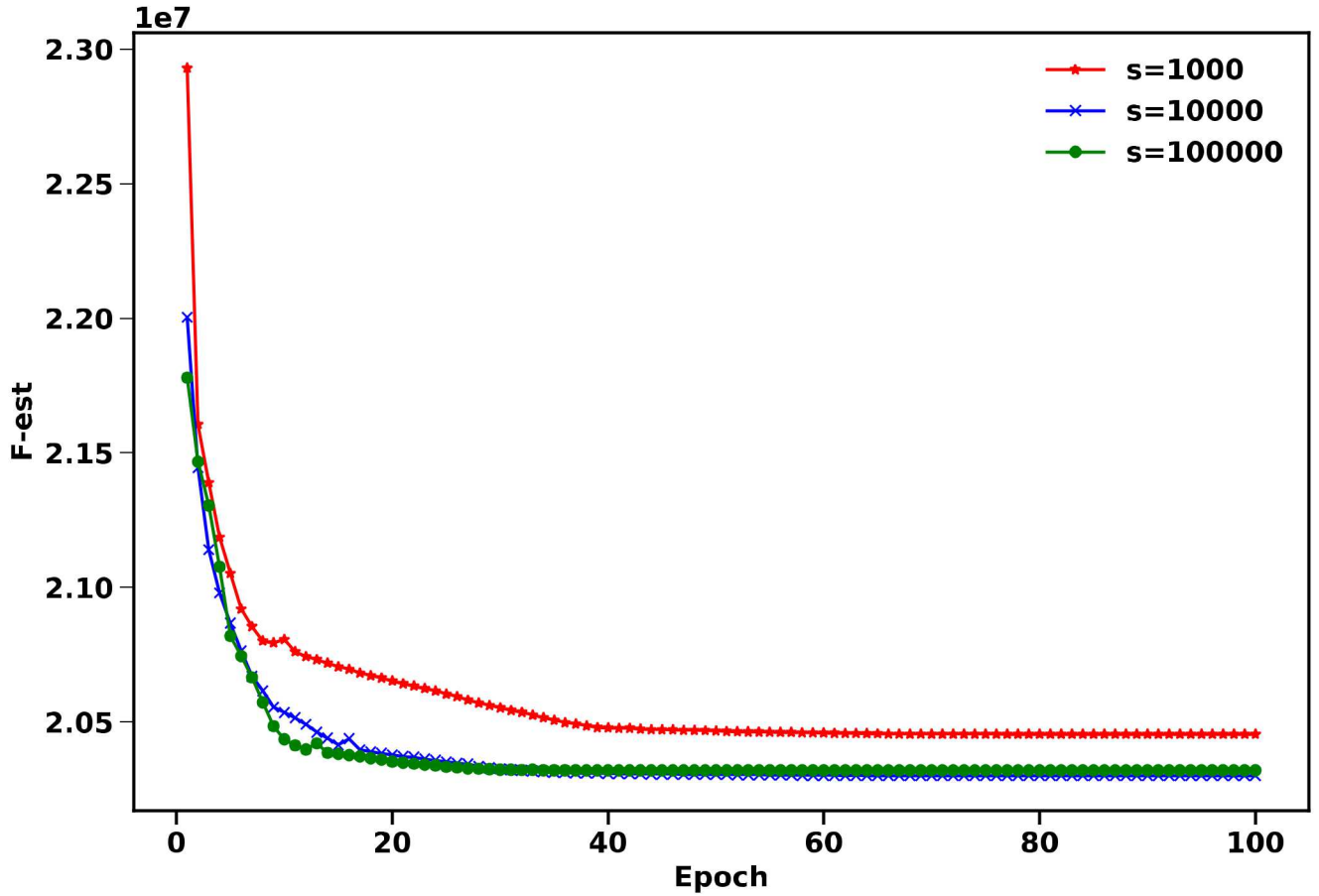
- Permutation method requires *N* sorting operations
  - Sort tensor coordinates for each mode
  - Small, but not inconsequential cost

- Sorting methods used
  - OpenMP: Parallel stable sort from Intel
  - Cuda: Thrust stable sort

Sorting time divided by avg. (permuted) CP-ALS iteration time

Architecture	LBNL	Uber	Enron	VAST	NELL2	Delicious
HSW	0.6	4.3	9.2	5.3	6.2	4.0
KNL	1.0	5.5	8.6	1.3	7.6	4.9
K80	1.1	3.0	3.3	3.3	2.6	–
P100	0.5	1.3	4.2	3.6	4.6	–

SGD-ADAM Convergence with Varying Sample Sizes

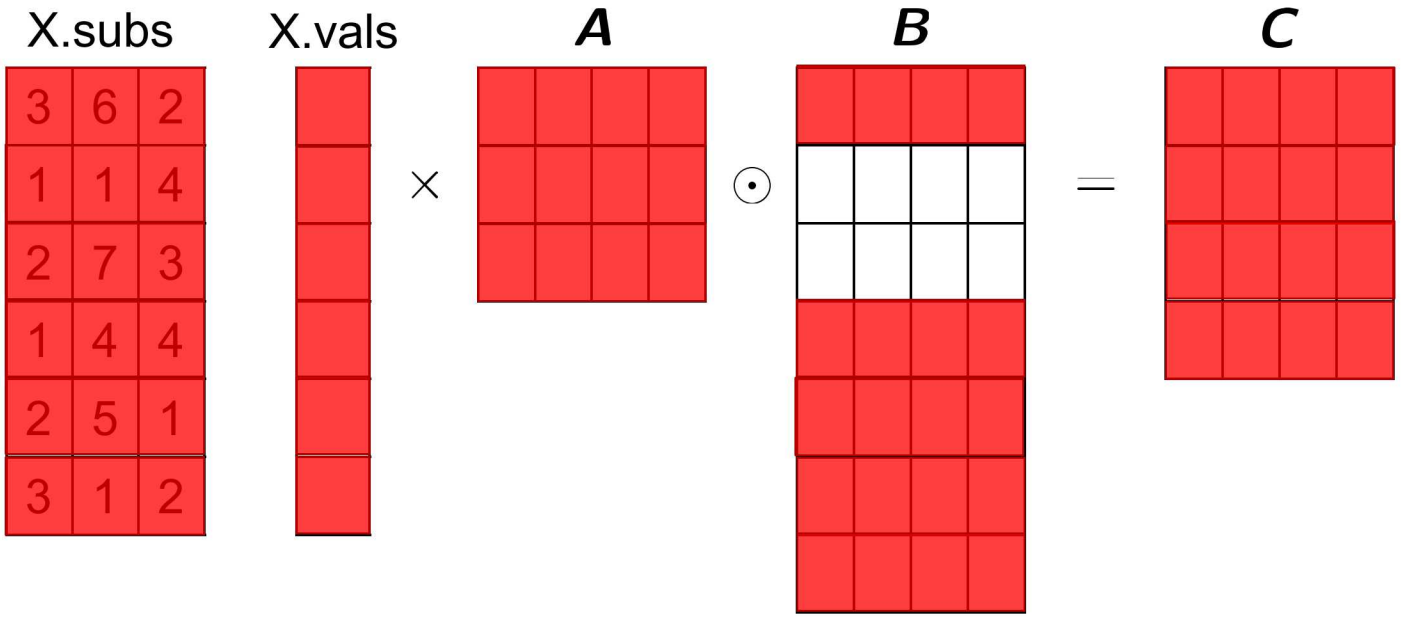


MTTKRP

- Sparse tensors stored in coordinate (COO) format
  - List of nonzero values and coordinates (subscripts)

$$\mathcal{X} \in \mathbb{R}^{3\times 7\times 4}, \quad R = 4$$

$$\boldsymbol{C} = \mathcal{X}_{(3)}(\boldsymbol{A} \odot \boldsymbol{B}) \implies c(k,l) = \sum_{(i,j,k)\in\mathcal{N}(\mathcal{X})} x(i,j,k)a(i,l)b(j,l), \quad l = 1,\ldots,R$$



Serial MTTKRP Algorithm

```
Sptensor X;
Ktensor u;
FacMatrix v;

const int nnz = X.nnz();
for (int i=0; i<nnz; ++i)
{
  double *tmp = new double[R];

  const size_t k = X.subscript(i,n);
  const double x_val = X.value(i);
  for (int j = 0; j < R; ++j)
    tmp[j] = x_val;

  for (int m = 0; m < N; m++) {
    if (m != n) {
      const double *row = u[m].rowptr(X.subscript(i,m));
      for (int j = 0; j < R; ++j)
        tmp[j] *= row[j];
    }
  }

  for (int j = 0; j < R; ++j)
    v.entry(k,j) += tmp[j];

  delete [] tmp;
}
```

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

- Factor matrices stored row-wise (Kokkos::LayoutRight)

$$\boldsymbol{V} = \boldsymbol{\mathcal{X}}_{(n)} \left(\boldsymbol{U}^N \odot \cdots \odot \boldsymbol{U}^{n+1} \odot \boldsymbol{U}^{n-1} \odot \cdots \odot \boldsymbol{U}^1 \right)$$

$$v(i_n, :) = \sum_{i \in \mathcal{N}(\boldsymbol{\mathcal{X}})} x_i u^1(i_1, :) \ldots u^{n-1}(i_{n-1}, :) u^{n+1}(i_{n+1}, :) \ldots u^N(i_N, :)$$

Flat Parallelism Over Tensor Non-zeros

Sandia National Laboratories

```
typedef Kokkos::RangePolicy<ExecSpace> Policy;
const int nnz = X.nnz();
Policy policy(0,nnz);
Kokkos::parallel_for( policy,
                      KOKKOS_LAMBDA(const int i)
{
  double *tmp = new double[R];

  const size_t k = X.subscript(i,n);
  const double x_val = X.value(i);

  for (int j = 0; j < R; ++j)
    tmp[j] = x_val;

  for (int m = 0; m < N; m++) {
    if (m != n) {
      const double *row = u[m].rowptr(X.subscript(i,m));
      for (int j = 0; j < R; ++j)
        tmp[j] *= row[j];
    }
  }

  for (int j = 0; j < R; ++j)
    Kokkos::atomic_add(&v.entry(k,j), tmp[j]);

  delete [] tmp;
});
```

Parallelize MTTKRP over tensor non-zeros

Multiple threads writing to same row of *V* requires atomic update (not the only way to do it)

Flat Parallelism Over Tensor Non-zeros

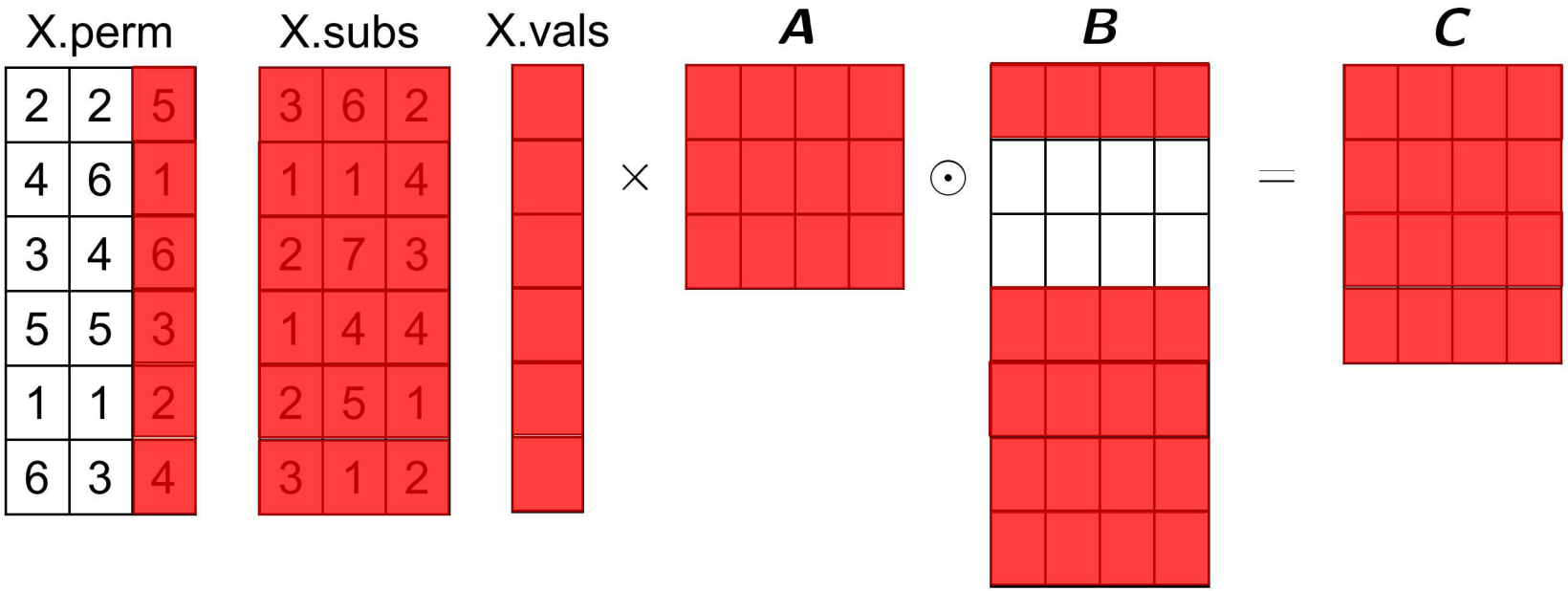
```
typedef Kokkos::RangePolicy<ExecSpace> Policy;
const int nnz = X.nnz();
Policy policy(0,nnz);
Kokkos::parallel_for( policy,
                KOKKOS_LAMBDA(const int i)
{
  double *tmp = new double[R];

  const size_t k = X.subscript(i,n);
  const double x_val = X.value(i);

  for (int j = 0; j < R; ++j)
    tmp[j] = x_val;

  for (int m = 0; m < N; m++) {
    if (m != n) {
      const double *row = u[m].rowptr(X.subscript(i,m));
      for (int j = 0; j < R; ++j)
        tmp[j] *= row[j];
    }
  }

  for (int j = 0; j < R; ++j)
    Kokkos::atomic_add(&v.entry(k,j), tmp[j]);

  delete [] tmp;
});
```

Improvements in MTTKRP implementation:

- Thread teams to process in parallel:
  - Tensor non-zeros
  - Factor matrix columns
- Thread-scalable mechanism for tmp buffer
  - Kokkos scratch-pad memory (MTTKRP-A)
  - Thread-local, compile-time polymorphic arrays (MTTKRP-B)
- Blocking on R for better cache performance, occupancy
- Alternatives to atomics
  - Thread privatization buffers

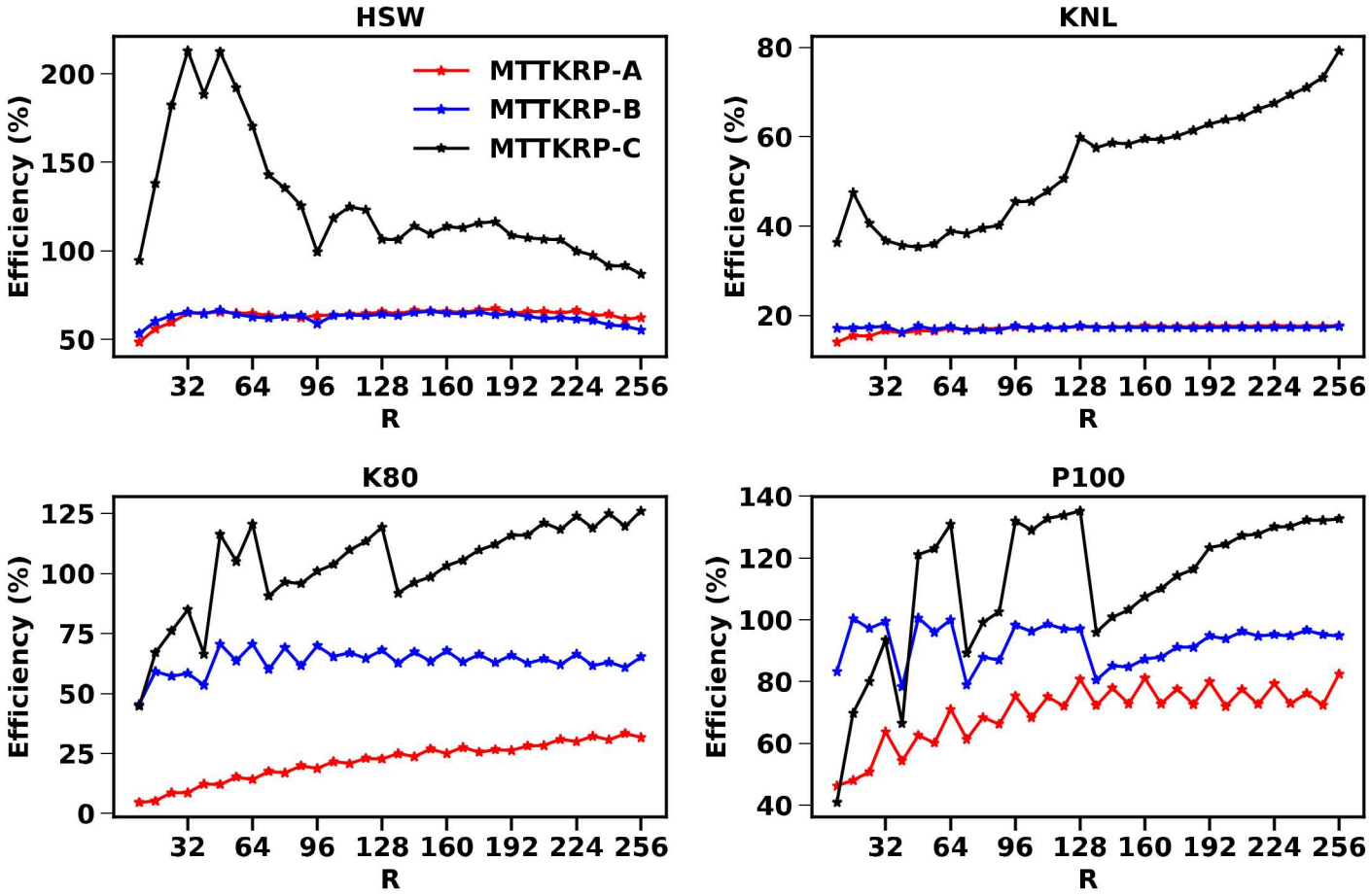
Permuted MTTKRP Inspired by COO Mat-Vec

- Poorly structured tensors result in high atomic contention
  - COO sparse matrix-vector product leverages sorting by increasing row-index
- Apply this to coordinate-based sparse tensor format
  - Don't want multiple copies of tensor with different sortings
  - Instead, compute permutation array p so that X.subscript(p(:,n),n) is increasing
  - Each thread only writes when row index X.subscript(p(:,n),n) changes
  - Substantially reduces number of atomic updates and improves their locality



MTTKRP Percentage of Peak Bandwidth

$$\text{GB/s} = \Big((NR + 3) \times sizeof(\text{Real}) + N \times sizeof(\text{Ordinal}) \Big) \times \frac{NNZ}{t}$$



- MTTKRP-A: Kokkos scratch-pad arrays
- MTTKRP-B: Compile-time polymorphic arrays
- MTTKRP-C: Permutation-based MTTKRP