BÁO CÁO THỰC HÀNH: TỐI ƯU HÓA GEMM TRÊN KIẾN TRÚC AMD

Họ và tên: Đỗ Hạnh Nhi

Môi trường thực hành: AMD Instinct MI250 (Architecture: `gfx90a`)

---

1. Tổng quan

Mục tiêu của bài thực hành là chuyển đổi (porting) các kernel nhân ma trận (GEMM - General Matrix Multiply) từ CUDA sang HIP để chạy trên hệ thống GPU AMD Instinct, đồng thời áp dụng và phân tích các kỹ thuật tối ưu hóa bộ nhớ, tính toán song song để đạt hiệu năng cao nhất.

Bài lab này đã thực hiện:

1. Chuyển đổi mã nguồn từ CUDA sang HIP.
2. Khắc phục các vấn đề tương thích kiến trúc (Wavefront size 64 vs 32).
3. Benchmark hiệu năng trên nhiều kích thước ma trận khác nhau.

---

2. Các kỹ thuật tối ưu hóa đã sử dụng

Dưới đây là danh sách các phiên bản kernel đã được thực hiện và kỹ thuật tối ưu tương ứng:

1. **Naive (`naive`):**
    - **Mô tả:** Cài đặt nhân ma trận cơ bản theo định nghĩa toán học với 3 vòng lặp lồng nhau.
    - **Đặc điểm:** Sử dụng trực tiếp **Global Memory** cho mọi thao tác đọc/ghi. Hiệu năng phụ thuộc hoàn toàn vào L2 Cache của GPU.
2. **Shared Memory Tiling (`tiled`):**
    - **Kỹ thuật:** Chia ma trận lớn thành các khối nhỏ (Tiles) và tải dữ liệu vào **Shared Memory**.
    - **Tác dụng:** Giảm độ trễ truy cập bộ nhớ bằng cách tái sử dụng dữ liệu trong Shared Memory (tốc độ truy cập nhanh gấp ~100 lần Global Memory).
3. **1D Register Tiling (`tiled_1d`):**
    - **Kỹ thuật:** Mỗi thread tính toán một vector kết quả của ma trận C (theo 1 chiều) thay vì 1 điểm.
    - **Tác dụng:** Tăng cường **ILP (Instruction Level Parallelism)** và giảm số lượng chỉ thị truy cập bộ nhớ bằng cách giữ các kết quả trung gian trong **Registers** (Thanh ghi).
4. **2D Register Tiling (`tiled_2d`):**
    - **Kỹ thuật:** Mở rộng `tiled_1d` ra 2 chiều. Mỗi thread chịu trách nhiệm tính toán một ô vuông con (ví dụ 4x4 hoặc 8x8) của ma trận C.
    - **Tác dụng:** Tối ưu hóa tối đa việc sử dụng thanh ghi, giảm áp lực băng thông lên Shared Memory.
5. **Vectorization (`vectorize`):**
    - **Kỹ thuật:** Sử dụng kiểu dữ liệu `float4` để đọc/ghi 4 số thực (128-bit) trong một chu kỳ lệnh.
    - **Tác dụng:** Tăng băng thông bộ nhớ hiệu dụng và giảm số lượng chỉ thị (instruction count) cần thiết để nạp dữ liệu.
6. **Warp Tiling (`warp_tiled`):**
    - **Kỹ thuật:** Phân chia công việc theo cấp độ Wavefront (AMD). Các thread trong cùng một Wavefront hợp tác tải dữ liệu và tính toán để tránh Bank Conflict trong Shared Memory.

---

3. Giải thích cơ chế & Phân tích hiệu năng

3.1. Cơ chế hoạt động

- **Latency Hiding (Che giấu độ trễ):** GPU có độ trễ bộ nhớ cao. Các kỹ thuật Tiling giúp GPU có dữ liệu sẵn trong bộ nhớ nhanh (Shared/Register) để các nhân tính toán (ALU) không phải chờ đợi.
- **Memory Hierarchy:**
    - `Naive` : Phụ thuộc Global Memory (VRAM) -> Chậm.
    - `Tiled` : Sử dụng Shared Memory (User-managed Cache) -> Nhanh hơn.
    - `Register Tiling` : Sử dụng Registers (Nhanh nhất) -> Tối ưu nhất.

3.2. Kết quả Benchmark thực tế

Các kết quả dưới đây được ghi nhận trên node `MV-DZ-MI250` với cấu hình biên dịch `-O0` (tắt tối ưu hóa trình biên dịch) để đánh giá hiệu năng thuần túy của thuật toán.

3.2.1 Cấu hình 1: 1024 x 1024 x 128 (Kích thước nhỏ)

Dữ liệu kích thước nhỏ (~0.5 MB), nằm gọn trong L2 Cache.

Executable	Runs	Avg Time (ms)	Avg Perf (GFLOP/s)
./naiveo0	10	**9.79**	**219.30**
./tiledo0	10	8.02	33.48
./tiled_1do0	10	7.42	36.34
./tiled_2do0	10	17.21	15.65
./vectorizeo0	10	17.66	15.20
./warp_tiledo0	10	34.08	7.91

Phân tích: Kernel `naive` đạt hiệu năng cao nhất. Do kích thước ma trận nhỏ, dữ liệu được cache hiệu quả bởi phần cứng. Các kỹ thuật Tiling gây ra chi phí quản lý (overhead) sao chép dữ liệu vào Shared Memory không cần thiết trong trường hợp này.

3.2.2 Cấu hình 2: 1024 x 1024 x 1024 (Kích thước tiêu chuẩn)

Dữ liệu vượt quá dung lượng L2 Cache.

Executable	Runs	Avg Time (ms)	Avg Perf (GFLOP/s)
./naiveo0	10	51.50	41.70
./tiledo0	10	31.09	69.07
./tiled_1do0	10	**29.23**	**73.48**
./tiled_2do0	10	109.71	19.57
./vectorizeo0	10	110.88	19.37
./warp_tiledo0	10	234.88	9.14

Phân tích: Kernel `naive` giảm hiệu năng đáng kể do nghẽn băng thông Global Memory. Các kernel sử dụng Shared Memory (`tiled` , `tiled_1d`) cho thấy hiệu quả rõ rệt, đạt hiệu năng cao nhất (~73 GFLOP/s). Các kernel phức tạp (`tiled_2d` , `warp_tiled`) có hiệu năng thấp do hiện tượng tràn thanh ghi (register spilling) khi không có tối ưu hóa `-O3` .

3.2.3 Cấu hình 3: 512 x 2048 x 4096 (Ma trận chữ nhật lớn)

Kích thước K lớn, kiểm tra khả năng duy trì băng thông.

Executable	Runs	Avg Time (ms)	Avg Perf (GFLOP/s)
./naiveo0	5	**209.85**	**81.87**
./tiledo0	5	116.03	74.03
./tiled_1do0	5	109.92	78.15
./tiled_2do0	5	433.94	19.80
./vectorizeo0	5	430.03	19.98
./warp_tiledo0	5	929.80	9.24

Phân tích: Kernel `naive` và `tiled_1d` có hiệu năng tương đương. Cấu trúc lặp dài theo chiều K giúp phần cứng prefetcher hoạt động hiệu quả cho kernel `naive` . Kernel `warp_tiled` tiếp tục có thời gian thực thi lâu nhất do độ phức tạp tính toán chỉ số cao mà không được trình biên dịch tối ưu.

---

4. Cài đặt "Best GEMM" (Best Performing Implementation)

4.1. Cài đặt

Để đạt hiệu năng cao trên kiến trúc AMD Instinct MI250, phiên bản `best_gemm.cpp` đã được cài đặt bằng cách kết hợp các kỹ thuật sau:

- **Thích ứng với Wavefront 64 (AMD Architecture):**
  - Code gốc được thiết kế cho Warp size 32 (NVIDIA). Điều chỉnh lại kích thước Warp Tile (`WM`) lên 128 và định nghĩa lại `WARPSIZE=64` .
  - **Mục đích:** Đảm bảo sử dụng hết 64 luồng trong một Wavefront của AMD, tránh lãng phí tài nguyên tính toán và sai lệch chỉ số thread.
- **Vectorized Memory Access (float4):**
  - Sử dụng kiểu dữ liệu `float4` để tải dữ liệu từ Global Memory vào Shared Memory thay vì tải từng số thực (`float`).
  - **Mục đích:** Tận dụng chỉ thị load 128-bit của phần cứng, giảm số lượng câu lệnh truy cập bộ nhớ đi 4 lần và tối ưu hóa băng thông.
- **Tối ưu hóa Thanh ghi và Vòng lặp:**
  - Sử dụng `#pragma unroll` để trải phẳng các vòng lặp tính toán bên trong, loại bỏ chi phí kiểm tra điều kiện lặp (`branching`).
  - Tinh chỉnh kích thước Thread Tile (8x8) để cân bằng giữa việc tái sử dụng thanh ghi và chỉ số Occupancy.
- **Cờ biên dịch (Compiler Flags):**
  - Sử dụng lệnh: `hipcc -O3 --offload-arch=gfx90a` .
  - **Mục đích:** Kích hoạt các tối ưu hóa mức máy của trình biên dịch (như Loop Unrolling tự động, FMA - Fused Multiply Add) dành riêng cho kiến trúc `gfx90a` (MI250).

4.2. Kết quả Benchmark "Best GEMM" (Final Results)

Dưới đây là hiệu năng thực tế của phiên bản `best_gemm` sau khi áp dụng các kỹ thuật tối ưu hóa và biên dịch với cờ `-O3` trên kiến trúc AMD MI250:

Cấu hình (MxNxK)	Thời gian chạy (ms)	Hiệu năng (GFLOP/s)	Quy đổi (TFLOP/s)
1024 x 1024 x 128	0.0636	4,221.22	4.22
1024 x 1024 x 1024	0.3509	6,119.86	6.12
512 x 2048 x 4096	1.3291	6,463.08	6.46
8192 x 8192 x 8192	86.6248	**12,692.80**	**12.69**

---

5. Kết luận

Bài thực hành cho thấy sự khác biệt rõ rệt giữa lý thuyết thuật toán và thực tế phần cứng.

1. Với kích thước ma trận nhỏ/trung bình, **L2 Cache** và **Compiler Optimization** đóng vai trò quyết định (Naive có thể chạy rất nhanh).
2. Với kích thước ma trận lớn, các kỹ thuật **Tiling** và quản lý bộ nhớ thủ công là bắt buộc để tránh nghẽn băng thông.