# Multicore and GPU Programming - Matrix Multiplication

**Prof. Lee Jinho**

Due - 05/04 Fri 23:59

## 0 Introduction

In this project, you will implement matrix multiplication with GPU Programming. The specification has the following contents. You just need to use CUDA on it.

### !!!Caution: Plagiarism!!!

You need to be careful to comply with plagiarism regulations. There is an ambiguity between discussion, idea, and solution. Therefore, be careful not to take away the opportunity of your colleague.

- No web code-searching
- No idea sharing
- Ok discussion about the project

## 1 Environment Setup

The same manner as in previous homeworks.

You don't need to set up the environment. We prepared everything you need to do an experiment about GPU programming. Check GNU GCC/G++ 9.2.1, GNU make, CUDA 11.3 and HTCondor documentation for more information.

## 2 Framework

### 2-1 Matrix Multiplication

#### 2-1-0 Structure of template code

```
/HW5/matmul$ tree .
.
├── bin                      // make file generate run file to here
├── cuobj                    // make file generate CUDA objective files to here
├── obj                      // main objective file in here
│   └── driver.o             // includes main(). Only the objective file provided!
├── Makefile                 // Compile, Clean, Format, Submit
├── matmul4096.cmd           // Command for condor
├── matmul2048.cmd           // Command for condor
├── matmul1024.cmd           // Command for condor
├── matmul128.cmd            // Command for condor
├── matmul64.cmd             // Command for condor
├── result                   // Result from condor
└── src                      // Source Code
    ├── matmul_kernel.cu
    └── matmul.h

6 directories, 6 files
```

#### 2-1-1 Implement your optimized solution

```
/HW5/matmul$ cat src/matmul_kernel.cu

#include <stdio.h>
#include <iostream>
#include <chrono>
#include <assert.h>
#include "matmul.h"
using namespace std;
```

```
void allocateDeviceMemory(void** M, int size)
{
  cudaError_t err = cudaMalloc(M, size);
  assert(err==cudaSuccess);
}

void deallocateDeviceMemory(void* M)
{
  cudaError_t err = cudaFree(M);
  assert(err==cudaSuccess);
}

void matmul_ref(const int* const matrixA, const int* const matrixB,
                int* const matrixC, const int n) {
  // You can assume matrixC is initialized with zero
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
        matrixC[i * n + j] += matrixA[i * n + k] * matrixB[k * n + j];
}

void matmul_optimized(const int* const matrixA, const int* const matrixB,
                      int* const matrixC, const int* const d_A, const int* const d_B,  int*
const d_C, const int n) {

  // TODO: Implement your CUDA code
}
```

As you see, the code given is almost the same to your HW3. However, you're supposed to use GPU to get the performance even lower. The target is 0.20sec (which is easy).

Your **memcpy** should be **included** within the measurements. However, you don't include the time for allocating the device memories (because we didn't include CPU memory allocation either). In case you want to do something about the allocation (You probably don't) we have put separate alloc/dealloc functions. In the driver.cpp that is provided as pre-compiled binaries, it would look something like this:

```
int main()
{
…
A = new int[N*N];
B = new int[N*N];
C = new int[N*N];
allocateDeviceMemory(&d_A);
allocateDeviceMemory(&d_B);
allocateDeviceMemory(&d_C);
time_start();
matmul_optimized(A, B, C, d_A, d_B, d_C, N);
time_end();
deallocateDeviceMemory(&d_A);
deallocateDeviceMemory(&d_B);
deallocateDeviceMemory(&d_C);
free A, B, C;
…
}
```

If you want to allocate anything else, you should do it within **matmul_optimized().**
Of course, you're not allowed to change or re-write the main(). If you think you need to, please contact us.

### 2-1-2 Compile and run matmul

The 'matmul' program gets two arguments (we have removed SanityCheck)

- inputPath: path to input file
- outputPath: path to output file

```
# Compile
/HW5/matmul$ make
mkdir -p cuobj
/usr/local/cuda-10.2/bin/nvcc -g --ptxas-options=-v -std=c++11 -O3 -Iinclude -c
src/matmul_kernel.cu -o cuobj/matmul_kernel.o
ptxas info    : 0 bytes gmem
CUDA Compiled src/matmul_kernel.cu successfully!
mkdir -p bin
OBJ: obj/driver.o
CUOBJ: cuobj/matmul_kernel.o
g++  obj/driver.o cuobj/matmul_kernel.o -o bin/matmul -g -fopenmp -std=c++11 -Wall
-Wno-sign-compare -O3 -L/usr/local/cuda/lib64 -lcudart -Iinclude
-I/usr/local/cuda/include
Compiled obj/driver.o successfully!

# Show help
/HW5/matmul$ ./bin/matmul
./matmul inputPath outputPath


# Local run
/HW5/matmul$ ./bin/matmul /nfs/home/mgp2022_data/hw5/input_2048.txt
/nfs/home/mgp2022_data/hw5/output_2048.txt

=====================================
      Matrix Multiplication
=======================================
The size of Matrix: 2048
=======================================

Read input file(/nfs/home/mgp2022_data/hw5/input_2048.txt)...
Read output file(/nfs/home/mgp2022_data/hw5/output_2048.txt)...

Run your solution...

matmul_optimal took 4.6e-08 sec
Correct
```

## 3 Remote Execution

There are five submit function, **submit_64, submit_128, submit_1024, submit_2048** and **submit_4096**. Each number means the size of matrix.

Step 0. Download or generate proper input and output files.
Step 1. Submit a job to condor.
Step 2. Wait until finished.(Check **result/matmul.log** for Job terminated)
Step 3. Check the output(**result/matmul.out**). There will be **TIME** and **CORRECTNESS** of your solution.

```
/HW5/matmul$ cat matmul2048.cmd
###############################################
##
## Matrix Multiplication Condor command file
##
###############################################

executable      = bin/matmul
output          = result/matmul.out
error           = result/matmul.err
log             = result/matmul.log
request_cpus = 16
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT
arguments               = /nfs/home/mgp2022_data/hw5/input_2048.txt
/nfs/home/mgp2022_data/hw5/output_2048.txt
queue


/HW5/matmul$ make submit_2048
condor_submit matmul2048.cmd
Submitting job(s).
```

```
1 job(s) submitted to cluster 12738.

/HW/matmul$ cat result/matmul.out
=======================================
      Matrix Multiplication
=======================================
The size of Matrix: 2048
=======================================

Read input file(/nfs/home/mgp2022_data/hw5/input_2048.txt )...
Read output file(/nfs/home/mgp2022_data/hw5/output_2048.txt)...

Run your solution...

matmul_optimal took TIME sec
CORRECTNESS
```

We check only **the runtime of submit_4096.** other submit cmds will be a faster indicators of whether you are doing well in our server. So, do not hesitate to use these.

# 4 Criteria

## 4-1 Requirements

### 4-1-0 General

- You should implement matrix multiplication with CUDA. We do not allow any matmul techniques in the CPU.
- No external libraries. You can only use STL.
    - This of course means: No cuBLAS
- Do not override driver.o

### 4-1-1 Matrix Multiplication

- Problem size: 4096x4096
- Performance requirements: **0.2**sec (4096x4096, This time includes running memcpy().)
- Correctness: Should work correctly for any square matrix input between size of 64x64 ~ 8192x8192

### 4-2 Measurements

We are measuring performance from a real machine, and it involves a certain amount of luck. To reduce the luck effect as much as possible, we do the following:

- Turn off DVFS of the machine
- Warm up the server for a few minutes before measuring. This means that the **program will run slower than your own tests** for all of you during our grading.
- We will run your program at least five times and **use the minimum exec.time**.

### 4-3 Guideline

This assignment is designed to be super-easy, in that most of the code is already provided in the lecture slides. You can directly dive into the final version, but we suggest that you do the following in sequence, such that you will gradually see how each technique improves your performance.

1. The naive version, which directly updates the partial sums to global memory (from slides p.7)
2. Replace the memory update to a local variable on register (p.10)
3. Blocked matmul with shared memory (p.26). This will already get you to the target performance.
4. Try to further use the remaining shared memory to get the extra credit.

## 5 Report

- No reports. just code :)

## 6 Grading

- You get 50 pts if your program runs legally within 0.2sec.
    - To get the full score, your program must not only run correctly within 0.2sec for input size 4096x4096, but also run correctly for <u>any square matrix</u> input between size of 64x64 ~ 8192x8192
- 25pts for being a correct program
    - Compile error or incorrect results: 0pts
    - If the program can be fixed with only with Makefile: 5pts penalty (20pts)
- +25 performance points for being correct and running faster than 0.2sec
    - If your program is correct on all test cases but runs slower than 0.2sec:
        - perf score = max(25 (1/t - 1/0.25), 0)
- +5 extra points if your program runs faster than 0.14sec

## 7 More materials

- GNU GCC/G++ 9.2.1
- GNU make
- CUDA
- HTCondor
- Our Lecture Slides