ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SỬ DỤNG THUẬT TOÁN IMAGE CONVOLUTION ĐỂ XỬ LÝ ẢNH XÁM Y TẾ VỚI THƯ VIỆN OPENMP

BÁO CÁO THỰC NGHIỆM MÔN HỌC TÍNH TOÁN HIỆU NĂNG CAO

Giáo viên: Ts. Nguyễn Việt Anh

Nhóm: 12

Mã học phần: IT6069 – K18

Nhóm thực hiện: Nguyễn Đức Anh
Nguyễn Viết Doanh
Lê Minh Hiếu
Tạ Công Lợi

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

Xử lý ảnh y tế & Thuật toán Tích chập

Tổng quan về Xử lý ảnh y tế

Lĩnh vực ứng dụng thuật toán để thu thập, xử lý, phân tích và cải thiện chất lượng hình ảnh y khoa.

- Mục tiêu: Chẩn đoán & Điều trị

Thách thức trong xử lý ảnh y tế

- Độ chính xác cực cao (sai số nhỏ)
- Nhiễu ảnh (làm mờ chi tiết quan trọng)
- Khối lượng tính toán lớn (độ phân giải cao)

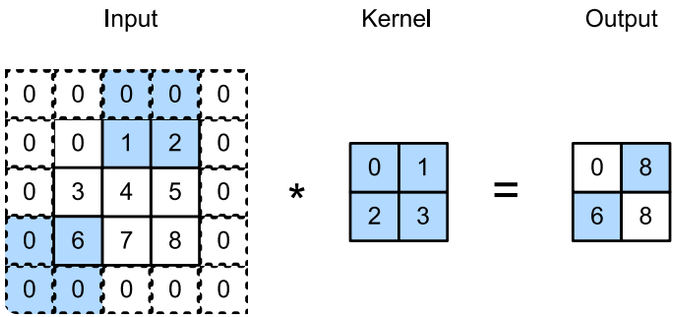
Đặc điểm của Ảnh xám y tế

Biểu diễn dưới dạng ma trận 2D, mỗi pixel là cường độ sáng. Thông tin **cấu trúc & mật độ mô** quan trọng hơn màu sắc.

Giảm khối lượng tính toán so với ảnh màu RGB, tập trung tài nguyên vào **độ phân giải không gian và độ sâu bit**.

Thuật toán Tích chập (Convolution)

Kỹ thuật nền tảng trong xử lý ảnh.



Tính lại giá trị pixel bằng trọng số lân cận.

Vai trò trong xử lý ảnh y tế

- Làm trơn & khử nhiễu (làm sạch ảnh)
- Làm nổi bật đặc trưng (phát hiện cạnh, biên dạng khối u)

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT (Tiếp theo)

Tổng quan về Lập trình song song & OpenMP

Giải pháp cho bài toán **tính toán lớn** và "nghẽn cổ chai" khi xử lý tuần tự trên ảnh y tế độ phân giải cao.

- Cung cấp khả năng song song hóa chương trình tuần tự.
- Hỗ trợ C/C++ và Fortran.

Lý thuyết về Tính toán Song song

Dựa trên nguyên lý **'chia để trị'**: chia tác vụ lớn thành nhiều tác vụ con thực hiện đồng thời trên nhiều lõi xử lý.

- **Độc lập dữ liệu:** Các phần ảnh xử lý độc lập.
- **Tối ưu hóa** sức mạnh CPU đa lõi.

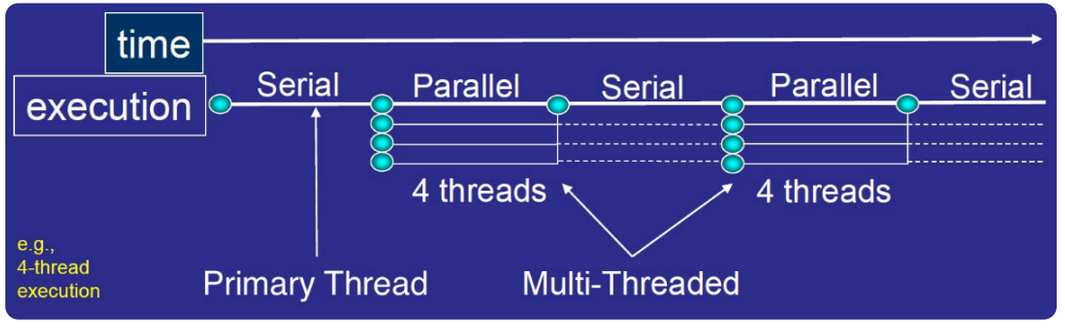
Mô hình OpenMP

Chuẩn lập trình ứng dụng phổ biến cho **tính toán song song** trên các hệ thống **bộ nhớ chia sẻ**.

- Tất cả các lõi/luồng truy cập chung bộ nhớ.
- Trao đổi dữ liệu qua biến chung, giảm độ trễ.

Mô hình Fork-Join

- **Fork (Phân nhánh):** Luồng chính tạo các luồng con.
- **Parallel:** Luồng con chia sẻ & xử lý đồng thời.
- **Join (Hợp nhất):** Luồng chính chờ & gộp kết quả.

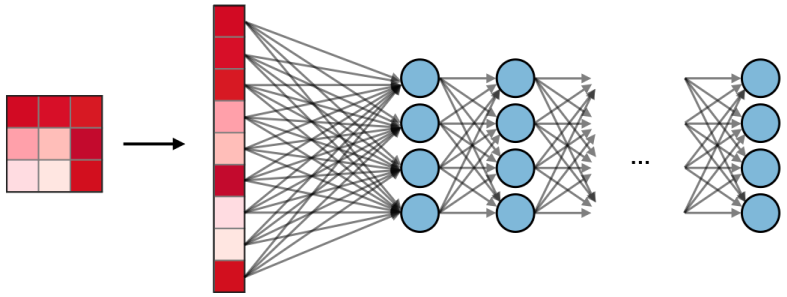


CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

1. Phân tích Lý thuyết Thuật toán Tuần tự

1.1. Mô hình Toán học của Phép Tích chập Rời rạc

Phép tích chập rời rạc 2D: $O(x,y) = \sum_{k_x,k_y} I(x+k_x, y+k_y) \cdot K(k_x,k_y)$. Đây là tổng có trọng số của các pixel lân cận, dùng cho khử nhiễu hoặc phát hiện biên.



1.2. Phân tích Độ phức tạp Tính toán

Thời gian: O(N² * k²) cho ảnh N x N, kernel k x k (2k² FLOPs/pixel). Tăng theo bình phương kích thước ảnh.

- Độ phức tạp không gian: O(N²)

2. Các vấn đề Hiệu năng trong Xử lý Tuần tự

2.1. Phân tích Đặc tính Truy xuất Bộ nhớ

Mảng 2D được lưu tuyến tính (row-major). Pixel không liền kề trong bộ nhớ gây ra Cache Miss, làm CPU chờ dữ liệu từ RAM.

- Hệ quả: Giảm hiệu suất thực tế đáng kể

2.2. Phân tích Điểm nghẽn (Bottlenecks)

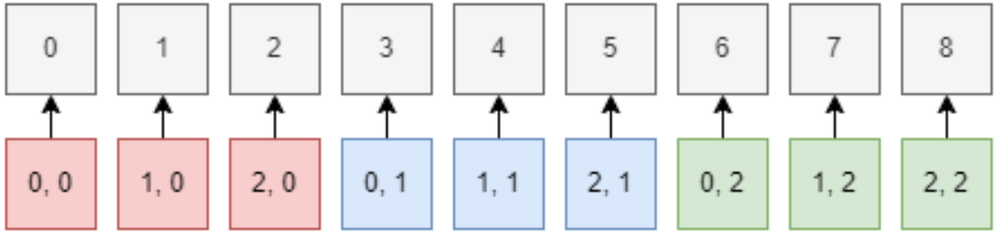
- **Lãng phí tài nguyên:** Chỉ dùng 1 nhân CPU, 90-95% sức mạnh hệ thống bị lãng phí.
- **Overhead do Rẽ nhánh:** Kiểm tra biên (if/else) hàng triệu lần, làm gián đoạn đường ống lệnh CPU.
- **Von Neumann Bottleneck:** Tốc độ CPU nhanh hơn RAM, CPU thường xuyên phải chờ dữ liệu.

CHƯƠNG 3: THIẾT KẾ GIẢI THUẬT

Thiết kế giải thuật Tuần tự

Chiến lược Tổ chức Dữ liệu

Mảng 1 chiều & Ánh xạ 2D sang 1D



Mô tả thuật toán

1. Khởi tạo, tính bán kính Kernel.
2. Duyệt pixel (x,y) lồng nhau.
3. Tích chập cục bộ (xử lý biên).
4. Chuẩn hóa và ghi kết quả.

Tranh chấp Dữ liệu (Race Condition)

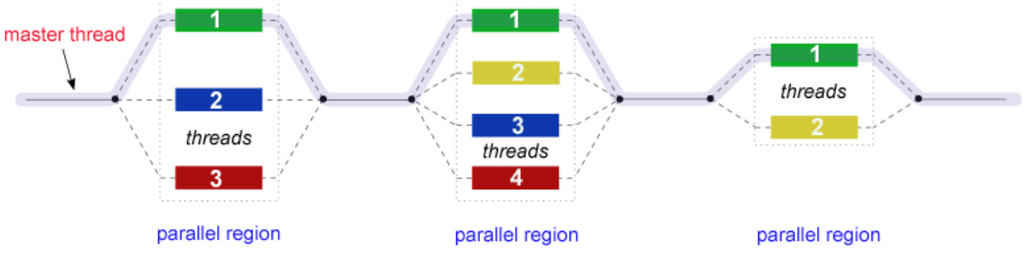
Nhiều luồng ghi cùng lúc gây lỗi. Giải pháp: Biến 'sum' phải **Private**.

Cơ sở lý thuyết OpenMP & Song song

Kiến trúc Bộ nhớ Chia sẻ (SMP)

Toàn bộ lõi truy cập RAM chung, giảm độ trễ.

Mô hình Thực thi Fork-Join



1. **Fork:** Luồng chính sinh luồng con.
2. **Parallel:** Luồng con xử lý đồng thời.
3. **Join:** Luồng con hợp nhất về chính.

Kỹ thuật OpenMP áp dụng

- Work-sharing Loop (`pragma omp parallel for`).
- Collapse Loop (gộp vòng lặp lồng nhau).
- Scheduling Strategy (Static).

CHƯƠNG 3: THIẾT KẾ GIẢI THUẬT SONG SONG

Chiến lược song song hóa phép tích chập ảnh, dựa trên phân tích điểm nghẽn thuật toán tuần tự và cơ sở lý thuyết OpenMP.

Phân tích phụ thuộc dữ liệu

Xác định các tác vụ độc lập. Trong tích chập ảnh, giá trị pixel đầu ra không phụ thuộc các pixel khác, cho phép song song hóa dễ dàng.

Chiến lược phân rã miền dữ liệu

Chia nhỏ không gian dữ liệu (ảnh HxW) bằng `collapse(2)` trong OpenMP để gộp vòng lặp, tăng cường cân bằng tải.

Quản lý phạm vi biến

Phân loại biến để tránh Race Condition:

SHARED: Input, Output, Kernel

PRIVATE: sum, biến vòng lặp (x, y)

Chiến lược lập lịch

Chọn cách phân phối công việc cho các luồng. Với tải cân bằng, **Static Scheduling** (`schedule(static)`) là tối ưu, giảm chi phí quản lý.

Thiết kế xử lý Biên

Giữ cơ chế kiểm tra `if` bên trong kernel cho xử lý biên. **Static scheduling** giúp giảm tác động của Branching Overhead.

CHƯƠNG 4: CÀI ĐẶT VÀ THỰC HIỆN

4.1. Môi trường cài đặt

Chương trình được phát triển và kiểm thử trên một môi trường cụ thể để đảm bảo tính nhất quán.

Phần mềm

- Hệ điều hành: Microsoft Windows 10/11 (64-bit).
- Ngôn ngữ lập trình: C++ (Tiêu chuẩn C++17).
- Trình biên dịch: GCC (MinGW-w64) hỗ trợ OpenMP.
- IDE: Visual Studio Code.
- Thư viện: stb_image.h, stb_image_write.h, omp.h, std::filesystem.

Phần cứng

- CPU: Intel Core i7 – 9850H (2.60 GHz, 12 luồng).
- RAM: 32GB Ram.
- Ổ cứng: SSD.

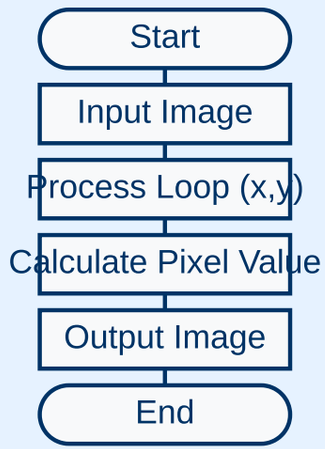
4.2. Cài đặt thuật toán Image Convolution xử lý ảnh tuần tự

Mục đích cài đặt

- Kiểm chứng độ chính xác của thuật toán tích chập.
- Làm cơ sở so sánh (Baseline) để đo độ tăng tốc (Speedup) khi song song hóa.

Mô tả thuật toán

Thuật toán trượt Kernel (thường 3x3) qua từng điểm ảnh. Giá trị điểm ảnh mới được tính bằng tổng có trọng số. Quy trình bao gồm Gaussian Blur để khử nhiễu và Sharpen để tăng cường biên.



CHƯƠNG 4: CÀI ĐẶT VÀ THỰC HIỆN (Tiếp theo)

1. Ý tưởng song song hóa

Bài toán xử lý ảnh có tính **"Song song dữ liệu"** cao. Giá trị pixel đầu ra độc lập với các vị trí khác. Sử dụng mô hình **Fork-Join** của OpenMP để chia nhỏ và xử lý đồng thời trên các luồng CPU.

2. Phân chia & phân phối dữ liệu

Sử dụng `pragma omp parallel for` với `collapse(2)` để chia không gian lặp (H x W) thành các khối công việc. Tăng khối lượng công việc mỗi luồng, cân bằng tải tốt hơn.

3. Tính toán cục bộ & Thu thập kết quả

Mỗi luồng tính toán trên vùng dữ liệu được phân công. Input là Read-only. Biến `sum`, `kx`, `ky` là riêng tư (**Private**). Ghi kết quả trực tiếp vào `output.pixels` an toàn.

4. Đánh giá sơ bộ

Đo thời gian bằng `std::chrono`. Song song hóa giảm thời gian xử lý đáng kể cho ảnh lớn. Cần lưu ý chi phí khởi tạo luồng có thể ảnh hưởng đến hiệu quả với ảnh nhỏ.

CHƯƠNG 5: THỰC NGHIỆM VÀ ĐÁNH GIÁ

1. Độ phức tạp của thuật toán

Tuần tự: $O(N^2)$ thời gian & $O(N^2)$ bộ nhớ.

Song song (OpenMP): Bậc độ phức tạp không đổi $O(N^2)$. Thời gian lý tưởng giảm: $T_{s\_par} = T_{s\_seq}/\text{num\_threads}$.

2. Cách đo thời gian

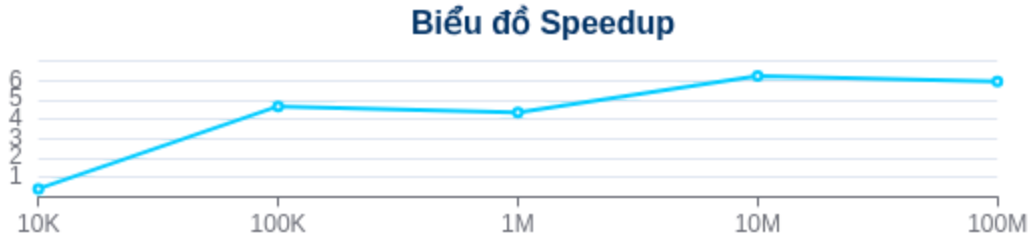
> Sử dụng thư viện C++ `std::chrono` để ghi lại thời điểm bắt đầu và kết thúc xử lý. Mỗi kiểm thử chạy 3 lần và lấy giá trị trung bình để đảm bảo độ chính xác.

3. Đánh giá và nhận xét

> Với ảnh nhỏ (N=10,000), OpenMP có overhead lớn hơn thời gian tính toán thực tế. Với ảnh lớn (N >= 100,000), OpenMP cho thấy hiệu quả rõ rệt. Speedup đạt đỉnh ~6x ở N=10 triệu pixel. Efficiency thấp ở N nhỏ, cải thiện đáng kể ở N lớn nhưng chững lại do băng thông bộ nhớ.

Kết quả thực nghiệm

N	Ts	Tomp	Ts/Tomp
10,000	0.0012247	0.0029925	0.409256
100,000	0.0126316	0.0027108	4.65973
1,000,000	0.0807984	0.0186202	4.33929
10,000,000	0.815536	0.131151	6.21829
100,000,000	8.26309	1.3905	5.94254



KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Tổng kết và Kết quả Đạt được

Đề tài đã xây dựng và tối ưu hóa thuật toán tích chập xử lý ảnh y tế bằng **OpenMP**. Chiến lược **phân rã miền dữ liệu** và **lập lịch tĩnh** đã được chứng minh là đúng đắn, mang lại hiệu quả vượt trội cho ảnh độ phân giải cao, khẳng định OpenMP là giải pháp phù hợp cho các bài toán xử lý ảnh y tế lớn.

Hạn chế của Đề tài

- Với ảnh kích thước nhỏ, chi phí khởi tạo luồng và quản lý vùng song song đáng kể, đôi khi tuần tự tối ưu hơn.
- Cần xác định ngưỡng dữ liệu phù hợp để kích hoạt chế độ song song.
- Khả năng mở rộng trên hệ thống quy mô lớn chưa được đánh giá toàn diện do giới hạn phần cứng thử nghiệm trên máy đơn.

Hướng Phát triển Tiếp theo

- Mở rộng nghiên cứu sang mô hình lập trình lai (MPI + OpenMP) cho hệ thống máy tính cụm.
- Chuyển đổi thuật toán sang nền tảng GPU với CUDA để khai thác khả năng xử lý song song hàng nghìn luồng.
- Tận dụng các chỉ thị vector hóa phần cứng (AVX) để xử lý đồng thời nhiều điểm ảnh trong một chu kỳ máy.