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

Moment Tensor

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 2×3 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:

$$\begin{aligned} c_{i_1,i_2,\ldots,i_d} =& \mathbb{E}(\mathbf{X}_{i_1}\mathbf{X}_{i_2}\mathbf{X}_{i_3}\mathbf{X}_{i_4}) - \mathbb{E}(\mathbf{X}_{i_1}\mathbf{X}_{i_2})\mathbb{E}(\mathbf{X}_{i_3}\mathbf{X}_{i_4}) \\ & - \mathbb{E}(\mathbf{X}_{i_1}\mathbf{X}_{i_3})\mathbb{E}(\mathbf{X}_{i_2}\mathbf{X}_{i_4}) - \mathbb{E}(\mathbf{X}_{i_1}\mathbf{X}_{i_4})\mathbb{E}(\mathbf{X}_{i_2}\mathbf{X}_{i_3}) \end{aligned}$$

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.

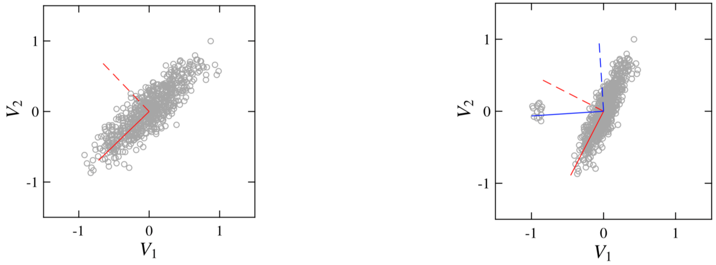


Figure: Principle components and 'principle kurtosis vecotrs'[1]

Supersymmetric Tensor

- An order-3 tensor $\mathfrak{X}^{(3)}$ 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.

Leveraging the Symmetry

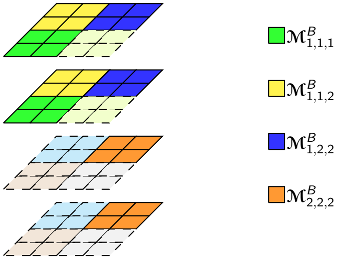


Figure: How a $4 \times 4 \times 4$ symmetric tensor is divided into 8 $2 \times 2 \times 2$ blocks, each in a different color. According to the Blocked Compact Symmetric Storage (BCSS) [2]

- 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

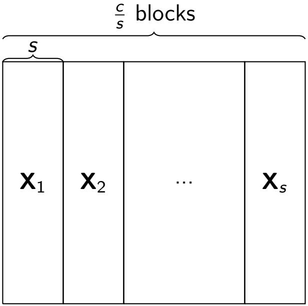
Denoting the vector outer product by $\circ$ and the j^{th} row of $\mathbf{X}$ by $\boldsymbol{x}_j$. We can express the 4th order moment tensor of $\mathbf{X} \in \mathbb{R}^{r \times c}$ by:

$$\mathcal{M} = \frac{1}{r} \sum_{j=1}^{r} \boldsymbol{x}_j \circ \boldsymbol{x}_j \circ \boldsymbol{x}_j \circ \boldsymbol{x}_j \tag{1}$$

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 $Mat_2(\mathcal{M})$ denote $\mathcal{M}$ unfolded on the first 2 modes, we have:

$$\mathcal{M}_{(1:2)} = Mat_2(\mathcal{M}) = \frac{1}{r}(\mathbf{X}^T \odot \mathbf{X}^T)(\mathbf{X}^T \odot \mathbf{X}^T)^T \tag{2}$$

Blocked Algorithm



- Break $\mathbf{X} \in \mathbb{R}^{r \times c}$ into block columns as shown on the left

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.

$$Mat_2(\mathcal{M}^B_{(i,j,k,l)}) = \frac{1}{r}(\mathbf{X}_i^T \odot \mathbf{X}_j^T)(\mathbf{X}_k^T \odot \mathbf{X}_l^T)^T \tag{3}$$

Computation/Cache Complexity Analysis

- Complexity for computing a d-th order moment tensor in total is:

$$\Big(\underbrace{2rs^{d/2}}_{\text{Khatri-Rao products}} + \underbrace{(2r-1)s^{d}}_{\texttt{gemm}} \Big) \underbrace{\binom{(c/s)+d-1}{d}}_{\#\text{ of blocks}} .$$

- Most of the computation goes into the gemm call.
- The KhatriRao product is better than elementary-wise in terms of cache efficiency because of less cache misses.

Sequential Performance

- Results collected using a AMD EPYC 7302 CPU.

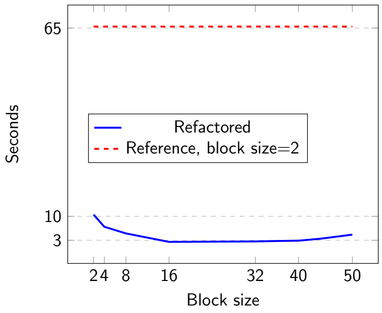


Figure: block size: 2, # of columns: 50

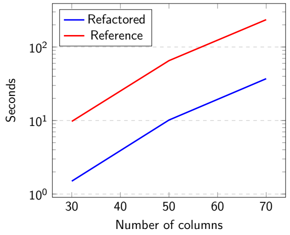


Figure: block size: 2, # of rows: 80,000

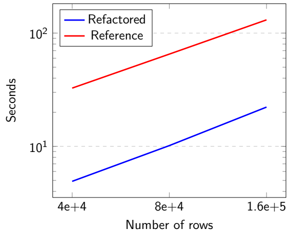
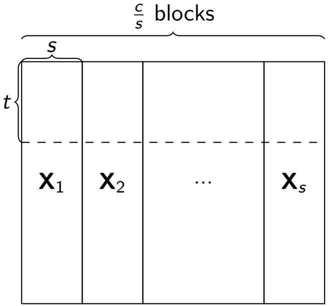


Figure: block size: 2, # of columns: 50

Parallel Algorithm

- 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.



Parallel Algorithm

Algorithm Parallel Algorithm for computing the 4th moment tensor

function $\mathcal{M}^{(4)} = \textsc{4thMomentTensor}(\mathbf{X}, b, t, s)$

- $\triangleright$ $s =$ block size, $t =$ tile size, $n = \#$ of teams, $\mathbf{X} \in \mathbb{R}^{r \times c}$
- $\bar{r} = r/t$ $\triangleright$ # of row tiles
- $nbm = ceil(c/s)$ $\triangleright$ # of blocks on each mode
- $nb = \binom{nbm+3}{4}$ $\triangleright$ total # of unique blocks in $\mathcal{M}^{(4)}$
- $\overline{nb} = nb/n$ $\triangleright$ # of blocks each team computes
- **teams_parallel_for**$(i = 1, \dots, n)$
  - $(i, \overline{nb}) \mapsto \{b_s, \dots, b_e\}$ $\triangleright$ block index range of this team
  - **for** $j = b_s, \dots, b_e$ **do**
    - $(i_1, i_2, i_3, i_4) = \mathbb{T}(j; nbm)$ $\triangleright$ multi-index this block
    - **for** $k = 1, \dots, \bar{r}$ **do**
      - $\mathbf{Y} = \textsc{KhatriRaoProduct}(\mathbf{X}_{i_1}^T, \mathbf{X}_{i_2}^T)$
      - $\mathbf{Z} = \textsc{KhatriRaoProduct}(\mathbf{X}_{i_3}^T, \mathbf{X}_{i_4}^T)$
      - $\mathcal{M}^B_{(i_1,i_2,i_3,i_4)} += \frac{1}{r}\text{team\_gemm}(\mathbf{Y}, \mathbf{Z}^T)$

Parallel Algorithm

Algorithm Parallel Algorithm for the Khatri-Rao Product

function $\mathbf{C} = \textsc{KhatriRaoProduct}(\mathbf{A}, \mathbf{B})$
  $\triangleright$ $\mathbf{A} \in \mathbb{R}^{a \times c}$, $\mathbf{B} \in \mathbb{R}^{b \times c}$
  threads_parallel_for(i from 1 to ab)
    vector_parallel_for(j from 1 to c)
      $(\alpha, \beta) = \mathbb{T}(i; a, b)$
      $\mathbf{C}_{i,j} = \mathbf{X}_{\alpha,j}\mathbf{X}_{\beta,j}$

Parallel Performance

- Results recorded on a Nvidia Volta V100 GPU.

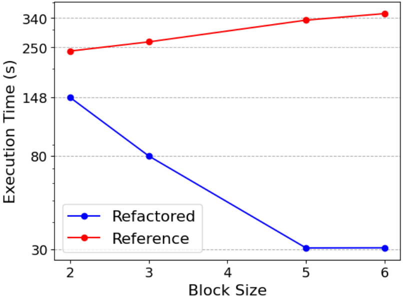


Figure: # of rows: 10^7, # of columns: 60

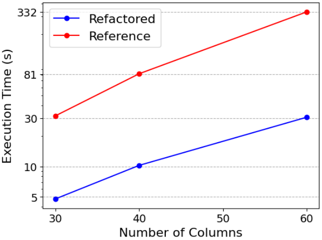


Figure: block size: 5, # of rows: 10^7

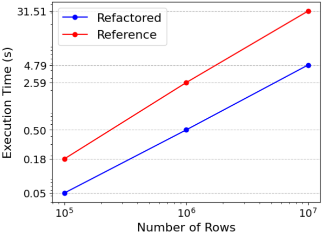
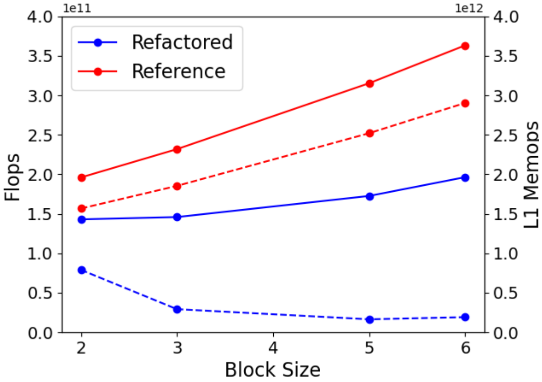


Figure: block size: 5, # of columns: 30

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

References

[1] Konduri Aditya, Hemanth Kolla, W. Philip Kegelmeyer, Timothy M. Shead, Julia Ling, and Warren L. Davis.
Anomaly detection in scientific data using joint statistical moments.
Journal of Computational Physics, 387:522–538, June 2019.

[2] Martin D. Schatz, Tze Meng Low, Robert A. van de Geijn, and Tamara G. Kolda.
Exploiting Symmetry in Tensors for High Performance: Multiplication with Symmetric Tensors.
SIAM Journal on Scientific Computing, 36(5):C453–C479, January 2014.

[3] Krzysztof Domino, Piotr Gawron, and Lukasz Pawela.
Efficient Computation of Higher-Order Cumulant Tensors.
SIAM Journal on Scientific Computing, 40(3):A1590–A1610, January 2018.
Publisher: Society for Industrial and Applied Mathematics.