Lab 09:
GPU Programming - Matrix Multiplication

Moreh Vietnam

Matrix Multiplication

Given:

- **Matrix A** with dimension **MxK** (M rows, N columns)
- **Matrix B** with dimension **KxN** (K rows, N columns)
- Result **Matrix C** with dimension **MxN**

Each element C[i][j] (where 0<=i<M, 0<=j<N) is computed as:

$$C[i][j] = \sum_{k=0}^{K-1} A[i][k] \cdot B[k][j]$$

This is the **dot product** of **the i row of A** and **the j column of B**

Practice

You are given examples of matrix multiplication at
https://colab.research.google.com/drive/1MqmttKRsckSpnw5cJmLl--R6MidbFywM

Your task is:

1. Port some implementations from CUDA to HIP
2. Make an optimized version
3. Benchmark all implementations (from 1. and 2.) on the practice server
4. Write a report.

Practice (1)

Port some implementations from CUDA to HIP, 5 to do:

1. Naive
2. Tiled (Block level)
3. Tiled (1D - ILP)
4. Tiled (2D - ILP)
5. Vectorized
6. Warp Tiled

Practice (2)

Make an optimized version:

- This is an implementation of your own
  - Written in “best_gemm.cpp”
  - Make sure that your implementation can:
    - Verify the accuracy of output data
    - Calculate the throughput (GFLOPS) of the kernel
- Using any technique from the course
  - Advance technology like matrix core, tensor core are not allowed
  - Only “float” is allowed for floating point data type
- Your implementation will be tested with the following set of MxNxK:
  - 1024x1024x128
  - 1024x1024x1024
  - 512x2048x4096
  - 8192x8192x8192

Practice (3)

Benchmark all implementations

- On the practice server (of course)
- Use one MI250 GPU only
  - You don’t have to do anything, just use `srun` like the example in the practice server introduction slides

Practice (4)

Write a report, your report should contain:

- What optimized techniques are used inside each implementation
- How do optimized techniques make improvements, compare to the previous implementation?
  - This is optional
  - But **it affects to your score**, you should try your best
- The performance number (GFLOPS) of each implementation

Scoring

Remember, this lab is a part of your midterm exam.

10 points in total:

- 8 for your report:
  - You will gain 5 points, if:
    - All the CUDA examples are ported into HIP and benchmarked on the practice server
    - All optimization techniques used in those CUDA to HIP implementations are mentioned
  - 3 more bonus points for:
    - Explain how optimization techniques works
    - What have been done in your “best_gemm” implementation
- 2 points for best performing implementation **in class**
  - Other students get points adjusted for their implementation performance relative to the best.
    - **Nothing for the worst**
  - Remember that we only count result for implementation in “best_gemm.cpp”

Submission

A zip file, which contains:

- 6 .cpp files of CUDA to HIP implementations
- “best_gemm.cpp”
- Your report in PDF

Due date:

- **23:59 09/04/2025**
- **You cannot submit after that**
- **This is not an endless assignment**

HIP

HIP is a C++ Runtime API and Kernel Language that allows developers to create portable applications for AMD and NVIDIA GPUs from single source code.

- https://github.com/ROCm/hip
- In short:
  - Replace “cuda” by “hip”
  - Compile with “hipcc” instead of “nvcc”

Example code

```
#include <iostream>
#include <hip/hip_runtime.h>

// Error checking macro for HIP calls
#define CHECK_HIP(cmd) \
{\
    hipError_t error = cmd;\
    if (error != hipSuccess) { \
        std::cerr << "HIP error: " << hipGetErrorString(error) << " at line " << __LINE__ <<
std::endl;\
        exit(EXIT_FAILURE); \
    }\
}

// Simple HIP kernel that prints from GPU
__global__ void helloFromGPU() {
    printf("Hello World from GPU thread %d!\n", threadIdx.x);
}

int main() {
    // Print from CPU first
    std::cout << "Hello World from CPU!" << std::endl;

    // Launch kernel with 1 block containing 8 threads
    helloFromGPU<<<dim3(1), dim3(8)>>>();
    CHECK_HIP(hipGetLastError());

    // Wait for GPU to finish
    CHECK_HIP(hipDeviceSynchronize());

    std::cout << "Done!" << std::endl;
    return 0;
}
```

Workflow

1. Compile code with `hipcc` on login-node

   Example: `hipcc -O3 hip_hello.cpp -o hip_hello --offload-arch=gfx90a`

2. Execute program with `srun` on working-node

   Example: `srun --time=01:00 ./hip_hello`