Project: A Study on GPU Matrix-matrix Products

Please at least finish the requirements marked with red. Then do as much as you can for the rest.

October 14, 2025

Background

- In our lecture, we discussed why GPU is very effective for massive parallel operation
- We also discussed block algorithms on CPU for reducing the cost of memory access in matrix-matrix products
- In this project, we want to study common techniques to accelerate matrix products on GPU and compare with optimized libraries

Overall Requirements

In this project, you are required to

- study, implement, and analyze at least three kernels (two designated + one optional) for matrix multiplication
- compare the performance of the above kernels with existing optimized BLAS on GPU, such as cuBLAS

Structure

The following is just a sample of the structure for this project. Feel free to study more on what you are interested in

- Kernel 1
- Kernel 2
- Compare Kernel 2 with CPU Block Algorithm
- Kernel 3
- Compare with existing optimized BLAS on GPU

Kernel 1: Memory Coalescing Kernel I

- In our slides of doing matrix multiplication, we use

```
int i = blockDim.y*blockIdx.y +
        threadIdx.y;
int j = blockDim.x*blockIdx.x +
        threadIdx.x;
```

  to assign which entry of C is calculated for each thread
- Alternatively, let us consider the one used in the MatAdd kernel:

Kernel 1: Memory Coalescing Kernel II

```
int i = blockDim.x*blockIdx.x +
        threadIdx.x;
int j = blockDim.y*blockIdx.y +
        threadIdx.y;
```

- Which is more efficient? Please compare the two settings and explain the reason in detail
- You are encouraged to draw some nice graphs to illustrate the memory access pattern for each setting

Kernel 2: Tiled Matrix Multiplication I

- For the second kernel, you want to utilize shared memory within the same block to accelerate matrix products.
  - Global memory can be accessed by all threads
  - Shared memory can only be accessed by threads within the same block. It can be declared using `__shared__` qualifier
- Assume the matrices $A \in \mathbb{R}^{M \times K}$, $B \in \mathbb{R}^{K \times N}$, and $C \in \mathbb{R}^{M \times N}$

Kernel 2: Tiled Matrix Multiplication II

- In this kernel, each thread still compute a $C_{i,j}$ entry
- To calculate $C_{i,j} = \sum_{k=1}^{K} A_{i,k} B_{k,j}$, we need to load $A_{i,k}$ and $B_{k,j}$ for $k = 1, \ldots, K$
  - $A_{i,1}$ is not only used for calculating $C_{i,j}$, but also for

$$C_{i,j+1} = \sum_{k=1}^{K} A_{i,k} B_{k,j+1}, C_{i,j+2} = \sum_{k=1}^{K} A_{i,k} B_{k,j+2},$$

    and so on

Kernel 2: Tiled Matrix Multiplication III

- Note that, the computation for $C_{i,j+1}, C_{i,j+2}, \ldots$ may happen in the same block, but elements like $A_{i,1}, A_{i,2}, \ldots$ are repeatedly loaded for many times in different thread
- Therefore, the idea of Kernel 2 is to load elements like $A_{i,1}$ to the shared memory so that all threads in the same block can access it without loading them for many times

Kernel 2: Tiled Matrix Multiplication IV

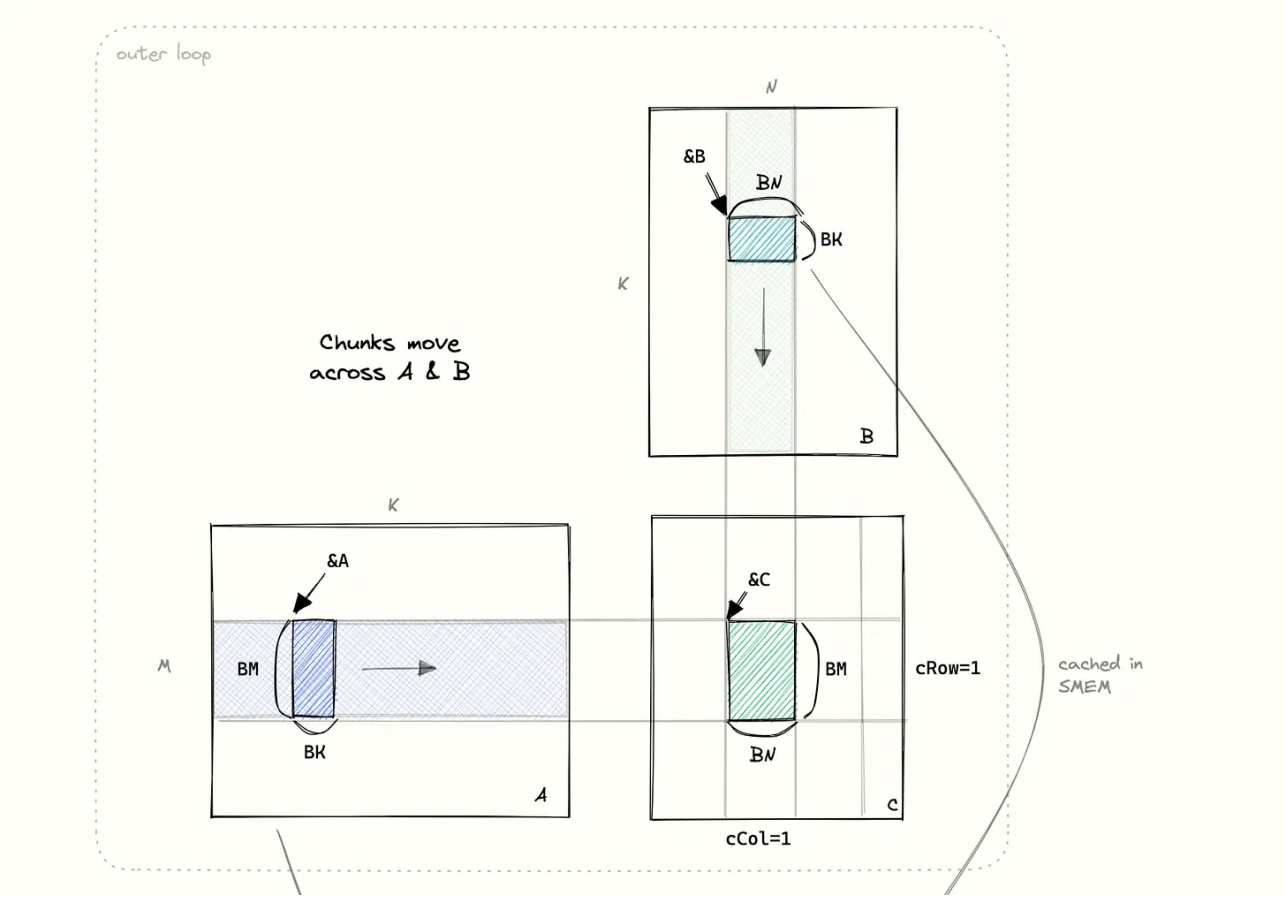
- Specifically, suppose a block with dimension (B_M, B_N) is responsible for calculating the submatrix $C_{i:i+B_N, j:j+B_M}$, which is given by

$$\begin{bmatrix} C_{i,j} & C_{i,j+1} & \cdots & C_{i,j+B_M-1} \\ C_{i+1,j} & C_{i+1,j+1} & \cdots & C_{i+1,j+B_M-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{i+B_N-1,j} & C_{i+B_N-1,j+1} & \cdots & C_{i+B_N-1,j+B_M-1} \end{bmatrix}$$

- We need the i-th to the $(i + B_N)$-th rows of A and j-th to the $(j + B_M)$-th columns of B

Kernel 2: Tiled Matrix Multiplication V

- See the following figure for the idea



1

Kernel 2: Tiled Matrix Multiplication VI

- We split these rows in A and columns in B into T tiles, say $A_1^{tile}, \ldots, A_T^{tile}$ and $B_1^{tile}, \ldots, B_T^{tile}$
- So Kernel 2 is like the following:
  For each $t = 1, \ldots, T$:
  - Load A_t^{tile} and B_t^{tile} to the shared memory
  - $C_{i:i+B_N, j:j+B_M} \leftarrow C_{i:i+B_N, j:j+B_M} + A_t^{tile} \times B_t^{tile}$
- Please implement this kernel and compare the one in the GPU course slides (including the performance and analysis of the number of memory accesses)

[1] Source from `https://siboehm.com/articles/22/CUDA-MMM`.

Kernel 2 vs. CPU Block Algorithm I

- Motivation: the "tile" mentioned in Kernel 2 looks similar to the "block" in the CPU block algorithm
- However, they are actually different
- Let us borrow the example from the course slides:

$$\begin{bmatrix} A_{11} & \cdots & A_{14} \\ & \vdots & \\ A_{41} & \cdots & A_{44} \end{bmatrix} \begin{bmatrix} B_{11} & \cdots & B_{14} \\ & \vdots & \\ B_{41} & \cdots & B_{44} \end{bmatrix}$$
$$= \begin{bmatrix} A_{11}B_{11} + \cdots + A_{14}B_{41} & \cdots \\ \vdots & \ddots \end{bmatrix}$$

Kernel 2 vs. CPU Block Algorithm II

- Here we abuse the notation a bit to denote $A_{11}, A_{12}, \ldots$ a block/tile instead of a single element
- In the CPU block algorithm, we sequentially do

$$\left.\begin{array}{l} C_{11} \leftarrow C_{11} + A_{11}B_{11} \\ \quad\vdots \\ C_{14} \leftarrow C_{14} + A_{11}B_{14} \end{array}\right\} \text{keep } A_{11} \text{ in memory}$$

$$\left.\begin{array}{l} C_{11} \leftarrow C_{11} + A_{12}B_{21} \\ \quad\vdots \\ C_{14} \leftarrow C_{14} + A_{12}B_{24} \end{array}\right\} \text{keep } A_{12} \text{ in memory}$$

$$\quad\vdots$$

Kernel 2 vs. CPU Block Algorithm III

- However, in Kernel 2, within each thread block, we sequentially do

$$\left.\begin{aligned} C_{11} &\leftarrow C_{11} + A_{11}B_{11} \\ C_{11} &\leftarrow C_{11} + A_{12}B_{21} \\ C_{11} &\leftarrow C_{11} + A_{13}B_{31} \\ C_{11} &\leftarrow C_{11} + A_{14}B_{41} \end{aligned}\right\}$$

Kernel 2 vs. CPU Block Algorithm IV

- Is it possible to design a kernel so that within a block, we do

$$\left.\begin{aligned} C_{11} &\leftarrow C_{11} + A_{11}B_{11} \\ C_{12} &\leftarrow C_{12} + A_{11}B_{12} \\ C_{13} &\leftarrow C_{13} + A_{11}B_{13} \\ C_{14} &\leftarrow C_{14} + A_{11}B_{14} \end{aligned}\right\}$$

  so that we can store the A_{11} in the shared memory to have a more efficient implementation?
- Please study the feasibility or even do some experiments

Kernel 3: Survey Other Kernels for Acceleration I

- For this kernel, please survey other techniques for faster matrix multiplication on GPU
- Please include the description, ideas, analysis, and the source codes