Parallel Memory-Efficient Computation of Symmetric Higher-Order Joint Moment Tensors

Zitong Li $^1$  Hemanth Kolla $^2$  Eric Phipps $^2$

$^1$Wake Forest University

$^2$Sandia National Laboratories

March 1, 2022
The elementary-wise definition of the 4-th order moment tensor of a matrix \( \mathbf{X} \) of size \( r \times c \) \((r >> c)\) is the following:

\[
m_{i_1, i_2, i_3, i_4} = \mathbb{E}(\mathbf{X}_{i_1} \mathbf{X}_{i_2} \mathbf{X}_{i_3} \mathbf{X}_{i_4}) = \frac{1}{r} \sum_{j=1}^{r} \prod_{k=1}^{4} x_{j, i_k}
\]

For example, given a \( 2 \times 3 \) input matrix, the \((i, j, k, l)\) entry of its 4rd order moment tensor is:

\[
m_{i,j,k,l} = \frac{1}{2}(x_{1,i} x_{1,j} x_{1,k} x_{1,l} + x_{2,i} x_{2,j} x_{2,k} x_{2,l})
\]

The 4th order cumulant tensor can be defined as the following:

\[
c_{i_1, i_2, \ldots, i_d} = \mathbb{E}(\mathbf{X}_{i_1} \mathbf{X}_{i_2} \mathbf{X}_{i_3} \mathbf{X}_{i_4}) - \mathbb{E}(\mathbf{X}_{i_1} \mathbf{X}_{i_2}) \mathbb{E}(\mathbf{X}_{i_3} \mathbf{X}_{i_4}) - \mathbb{E}(\mathbf{X}_{i_1} \mathbf{X}_{i_3}) \mathbb{E}(\mathbf{X}_{i_2} \mathbf{X}_{i_4}) - \mathbb{E}(\mathbf{X}_{i_1} \mathbf{X}_{i_4}) \mathbb{E}(\mathbf{X}_{i_2} \mathbf{X}_{i_3})
\]
Application

- Higher order moment tensors contains important statistical information when the data is non-Gaussian.
- For example, anomaly detection with co-kurtosis tensor. Similar to PCA, we can do a tensor decomposition of the higher order cumulant tensor.
- Forming cumulant tensors explicitly and efficiently rely on forming moment tensors.

Figure: Principle components and 'principle kurtosis vectors'[1]
An order-3 tensor $X^{(3)}$ is supersymmetric if and only if

$$X_{i,j,k} = X_{i,k,j} = X_{k,i,j} = X_{k,j,i} = X_{j,k,i} = X_{j,i,k}$$

for any $i$, $j$, and $k$.

In general, given a element in a supersymmetric tensor with index $(i_1, ..., i_d)$, all the permutations of this index give you a different element with the same value.

Important property: There are at most $\binom{c+d-1}{d} \approx \frac{c^d}{d!}$ unique elements in a $d$-way supersymmetric tensor of size $c$. 
Figure: How a $4 \times 4 \times 4$ symmetric tensor is divided into 8 $2 \times 2 \times 2$ blocks, each in a different color. According to the Blocked Compact Symmetric Storage (BCSS) [2]

- Compared to storing/computing the whole tensor, using this method saves both memory and computation by a factor of $O(d!)$ for a $d$-way tensor.
- Domino et al. used this idea, computing each element in each unique block. This is the algorithm we will compare against.[3]
Our Contribution

- We refactor main computation so that it is more cache friendly and enabled us to use optimized kernels such as gemm in BLAS.
- We analyze the computation and cache complexity of this new algorithm and demonstrate speed up in the sequential setting.
- We implement a parallel version of our algorithm for shared-memory system in Kokkos, a programming model allowing performance portability across a wide range of HPC architectures.
A Different Way of Computing Moment Tensors

Denoting the vector outer product by \( \circ \) and the \( j^{th} \) row of \( \mathbf{X} \) by \( \mathbf{x}_j \). We can express the 4th order moment tensor of \( \mathbf{X} \in \mathbb{R}^{r \times c} \) by:

\[
\mathbf{M} = \frac{1}{r} \sum_{j=1}^{r} \mathbf{x}_j \circ \mathbf{x}_j \circ \mathbf{x}_j \circ \mathbf{x}_j
\]  

(1)

Looking at it this way, the problem is nearly identical to forming the full tensor from factor matrices \( \{\mathbf{X}^T, \mathbf{X}^T, \mathbf{X}^T, \mathbf{X}^T\} \) of a CP decomposition. Let \( \text{Mat}_2(\mathbf{M}) \) denote \( \mathbf{M} \) unfolded on the first 2 modes, we have:

\[
\mathbf{M}_{(1:2)} = \text{Mat}_2(\mathbf{M}) = \frac{1}{r} (\mathbf{X}^T \circ \mathbf{X}^T)(\mathbf{X}^T \circ \mathbf{X}^T)^T
\]  

(2)
To compute a certain block in the 4th order moment tensor of $X$, we use the following formula. This allows us to take advantage of the symmetry.

$$Mat_2(M^B_{(i,j,k,l)}) = \frac{1}{r} (X_i^T \odot X_j^T)(X_k^T \odot X_l^T)^T$$  \hspace{1cm} (3)
Complexity for computing a $d$-th order moment tensor in total is:

$$
\left( \frac{2rs^d}{2} + (2r - 1)s^d \right) \binom{(c/s) + d - 1}{d}.
$$

- Most of the computation goes into the \texttt{gemm} call.
- The KhatriRao product is better than elementary-wise in terms of cache efficiency because of less cache misses.
Results collected using an AMD EPYC 7302 CPU.
Kokkos exposes 3 levels of parallelisms: team, thread, and vector.
Each team, corresponding to a warp in GPU, computes several blocks.
We use a tiling strategy to limit the memory usage of each team.
**Algorithm** Parallel Algorithm for computing the 4th moment tensor

1: function $\mathcal{M}^{(4)} = 4th\text{MomentTensor}(\mathbf{X}, b, t, s)$
2: $\triangleright s =$ block size, $t =$ tile size, $n =$ # of teams, $\mathbf{X} \in \mathbb{R}^{r \times c}$
3: $\bar{r} = r / t$ $\triangleright$ # of row tiles
4: $n bm = \lceil c / s \rceil$ $\triangleright$ # of blocks on each mode
5: $n b = \binom{n bm + 3}{4}$ $\triangleright$ total # of unique blocks in $\mathcal{M}^{(4)}$
6: $\overline{n b} = n b / n$ $\triangleright$ # of blocks each team computes
7: $\text{teams parallel for}(i = 1, \ldots, n)$
8: $(i, \overline{n b}) \mapsto \{b_s, \ldots, b_e\}$
9: for $j = b_s, \ldots, b_e$ do
10: $(i_1, i_2, i_3, i_4) = \mathbb{T}(j; nbm)$ $\triangleright$ multi-index this block
11: for $k = 1, \ldots, \bar{r}$ do
12: $\mathbf{Y} = \text{KhatriRaoProduct}(\mathbf{X}^{T}_{i_1}, \mathbf{X}^{T}_{i_2})$
13: $\mathbf{Z} = \text{KhatriRaoProduct}(\mathbf{X}^{T}_{i_3}, \mathbf{X}^{T}_{i_4})$
14: $\mathcal{M}^{B}_{(i_1, i_2, i_3, i_4)} += \frac{1}{\bar{r}} \text{team_gemm} (\mathbf{Y}, \mathbf{Z}^{T})$
Algorithm Parallel Algorithm for the Khatri-Rao Product

1: function C = KhatriRaoProduct(A, B)  
2: ▷ A ∈ \mathbb{R}^{a \times c}, B ∈ \mathbb{R}^{b \times c}  
3:  threads_parallel_for(i from 1 to ab)  
4:     vector_parallel_for(j from 1 to c)  
5:     (\alpha, \beta) = T(i; a, b)  
6:     C_{i,j} = X_{\alpha,j}X_{\beta,j}
Parallel Performance

- Results recorded on a Nvidia Volta V100 GPU.

- Figure: # of rows: $10^7$, # of columns: 60

- Figure: block size: 5, # of rows: $10^7$

- Figure: block size: 5, # of columns: 30
Cache Performance

- Input size: $10^6 \times 30$
Future Work

- Examine how performance portable our implementation is.
- Extend this result for computing cumulant tensor.
Thank you!

Questions?

- email: liz20@wfu.edu
Anomaly detection in scientific data using joint statistical moments.  

Exploiting Symmetry in Tensors for High Performance: Multiplication with Symmetric Tensors.  

Efficient Computation of Higher-Order Cumulant Tensors.  
Publisher: Society for Industrial and Applied Mathematics.