CHALMERS
UNIVERSITY OF TECHNOLOGY

DAT400 Lab 6
Matrix Multiplication Parallelization using CUDA

Contacts: Sonia Rani Gupta, Hari Abram, Miquel Pericas

October 9, 2023

In this lab, you are going to implement the parallelization of matrix multiplication in CUDA on GPU, A(n×n) × B(n×n) = C(n×n). We provide you with a basic but incomplete CUDA code in Canvas named cuda_mm.cu.

Setup CUDA path first:

```
vim ~/.bashrc
export PATH=/chalmers/sw/sup64/cuda_toolkit-12.1.1/bin:$PATH
export LD_LIBRARY_PATH=/chalmers/sw/sup64/cuda_toolkit-12.1.1/targets/
x86_64-linux/lib:$LD_LIBRARY_PATH
SAVE and QUIT
source ~/.bashrc
```

Verify CUDA installation on the lab machine:

```
hariv@f-7204-08:~$ cat /proc/driver/nvidia/version
NVRM version: NVIDIA UNIX x86_64 Kernel Module  535.54.03  Tue Jun  6 22:20:39 UTC 2023
GCC version:  gcc version 10.2.1 20210110 (Debian 10.2.1-6)

hariv@f-7204-08:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Mon_Apr__3_17:16:06_PDT_2023
Cuda compilation tools, release 12.1, V12.1.105
Build cuda_12.1.r12.1/compiler.32688072_0
```

1 Basic Implementation

Figure 1 shows an example of the basic implementation of matrix multiplication using CUDA, where the 32 × 32 matrix C is divided into four tiles and each tile is sized 16 × 16. Therefore, one way to implement this is that one thread block computes one tile of matrix C. One thread in the thread block computes one element of the tile.

1.1 Task 1:

- Open cuda_mm.cu, complete **cudaMalloc**, **cudaMemcpy(both directions)**, **Block_num** and **Thread_num** settings.
- **Implement** the kernel function.

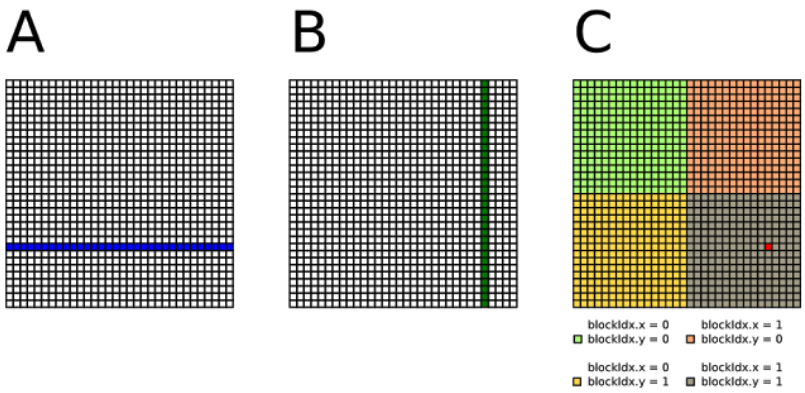

Figure 1: Basic implementation example.

- How many thread blocks do you use? What is your thread block dimension? How many threads in each block? What is the thread dimension within a block? Try out at least four different configurations.
- Compile and run: nvcc -o matmul_v1 cuda_mm.cu; ./matmul_v1
- Check the correctness of your implementation using printed Max error and Average error. These errors are calculated by comparing the results of CPU computation (matrix D) and GPU computation (matrix C).
- The Nsight Systems tool from NVIDIA can be used to create detailed profiles of where codes are spending time and what resources they are using. It works for compiled CUDA code. By running with **ncu ./matmul_v1**, you can quickly see a summary of all the kernels and memory copies that it uses. Report the corresponding execution times of your implemented kernel when using different configurations and explain the results. You can also create a profile using the command **ncu -o profile ./matmul_v1**, this profile can be used to visualize kernel execution time and memory copies that your code executes, with the command **ncu-ui ./⟨ generated-profile-report ⟩** NVIDIA Nsight Compute UI is launched.

2 Tiling Implementation using Shared Memory

In section 1, the implementation of the basic tiling matrix C loads data from global memory. While global memory resides in device memory (DRAM), it is much slower to access than shared memory. Thus a profitable way of performing optimized computation on the device is to tile data to take advantage of the faster-shared memory as follows:

- Partition the input data into subsets that **can fit into the shared memory**;
- Handle **each data subset with one thread block** by loading the subset from global memory to shared memory;
- Perform the computation on the subset from shared memory, each thread can efficiently multi-pass over any data element;
- Copy the results from shared memory back to global memory.

Figure 2 shows an example of the approach. The GPU kernel computes matrix C in multiple iterations. In each iteration, one thread block loads one tile of A and one tile of B from global memory

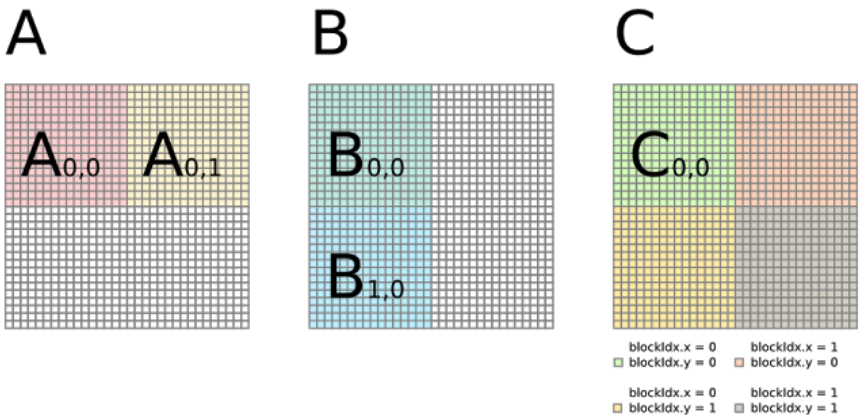

Figure 2: Tiling technique using the shared memory.

to shared memory, then performs the computation, and stores temporal results of C in registers. For example, a thread block can compute the tile C(0,0) in two iterations: C(0,0) = A(0,0)× B(0,0) + A(0,1) × B(1,0).

2.1 Task 2

- Report the shared memory size of your testing machine.
- How many elements can fit in the shared memory?
- Implement new tiling matrix multiplication kernel.
- How many blocks do you use? Dimension? How many threads are in each block? Dimension? Try out at least four different configurations.
- Report the corresponding execution time of different configurations using ncu and explain the results.

Lab 6 Report Submission:

- Submit the report as a group of two. Write down the group number and CIDs of each partner (i.e., who participated in this lab).
- You will get a PASS only if all the tasks are properly addressed in the report, including your code implementation details.