High Performance Hierarchical Tucker Tensor Learning Using GPU Tensor Cores

Hao Huang[1], Xiao-Yang Liu[2], Weiqin Tong[1], Tao Zhang[1], Anwar Walid[2]

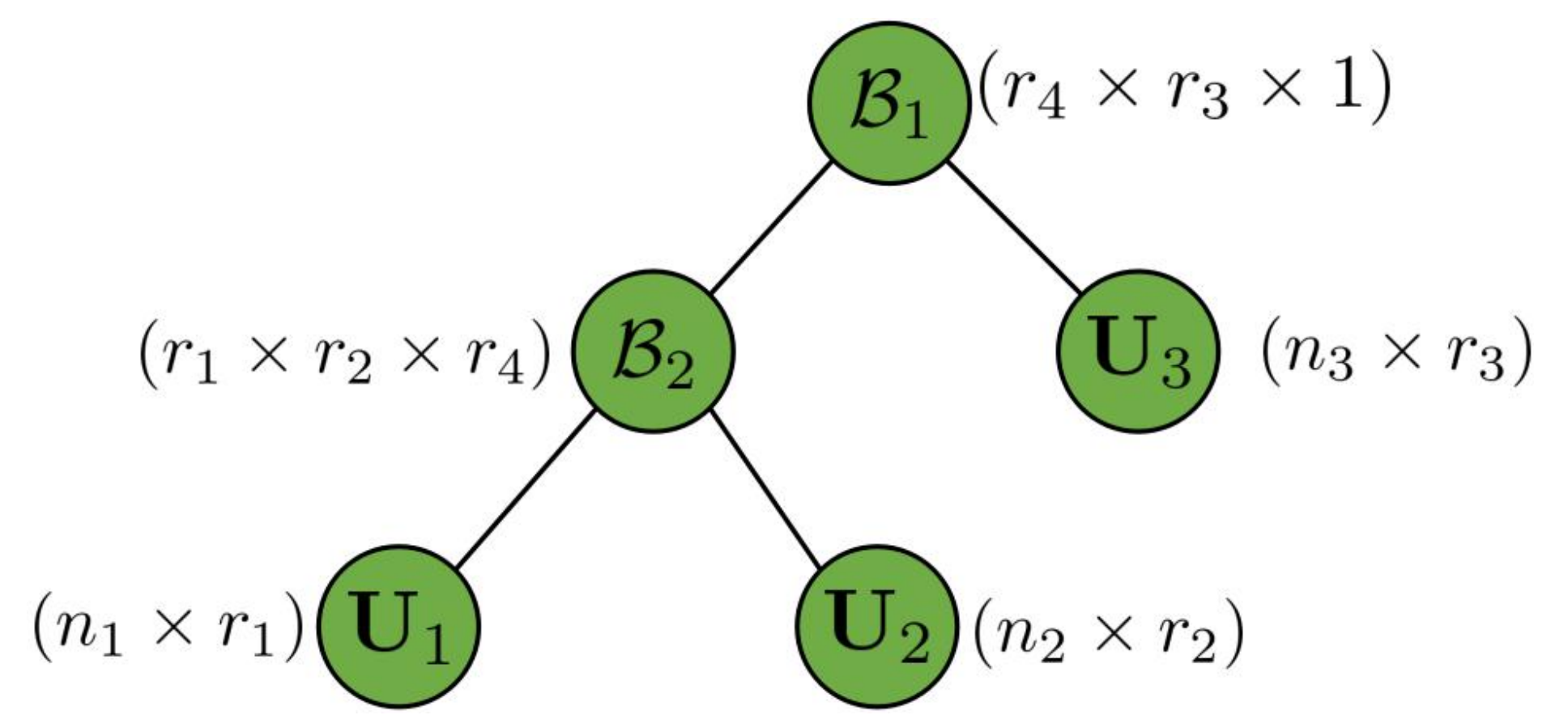
[1]Shanghai University, Shanghai, China; [2] Columbia University, NewYork, USA.

Introduction

In many real-world applications, data are usually represented as Hierarchical Tucker (HT) tensor model, such as big data analysis, neural network compression and quantum machine learning. However, it is compute intensive due to the time complexity grows exponentially with the order of the data tensor. In this work, we present efficient HT tensor learning primitives using GPU tensor cores.

Hierarchical Tucker Decomposition

The HT format is stored in the form of a binary tree T, where each branch is a hierarchical division of the tensor mode set.


$\mathcal{B}_1$ $(r_4 \times r_3 \times 1)$
$(r_1 \times r_2 \times r_4)$ $\mathcal{B}_2$
$\mathbf{U}_3$ $(n_3 \times r_3)$
$(n_1 \times r_1)$ $\mathbf{U}_1$
$\mathbf{U}_2$ $(n_2 \times r_2)$

Parallel HT Decomposition on the GPU

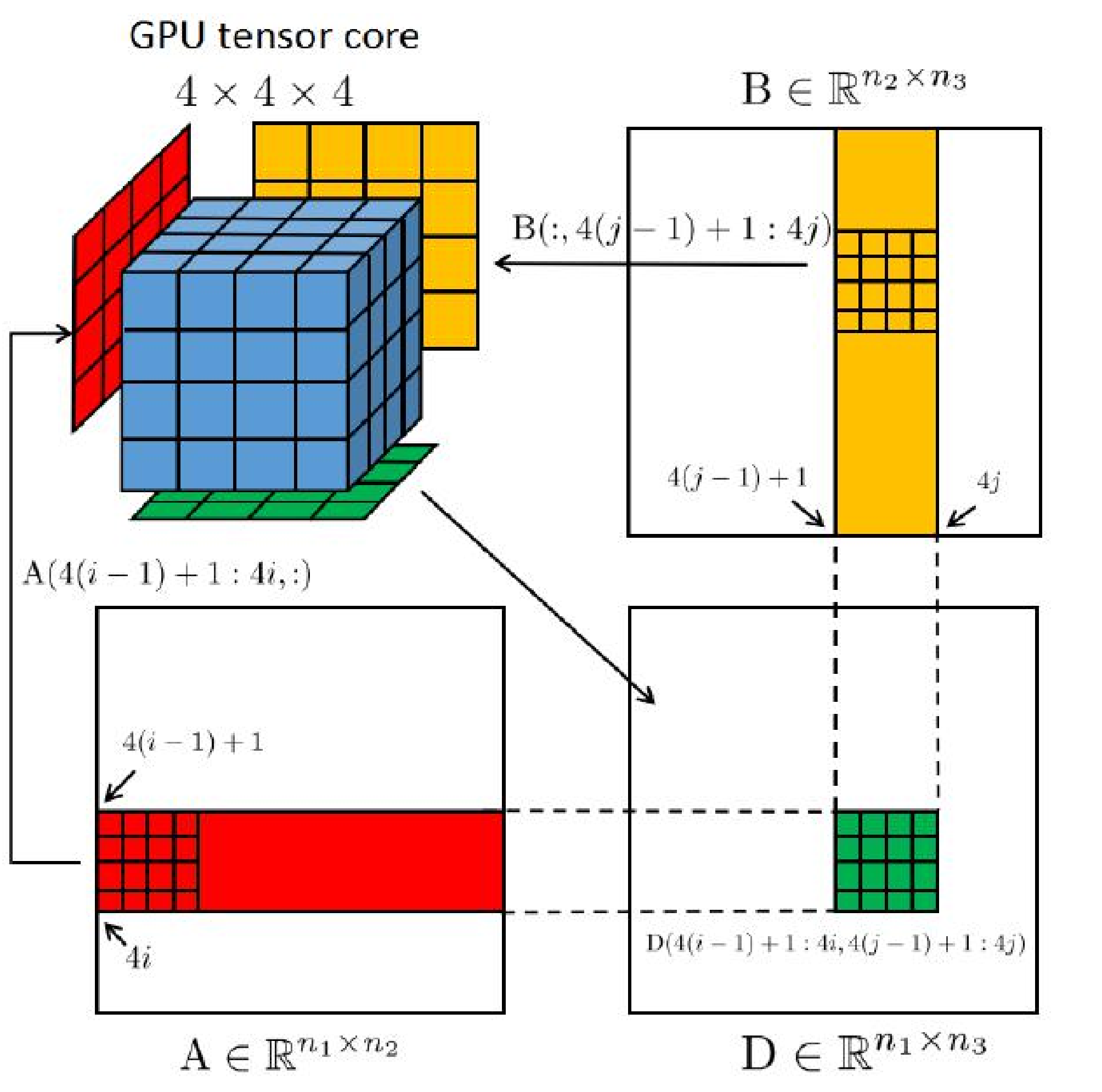
- **Overview of the algorithm**

Algorithm 2 Optimized hierarchical Tucker tensor decomposition using GPU tensor cores

```
1:  Input: Tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, ranks $r_1, r_2, r_3, r_4$.
2:  for $i = 1$ to 3 do in parallel
3:      $\boldsymbol{H}_i = \boldsymbol{X}_{(i)} \boldsymbol{X}_{(i)}^T$,
4:  end for
5:  for $i = 1$ to 3 do in parallel
6:      $\boldsymbol{U}_i \leftarrow$($r_i$ leading eigen vectors) of $\boldsymbol{H}_i$,
7:  end for
8:  $\boldsymbol{U}_4 \leftarrow$($r_4$ leading left singular vectors) of $\boldsymbol{X}_{(3)}^T$ using the
    TSQR algorithm,
9:  $\mathcal{B}_2 = \mathcal{U}_4 \times_1 \boldsymbol{U}_1^T \times_2 \boldsymbol{U}_2^T$,
10: $\mathcal{B}_1 = \mathcal{X} \times_1 \boldsymbol{U}_4^T \times_2 \boldsymbol{U}_3^T$,
11: Output: $\mathcal{B}_1 \in \mathbb{R}^{r_4 \times r_3 \times 1}, \mathcal{B}_2 \in \mathbb{R}^{r_1 \times r_2 \times r_4}, \boldsymbol{U}_3 \in$
    $\mathbb{R}^{n_3 \times r_3}, \boldsymbol{U}_2 \in \mathbb{R}^{n_2 \times r_2}, \boldsymbol{U}_1 \in \mathbb{R}^{n_1 \times r_1}$.
```

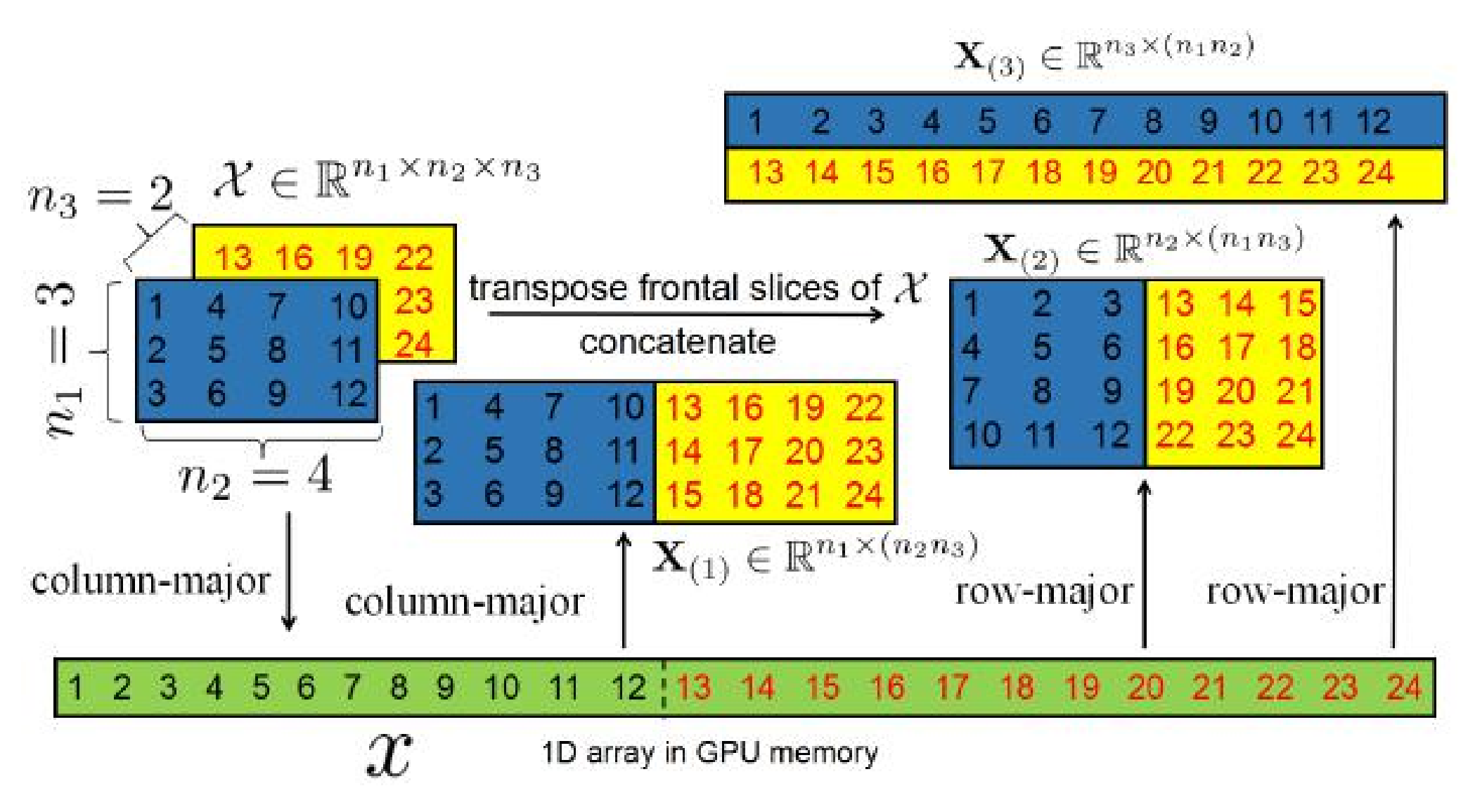
- **Matrix Multiplication Using GPU Tensor Cores**

We schedule GPU tensor cores to optimize matrix multiplication.


GPU tensor core
$4 \times 4 \times 4$
$\mathrm{B} \in \mathbb{R}^{n_2 \times n_3}$
$\mathrm{B}(:, 4(j-1)+1 : 4j)$
$4(j-1)+1$
$4j$
$\mathrm{A}(4(i-1)+1 : 4i, :)$
$4(i-1)+1$
$4i$
$\mathrm{D}(4(i-1)+1 : 4i, 4(j-1)+1 : 4j)$
$\mathrm{A} \in \mathbb{R}^{n_1 \times n_2}$
$\mathrm{D} \in \mathbb{R}^{n_1 \times n_3}$

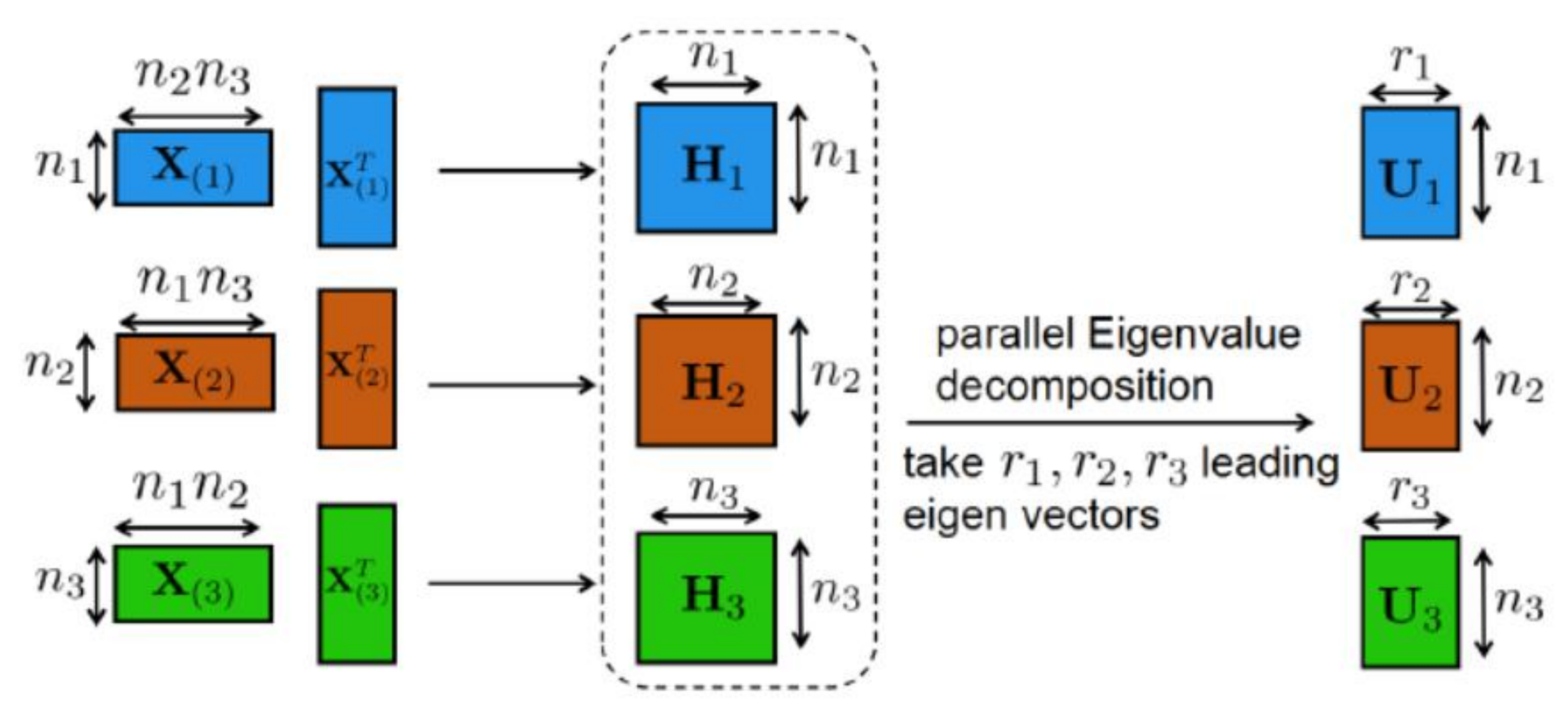
- **Matricization-free Memory Access**

We use a matricization-free memory access and avoid consumptions of conversions between tensors and matrices.


$n_3 = 2$
$\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$
$n_1 = 3$
$n_2 = 4$
$\mathbf{X}_{(3)} \in \mathbb{R}^{n_3 \times (n_1 n_2)}$
$\mathbf{X}_{(2)} \in \mathbb{R}^{n_2 \times (n_1 n_3)}$
$\mathbf{X}_{(1)} \in \mathbb{R}^{n_1 \times (n_2 n_3)}$
transpose frontal slices of $\mathcal{X}$
concatenate
column-major
column-major
row-major
row-major
$\mathcal{X}$
1D array in GPU memory

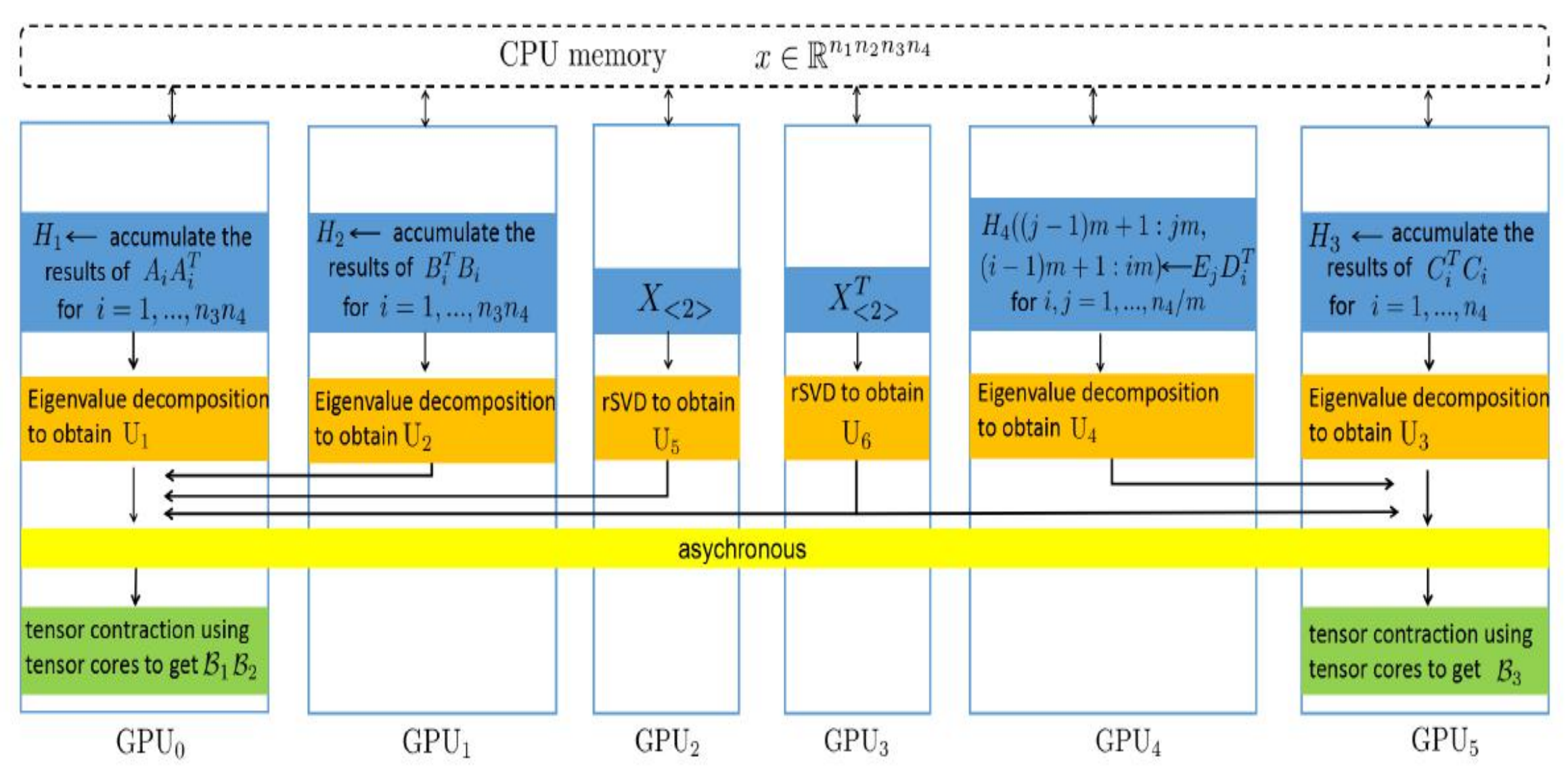
- **Batch operation**

Use Eigenvalue decomposition instead of SVD and take the batch solution to calculate intermediate variables in parallel.

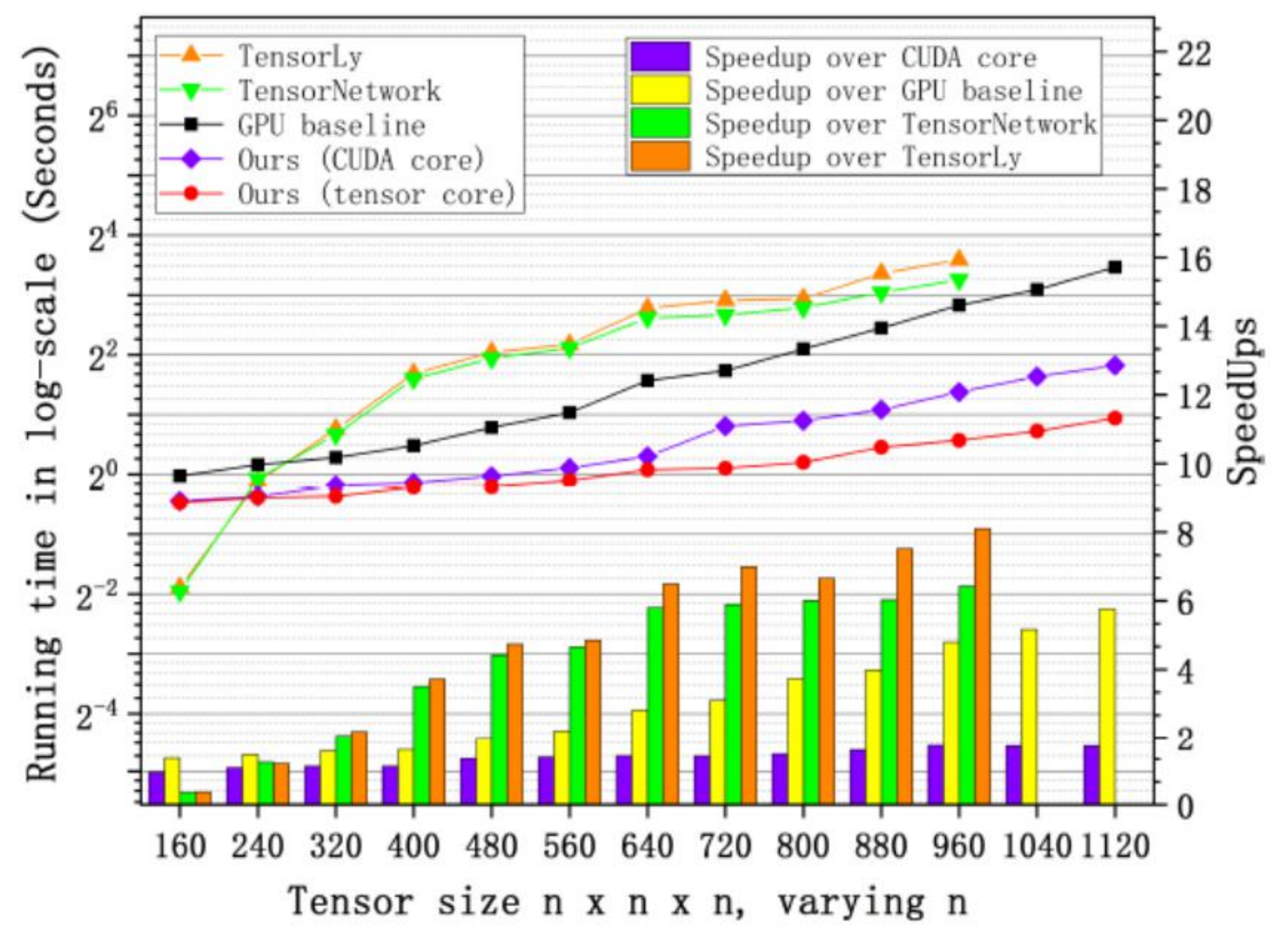

$n_2 n_3$
$n_1 n_3$
$n_1 n_2$
$\mathbf{X}_{(1)}$ $\mathbf{X}_{(1)}^T$
$\mathbf{X}_{(2)}$ $\mathbf{X}_{(2)}^T$
$\mathbf{X}_{(3)}$ $\mathbf{X}_{(3)}^T$
$\mathbf{H}_1$ $\mathbf{H}_2$ $\mathbf{H}_3$
parallel Eigenvalue decomposition
take r_1, r_2, r_3 leading eigen vectors
$\mathbf{U}_1$ $\mathbf{U}_2$ $\mathbf{U}_3$

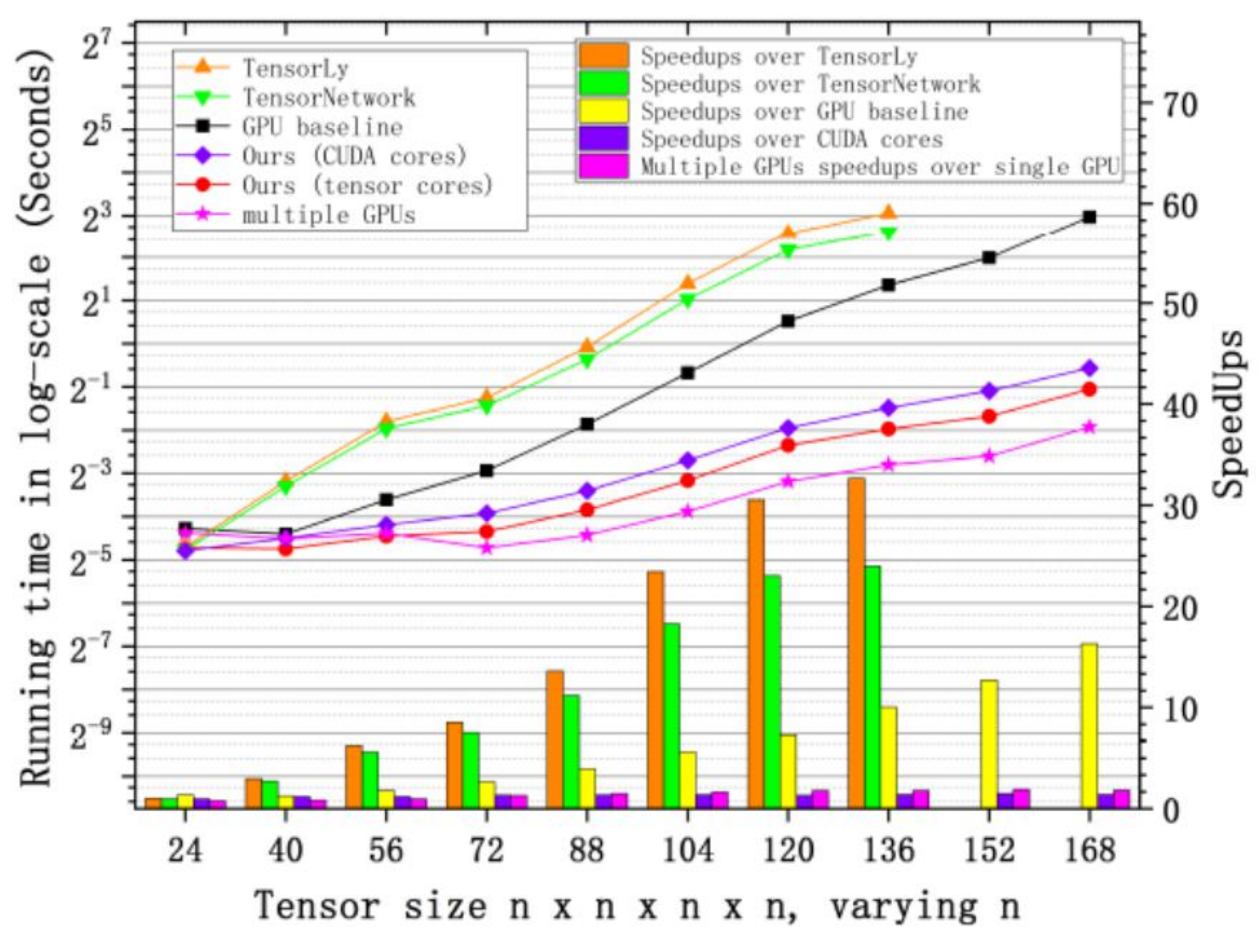
- **Shard mode on multiple GPUs**

High-order HT tensor decomposition on multiple GPUs using shard mode. We divide the original input tensor to shards and transfer each shard to different GPU memory.


CPU memory $x \in \mathbb{R}^{n_1 n_2 n_3 n_4}$
$H_1 \leftarrow$ accumulate the results of $A_i A_i^T$ for $i = 1, ..., n_3 n_4$
$H_2 \leftarrow$ accumulate the results of $B_i^T B_i$ for $i = 1, ..., n_3 n_4$
$X_{<2>}$
$X_{<2>}^T$
$H_4((j-1)m+1 : jm, (i-1)m+1 : im) \leftarrow E_j D_i^T$ for $i, j = 1, ..., n_4/m$
$H_3 \leftarrow$ accumulate the results of $C_i^T C_i$ for $i = 1, ..., n_4$
Eigenvalue decomposition to obtain U_1
Eigenvalue decomposition to obtain U_2
rSVD to obtain U_5
rSVD to obtain U_6
Eigenvalue decomposition to obtain U_4
Eigenvalue decomposition to obtain U_3
asynchronous
tensor contraction using tensor cores to get $\mathcal{B}_1 \mathcal{B}_2$
tensor contraction using tensor cores to get $\mathcal{B}_3$
GPU0 GPU1 GPU2 GPU3 GPU4 GPU5

- **Experimental results:**

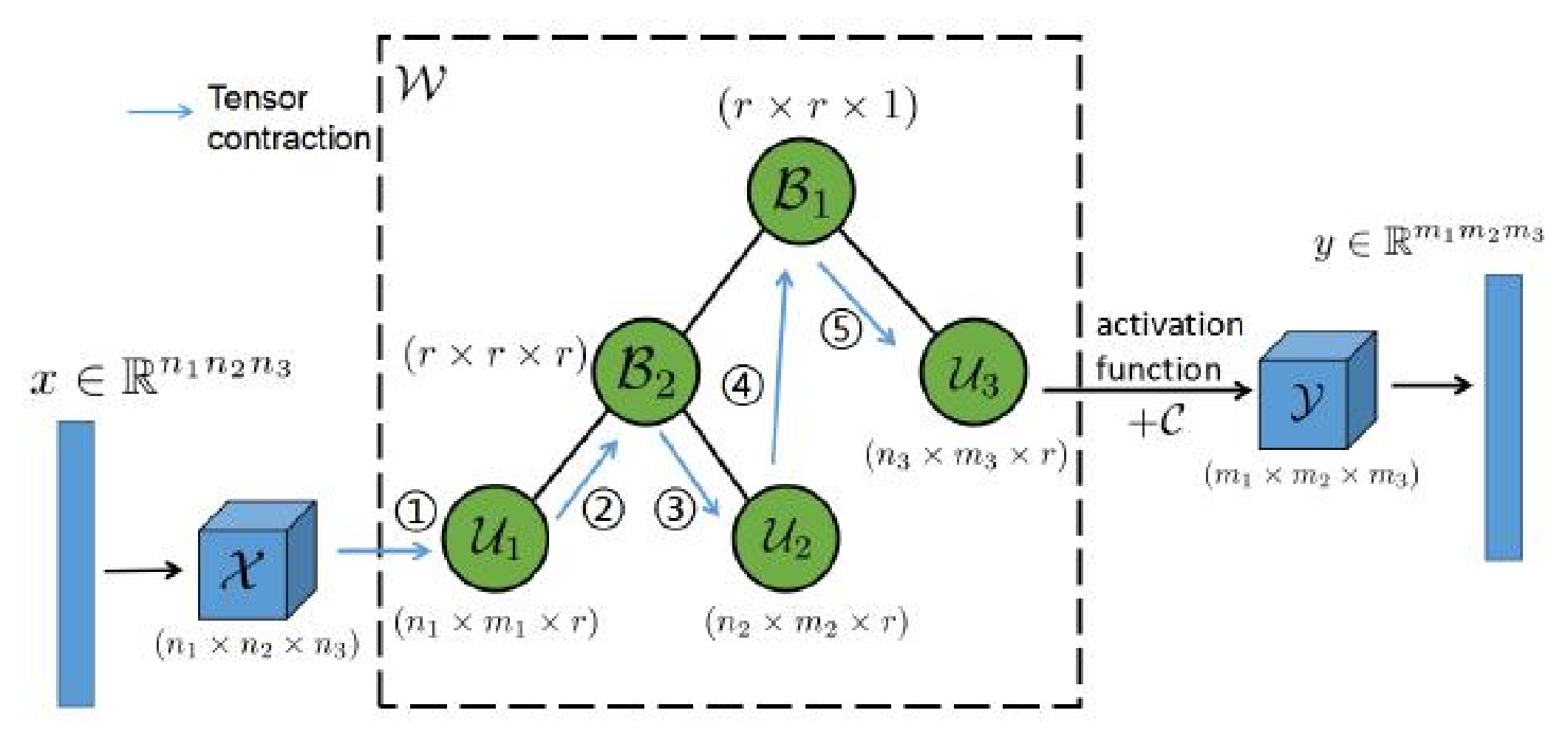




HT Tensor Layer for Deep Neural Networks Compression

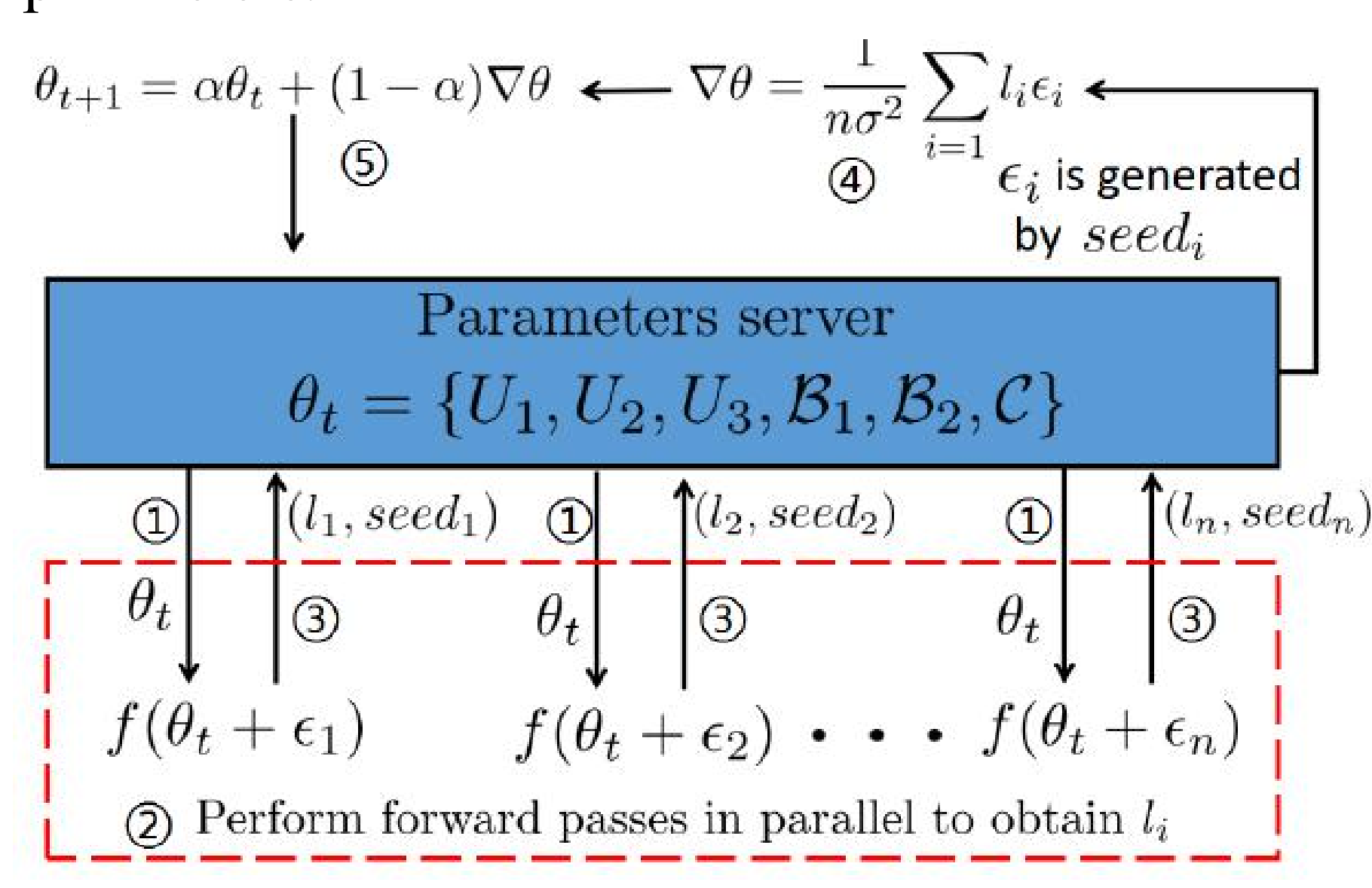
- **forward pass**

For forward pass, we use a HT tensor layer to represent W in a HT format and perform optimized tensor contractions.

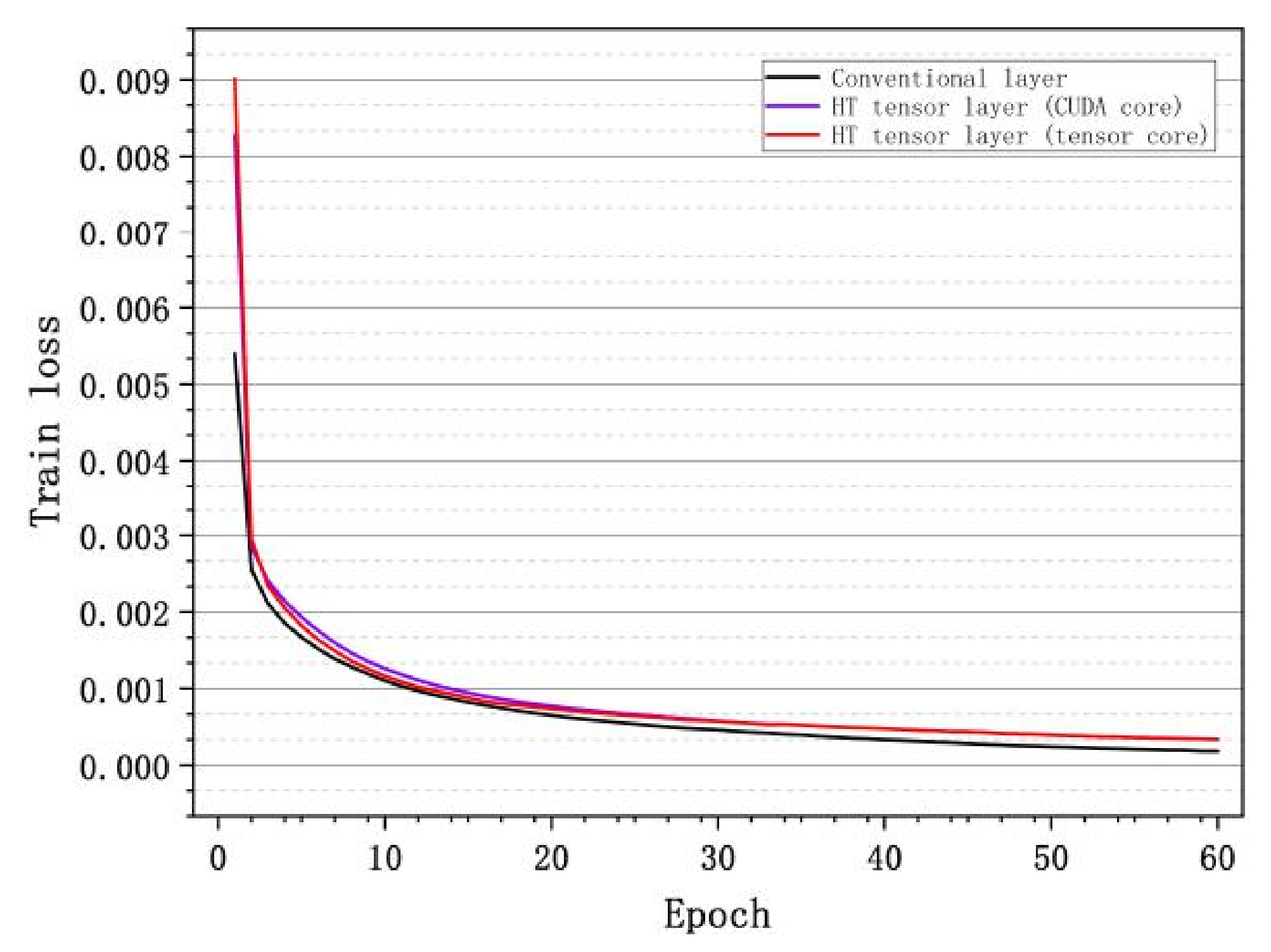


- **Parameters update process**

Use natural gradients to compute in parallel and update the parameters.


$\theta_{t+1} = \alpha\theta_t + (1-\alpha)\nabla\theta$
$\nabla\theta = \frac{1}{n\sigma^2}\sum_{i=1} l_i \epsilon_i$
ϵ_i is generated by $seed_i$
Parameters server
$\theta_t = \{U_1, U_2, U_3, \mathcal{B}_1, \mathcal{B}_2, \mathcal{C}\}$
$(l_1, seed_1)$
$(l_2, seed_2)$
$(l_n, seed_n)$
θ_t
$f(\theta_t + \epsilon_1)$
$f(\theta_t + \epsilon_2)$
$f(\theta_t + \epsilon_n)$
② Perform forward passes in parallel to obtain l_i

- **Experimental results:**

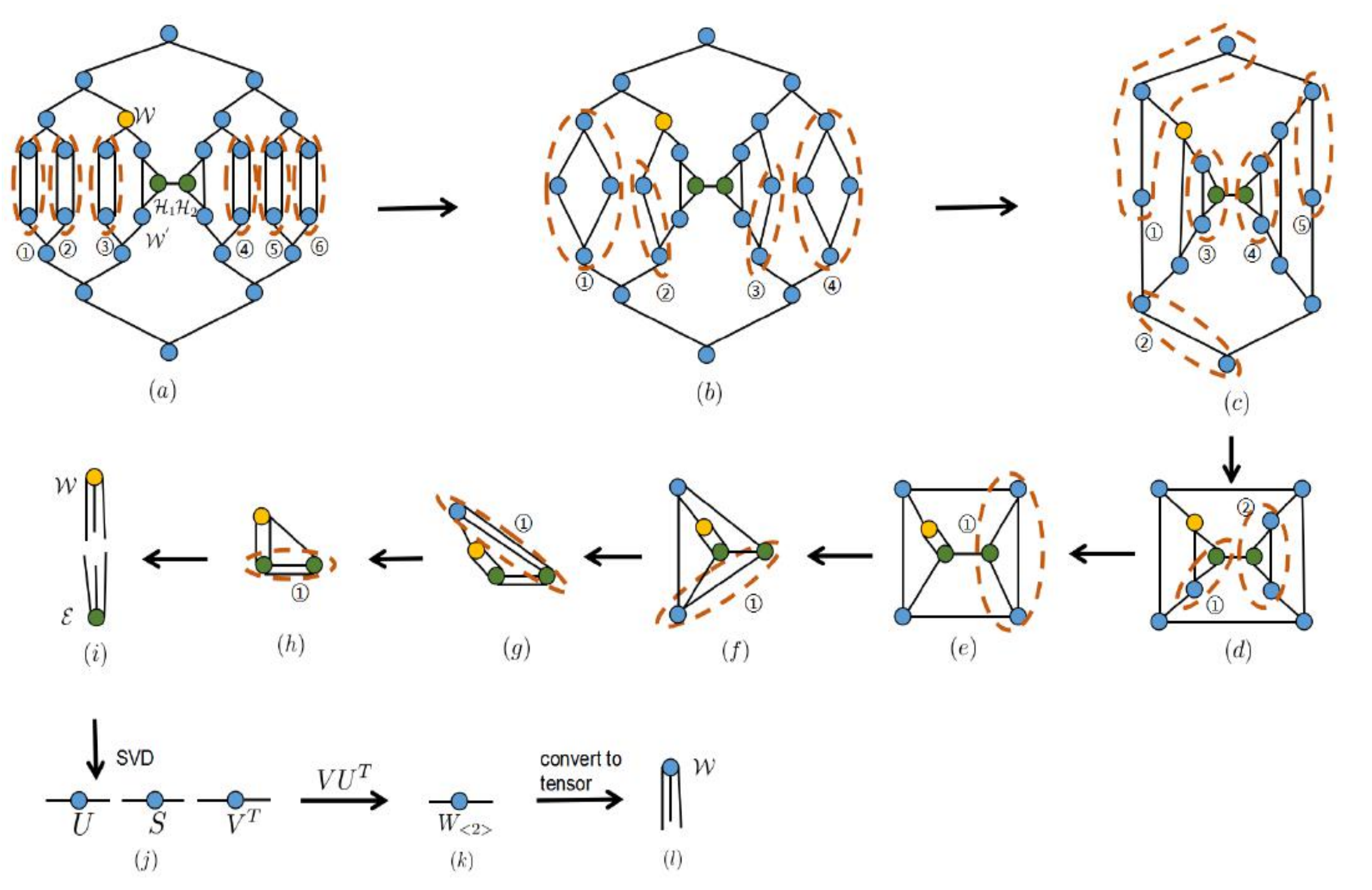


Methods	Parameters	Accuracy	Running time (s)
Conventional layer	1, 071, 370	98.8%	38, 306
HT tensor layer (CUDA cores)	21, 770 (49.2×)	98.3%	102, 705
HT tensor layer (tensor cores) **(ours)**	21, 770 (49.2×)	98.3%	72, 327

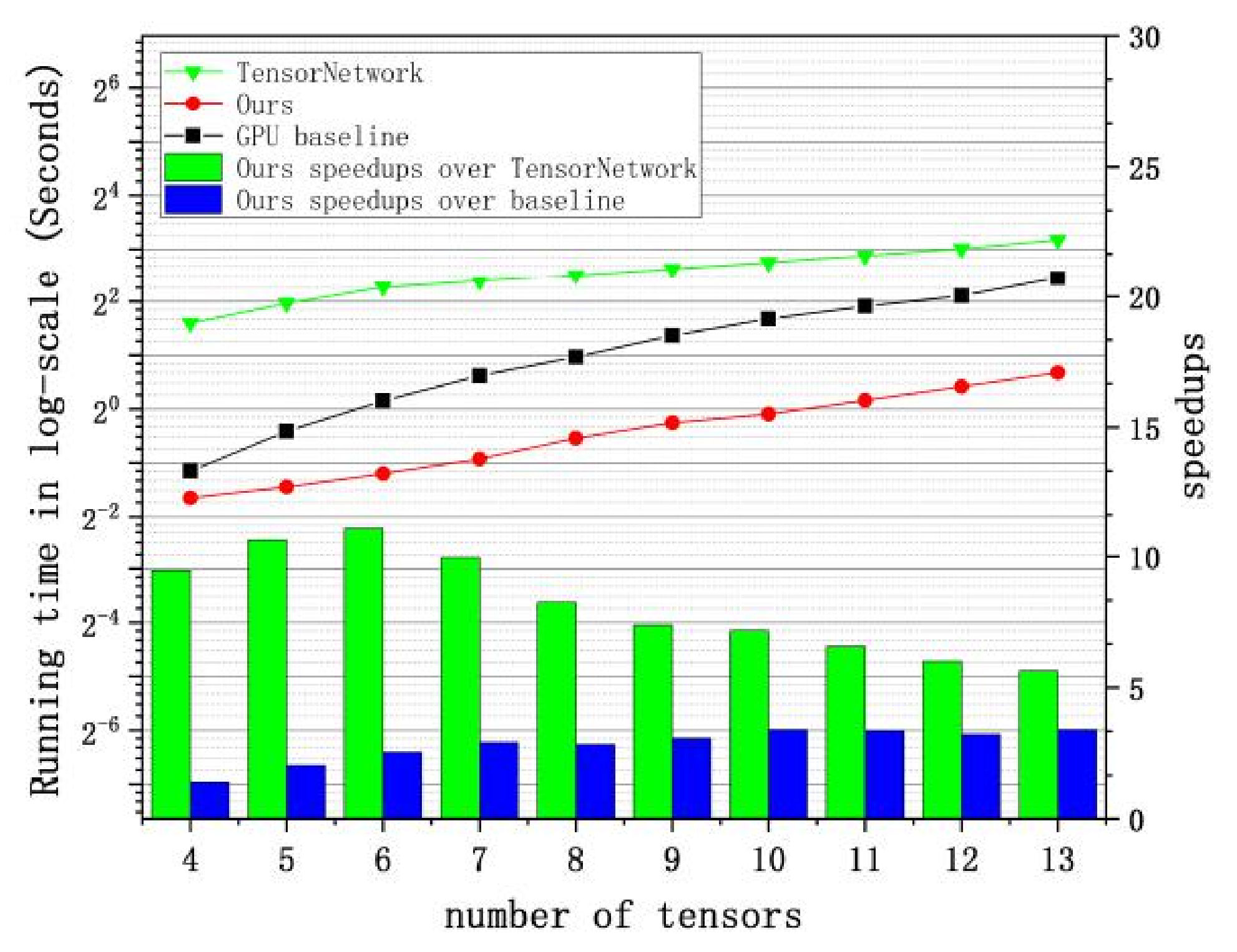
Efficient TTN Algorithm Using GPU Tensor Cores

- **Algorithm optimization:**

TTN algorithm consists of multiple tensor contractions, and we optimize thealgorithm according to the parallelism.



- **Experimental results:**



Conclusions

In this work, we implement the optimized tensor learning primitives into HT decompositions, a HT tensor layer for deep neural network and TTN algorithm. Experimental results demonstrated that our optimized algorithms have higher efficiency.

[1] H. Huang, X. Liu, W. Tong, T. Zhang and A. Walid. High Performance Hierarchical Tucker Tensor Learning Using GPU Tensor Cores. IEEE Transactions on Computers. Under review.