ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO THỰC NGHIỆM MÔN HỌC

TÍNH TOÁN HIỆU NĂNG CAO

ĐỀ TÀI

Sử dụng thuật toán Image Convolution để xử lý ảnh xám y tế với thư viện OpenMP.

Giáo viên:	Ts. Nguyễn Việt Anh
Nhóm:	12
Mã học phần:	IT6069 – K18
Nhóm sinh viên thực hiện:	
Nguyễn Đức Anh	MSV: 2023601621
Nguyễn Viết Doanh	MSV: 2023600535
Lê Minh Hiếu	MSV: 2023602439
Tạ Công Lợi	MSV: 2023601856

Hà Nội, 2025

ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO THỰC NGHIỆM MÔN HỌC
TÍNH TOÁN HIỆU NĂNG CAO

ĐỀ TÀI
Sử dụng thuật toán Image Convolution để xử lý ảnh xám y tế với thư viện OpenMP.

Giáo viên:	Ts. Nguyễn Việt Anh
Nhóm:	12
Mã học phần:	IT6069 – K18
Nhóm sinh viên thực hiện:	
Nguyễn Đức Anh	MSV: 2023601621
Nguyễn Viết Doanh	MSV: 2023600535
Lê Minh Hiếu	MSV: 2023602439
Tạ Công Lợi	MSV: 2023601856

Hà Nội, 2025

LỜI MỞ ĐẦU

Trong thời đại dữ liệu và trí tuệ nhân tạo phát triển mạnh mẽ, nhu cầu xử lý các bài toán tính toán lớn với tốc độ cao ngày càng trở thành yêu cầu cấp thiết. Các phương pháp tính toán truyền thống dần bộc lộ hạn chế khi phải làm việc với các mô hình và tập dữ liệu có quy mô ngày càng tăng. Vì vậy, tính toán hiệu năng cao (High Performance Computing–HPC) trở thành giải pháp quan trọng nhằm khai thác khả năng xử lý song song để rút ngắn thời gian và nâng cao hiệu quả.

Image Convolution trong CNN là một phần quan trọng trong việc xử lý ảnh. Nó giúp mô hình nhận diện các đặc trưng quan trọng trong ảnh một cách hiệu quả. Khi một ảnh đầu vào được đưa vào mạng CNN, nó sẽ trải qua các lớp tích chập (Convolutional layers), sử dụng các bộ lọc (filters) để trích xuất đặc trưng. Khi số pixel lớn, thuật toán này đòi hỏi tài nguyên đáng kể, khiến việc tối ưu hóa bằng HPC trở nên đặc biệt cần thiết

Đề tài "Sử dụng thuật toán Image Convolution để xử lý ảnh xám y tế với thư viện OpenMP" được thực hiện nhằm tìm hiểu nguyên lý song song hóa, triển khai các phương pháp tối ưu và đánh giá hiệu năng dựa trên thực nghiệm. Qua đó, đề tài góp phần minh chứng vai trò của HPC trong việc giải quyết các bài toán tính toán nặng hiện nay.

Nhóm em xin chân thành cảm ơn Tiến sĩ Nguyễn Việt Anh đã hướng dẫn và hỗ trợ nhóm em trong quá trình học tập và hoàn thành báo cáo này

MỤC LỤC

DANH MỤC HÌNH ẢNH

CHƯƠNG 1: GIỚI THIỆU

1.1. Lý do chọn đề tài

Trong kỷ nguyên số hóa y tế, việc xử lý và phân tích hình ảnh (như ảnh X-quang, CT, MRI) đóng vai trò then chốt trong chẩn đoán và điều trị bệnh[1]. Các hình ảnh y tế thường là ảnh xám có độ phân giải rất cao, đòi hỏi các thao tác xử lý như làm mịn, tăng cường độ sắc nét hoặc phát hiện biên thông qua thuật toán **Image Convolution** (phép nhân chập).

Tuy nhiên, với độ phức tạp tính toán lớn, việc thực hiện nhân chập trên các tập dữ liệu hình ảnh quy mô lớn bằng phương pháp tuần tự dần bộc lộ hạn chế, gây mất thời gian và không đáp ứng được yêu cầu về tốc độ xử lý trong y khoa. Trong bối cảnh đó, tính toán hiệu năng cao (HPC) với mô hình lập trình song song bộ nhớ chia sẻ như **OpenMP** trở thành giải pháp tối ưu nhằm khai thác sức mạnh của bộ vi xử lý đa lõi để rút ngắn thời gian xử lý.

Nhận thấy tầm quan trọng của bài toán, đề tài **"Sử dụng thuật toán Image Convolution để xử lý ảnh xám y tế với thư viện OpenMP"** được thực hiện nhằm tìm hiểu nguyên lý song song hóa, triển khai các phương pháp tối ưu và đánh giá hiệu năng dựa trên thực nghiệm.

1.2. Mục tiêu của đề tài

Đề tài được thực hiện nhằm nghiên cứu và đánh giá hiệu quả của việc áp dụng OpenMP vào bài toán xử lý ảnh y tế Các mục tiêu chính bao gồm:

- Cài đặt thuật toán Image Convolution tuần tự làm cơ sở so sánh, dùng để đo thời gian thực thi chuẩn (Ts)
- Cài đặt thuật toán xử lý ảnh song song bằng thư viện OpenMP, phân chia công việc cho nhiều luồng để giảm thời gian thực thi.
- Thực nghiệm và đo thời gian thực thi với các kích thước ảnh xám y tế và số lượng luồng (threads) khác nhau.
- Tính toán các chỉ số đánh giá hiệu năng như tốc độ tăng (**Speedup**) và hiệu suất song song (**Efficiency**).
- So sánh và đánh giá khả năng mở rộng cũng như ưu nhược điểm của phương pháp tuần tự và song song OpenMP.

1.3. Phạm vi và phương pháp nghiên cứu

1.3.1. Phạm vi nghiên cứu

- Chỉ tập trung vào thuật toán Image Convolution áp dụng cho ảnh xám y tế định dạng vuông (NxN)
- Môi trường lập trình: C/C++ kết hợp với thư viện OpenMP trên hệ thống máy tính đa lõi.
- Các kích thước ảnh được chọn tăng dần để đánh giá xu hướng hiệu năng (ví dụ: 512, 1024, 2048, 4096...).

1.3.2. Phương pháp nghiên cứu

- Cài đặt hai mô hình: xử lý tuần tự và xử lý song song với OpenMP.
- Đo thời gian thực thi bằng các hàm chuyên dụng để đảm bảo độ chính xác cao.
- Thu thập số liệu và phân tích hiệu năng dựa trên các thông số thực nghiệm.
- Phân tích các nguyên nhân gây chậm (bottleneck) như chi phí quản lý luồng hoặc truy cập bộ nhớ.
- Thể hiện kết quả thông qua bảng biểu và biểu đồ để dễ dàng quan sát và đối chiếu.

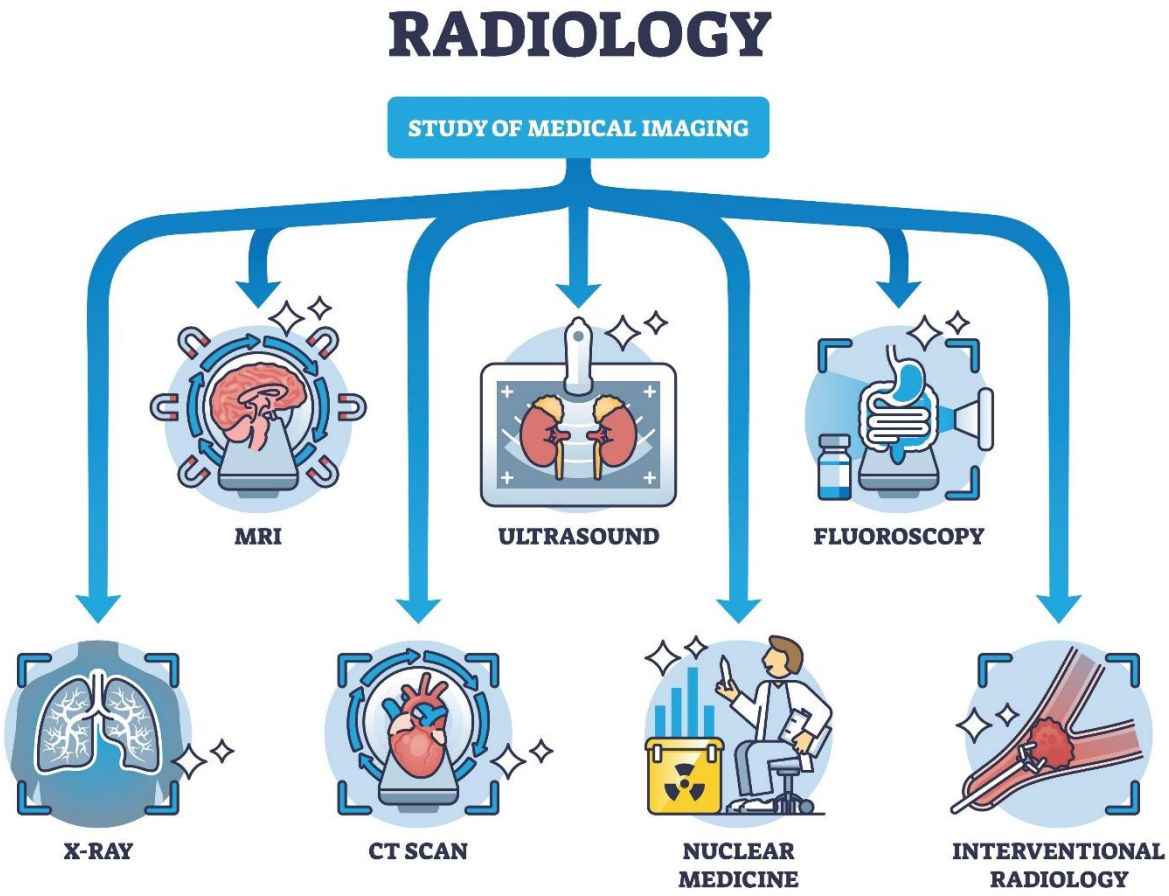
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về Xử lý ảnh y tế

Trong kỷ nguyên số hóa y tế hiện đại, Xử lý ảnh y tế (Medical Image Processing) đóng vai trò then chốt trong việc hỗ trợ chẩn đoán và điều trị bệnh. Đây là lĩnh vực giao thoa giữa khoa học máy tính, toán học ứng dụng và y học, tập trung vào việc áp dụng các thuật toán để tạo ra, phân tích và cải thiện chất lượng hình ảnh thu được từ các thiết bị y tế.

2.1.1. Khái niệm và Vai trò

Xử lý ảnh y tế không chỉ đơn thuần là hiển thị hình ảnh mà là quá trình thao tác trên dữ liệu số (ma trận điểm ảnh) để làm nổi bật các thông tin lâm sàng quan trọng. Các nguồn ảnh phổ biến bao gồm X-quang, Chụp cắt lớp vi tính (CT), Cộng hưởng từ (MRI) và Siêu âm.



Hình 2.1. Chuẩn đoán hình ảnh

Vai trò chính của quá trình này bao gồm:

- **Cải thiện chất lượng ảnh:** Loại bỏ nhiễu (noise) sinh ra do thiết bị hoặc chuyển động của bệnh nhân, tăng độ tương phản để làm rõ các mô mềm hoặc cấu trúc xương.

- **Trích xuất đặc trưng:** Tự động phát hiện biên dạng của khối u, đo kích thước tổn thương hoặc phân vùng ảnh (segmentation) để tách biệt các cơ quan.

- **Hỗ trợ chẩn đoán (CAD):** Cung cấp "ý kiến thứ hai" cho bác sĩ thông qua việc đánh dấu các vùng bất thường mà mắt thường có thể bỏ sót.

2.1.2. Đặc điểm của Ảnh xám y tế (Grayscale Medical Images)

Mặc dù thế giới thực đầy màu sắc, phần lớn dữ liệu ảnh y tế (đặc biệt là X-ray, CT, MRI) được lưu trữ và xử lý dưới dạng **ảnh xám (grayscale)**.

- **Bản chất dữ liệu:** Một ảnh xám y tế được biểu diễn dưới dạng một ma trận hai chiều, trong đó mỗi phần tử là một điểm ảnh (pixel). Giá trị của mỗi pixel biểu thị cường độ sáng (intensity), thường dao động từ 0 (đen tuyệt đối) đến 255 (trắng tuyệt đối) đối với ảnh 8-bit, hoặc cao hơn (12-bit, 16-bit) đối với các thiết bị chẩn đoán chuyên sâu.

- **Tại sao lại là ảnh xám?** Trong y tế, thông tin về cấu trúc và mật độ mô quan trọng hơn màu sắc. Ví dụ, trong ảnh X-quang, mức độ hấp thụ tia X của xương (trắng) khác với phổi (đen) tạo nên sự tương phản cần thiết để chẩn đoán. Việc xử lý trên một kênh màu (intensity channel) cũng giúp giảm khối lượng tính toán so với ảnh màu RGB (3 kênh), cho phép tập trung tài nguyên vào độ phân giải không gian và độ sâu bit.

2.1.3. Các thách thức trong xử lý ảnh y tế

Việc xử lý ảnh y tế đòi hỏi độ chính xác cực cao vì sai số có thể dẫn đến chẩn đoán sai. Hai thách thức lớn nhất là:

1. **Nhiễu (Noise):** Ảnh y tế thường bị lẫn nhiễu (như nhiễu hạt tiêu, nhiễu Gaussian) làm mờ các chi tiết quan trọng.
2. **Khối lượng tính toán:** Với sự phát triển của công nghệ, độ phân giải ảnh ngày càng cao (2K, 4K hoặc dữ liệu 3D từ CT/MRI), đòi hỏi năng lực tính toán lớn để xử lý trong thời gian thực.

Đây chính là động lực để áp dụng các kỹ thuật như **Tích chập (Convolution)** để lọc nhiễu/phát hiện biên và **Tính toán song song (OpenMP)** để giải quyết bài toán về tốc độ.

2.2. Thuật toán Tích chập (Convolution)

Tích chập (Convolution) là kỹ thuật nền tảng và phổ biến nhất trong lĩnh vực thị giác máy tính và xử lý ảnh số, đóng vai trò là "công cụ" chính để thực hiện các thao tác lọc và biến đổi ảnh.

2.2.1. Bản chất lý thuyết

Về mặt bản chất, tích chập trong xử lý ảnh là quá trình áp dụng một ma trận nhỏ (thường được gọi là Kernel, bộ lọc, hoặc cửa sổ trượt) lên toàn bộ các điểm ảnh của ảnh đầu vào.

Cơ chế hoạt động có thể hình dung như một "cửa sổ" trượt lần lượt qua từng vị trí trên ảnh. Tại mỗi vị trí, giá trị của điểm ảnh trung tâm sẽ được tính toán lại dựa trên mối quan hệ trọng số với các điểm ảnh lân cận nằm trong cửa sổ đó. Quá trình này giúp thông tin của một điểm ảnh không còn đứng riêng lẻ mà được đặt trong ngữ cảnh của vùng xung quanh nó.

2.2.2. Vai trò trong xử lý ảnh y tế

Trong ngữ cảnh của đề tài, thuật toán tích chập thực hiện hai nhiệm vụ lý thuyết chính:

- Làm trơn và khử nhiễu: Bằng cách tính trung bình hoặc trung bình có trọng số các pixel lân cận, thuật toán giúp làm mượt các điểm nhiễu (noise) dị biệt, giúp ảnh trở nên "sạch" hơn trước khi chẩn đoán.

- Làm nổi bật đặc trưng: Bằng cách tính toán sự thay đổi mức xám giữa các vùng kề nhau, tích chập giúp phát hiện các cạnh, biên dạng của cơ quan nội tạng hoặc khối u, làm cho ranh giới giữa các vùng bệnh lý và vùng khỏe mạnh trở nên rõ ràng hơn.

2.3. Tổng quan về Lập trình song song và OpenMP

Việc áp dụng thuật toán tích chập lên các ảnh y tế có độ phân giải lớn đòi hỏi khối lượng tính toán khổng lồ, dễ gây ra hiện tượng "nghẽn cổ chai" nếu chỉ xử lý tuần tự trên một bộ vi xử lý đơn lẻ. Lập trình song song là giải pháp lý thuyết để giải quyết bài toán hiệu năng này.

2.3.1. Lý thuyết về tính toán song song

Lập trình song song dựa trên nguyên lý chia để trị: một tác vụ lớn được chia nhỏ thành nhiều tác vụ con để thực hiện đồng thời trên nhiều lõi xử lý (CPU cores).

Đối với xử lý ảnh, đặc tính quan trọng nhất là sự độc lập dữ liệu (Data Independence). Việc tính toán giá trị mới cho một điểm ảnh ở góc trái bức ảnh hoàn toàn không phụ thuộc vào việc tính toán cho một điểm ảnh ở góc phải. Do đó, ta có thể xử lý hàng nghìn điểm ảnh cùng một lúc mà không sợ xung đột dữ liệu.

2.3.2. Mô hình OpenMP

OpenMP (Open Multi-Processing) là chuẩn giao diện lập trình ứng dụng phổ biến nhất để thực hiện tính toán song song trên các hệ thống bộ nhớ chia sẻ (Shared Memory).

Lý thuyết hoạt động của OpenMP dựa trên mô hình Fork-Join:

1. Fork (Phân nhánh): Chương trình bắt đầu với một luồng chính. Khi gặp vùng cần xử lý song song (như vòng lặp quét qua các pixel của ảnh), luồng chính sẽ tạo ra một nhóm các luồng con.
2. Parallel Execution (Thực thi song song): Các luồng con này chia nhau khối lượng công việc (mỗi luồng xử lý một phần của bức ảnh) và chạy đồng thời trên các nhân CPU khác nhau.
3. Join (Hợp nhất): Sau khi hoàn thành nhiệm vụ, các luồng con kết thúc và hợp nhất trở lại vào luồng chính để tiếp tục các bước xử lý tiếp theo hoặc xuất kết quả.

Việc sử dụng OpenMP giúp tận dụng tối đa sức mạnh phần cứng của các máy tính hiện đại, giảm đáng kể thời gian chờ đợi khi xử lý các tập dữ liệu ảnh y tế lớn.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

3.1. Phân tích Lý thuyết Thuật toán Tuần tự (Theoretical Analysis of Sequential Algorithm)

3.1.1. Mô hình Toán học của Phép Tích chập Rời rạc (Discrete Convolution)

Trong xử lý ảnh số, phép tích chập không phải là tích chập liên tục trong giải tích, mà là **Tích chập rời rạc 2 chiều (2D Discrete Convolution)**.

Giả sử ta có:

- *I*: Ảnh đầu vào (Input Image) kích thước *H x W*, trong đó *I(x,y)* là cường độ sáng tại toạ độ *(x,y)*.
- *K*: Bộ lọc (Kernel/Filter) kích thước *m x n* (thường là ma trận vuông lẻ *k x k*), đóng vai trò là cửa sổ trượt.
- *O*: Ảnh đầu ra (Output Image).

Công thức tổng quát để tính giá trị điểm ảnh tại vị trí *(x,y)* của ảnh đầu ra là:

$$O(x,y) = (I * K)(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j).I(x-i, y-j)$$

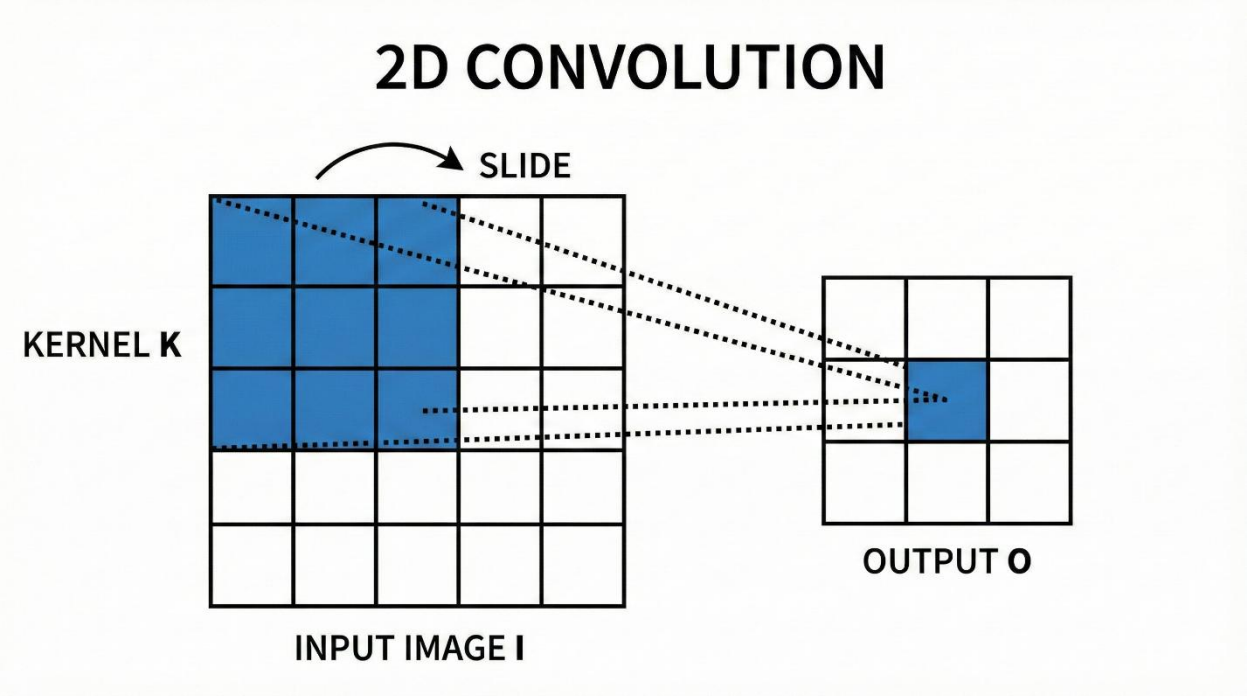
Trong đó:

- $a = (m-1)/2$ và $b = (n-1)/2$ là bán kính của Kernel theo trục dọc và ngang.
- Cặp chỉ số *(i, j)* đại diện cho độ lệch cục bộ (offset) so với tâm kernel.

Ý nghĩa vật lý trong ảnh y tế:

Phép toán này thực chất là tính tổng có trọng số (Weighted Sum) của các pixel lân cận.

- Nếu *K* là ma trận Gaussian (trọng số giảm dần từ tâm ra ngoài) -> **Làm mượt ảnh (Denoising)**.
- Nếu *K* là ma trận Sobel/Laplacian (trọng số âm dương đối xứng) -> **Phát hiện biên (Edge Detection)** để tách vùng xương hoặc khối u.



Hình 3.1. Minh họa phép tích chập

3.1.2. Phân tích Độ phức tạp Tính toán (Computational Complexity Analysis)

Để đánh giá hiệu năng, ta cần phân tích số lượng phép tính mà CPU phải thực hiện (Floating-point Operations - FLOPs).

Giả sử ảnh có kích thước $N \times N$ và Kernel có kích thước $k \times k$.

1. **Độ phức tạp thời gian (Time Complexity):**
    - Để tính 1 pixel đầu ra: Cần thực hiện k^2 phép nhân và $(k^2 - 1)$ phép cộng. Tổng cộng $\approx 2k^2$ phép tính (FLOPs).
    - Để tính cả bức ảnh $N \times N$: Cần lặp lại quá trình trên N^2 lần.
    - **Tổng số phép tính:** $T(N, k) \approx 2.N^2.k^2$
    - **Big-O Notation:** $O(N^2 . k^2)$.
    - *Nhận xét:* Độ phức tạp tăng theo bình phương kích thước ảnh. Với ảnh y tế độ phân giải cao (ví dụ X-quang phổi 4K: 4096 x 4096) và kernel 5 x 5, CPU đơn nhân phải thực hiện hàng tỷ phép tính, tạo ra gánh nặng xử lý rất lớn.
2. **Độ phức tạp không gian (Space Complexity):**
    - Cần bộ nhớ lưu trữ ảnh đầu vào I (N^2 phần tử) và ảnh đầu ra O (N^2 phần tử).

- Không gian phụ trợ cho Kernel là không đáng kể (k^2).
- **Big-O Notation:** $O(N^2)$.

3.1.3. Phân tích Đặc tính Truy xuất Bộ nhớ (Memory Access Pattern)

Đây là phần quan trọng nhất trong môn "Tính toán hiệu năng cao". Tại sao thuật toán tuần tự lại chậm, ngoài việc phải tính toán nhiều? Câu trả lời nằm ở **Cấu trúc bộ nhớ**.

1. Lưu trữ tuyến tính (Row-major Order):

Trong C/C++, mảng 2 chiều thực chất được lưu trải dài thành 1 dòng liên tục trong RAM.

Pixel *(x, y)* và *(x+1, y)* nằm cạnh nhau trong bộ nhớ (cách nhau 1 byte).

Tuy nhiên, pixel *(x, y)* và *(x, y+1)* (pixel ngay bên dưới) lại cách nhau một khoảng bằng độ rộng của ảnh (*W* bytes).

2. Tính cục bộ không gian (Spatial Locality) và Cache Miss:

Khi CPU tính tích chập, nó cần truy cập một vùng hình vuông *k x k* xung quanh *(x,y)*.

- Các phần tử trên cùng một hàng của vùng này nằm gần nhau -> Tận dụng tốt bộ nhớ Cache (Cache Hit).
- Các phần tử nằm ở hàng trên hoặc hàng dưới của vùng này nằm rất xa nhau trong bộ nhớ vật lý -> Gây ra hiện tượng **Cache Miss** (trượt dòng tin).
- *Hệ quả:* CPU phải tốn nhiều chu kỳ chờ dữ liệu được nạp từ RAM vào Cache, làm giảm hiệu suất thực tế so với hiệu suất lý thuyết của bộ vi xử lý.

3.1.4. Phân tích Điểm nghẽn (Bottlenecks) của Xử lý Tuần tự

Khi chạy thuật toán này trên một luồng đơn (Sequential), chúng ta gặp các giới hạn vật lý sau:

1. Lãng phí tài nguyên phần cứng:

Các CPU hiện đại (như Intel Core i7/i9 hoặc AMD Ryzen) thường có từ 8 đến 24 nhân vật lý. Giải thuật tuần tự chỉ sử dụng 1 nhân duy nhất, để lãng phí 90 - 95% sức mạnh xử lý của hệ thống.

2. Overhead do Rẽ nhánh (Branching Overhead):

Tại mỗi bước trượt kernel, thuật toán phải kiểm tra điều kiện biên (Boundary Check):

IF (tọa độ nằm trong ảnh) THEN ... ELSE ...

Việc kiểm tra if/else này diễn ra hàng triệu lần. Nó làm gián đoạn đường ống lệnh (Instruction Pipeline) của CPU do dự đoán rẽ nhánh sai (Branch Misprediction), làm giảm tốc độ xử lý.

3. Không đồng bộ hóa giữa CPU và Bộ nhớ (Von Neumann Bottleneck):

Tốc độ tính toán của CPU nhanh hơn rất nhiều so với tốc độ truy xuất dữ liệu từ RAM. Trong thuật toán tuần tự, CPU thường xuyên phải "ngồi chơi" (stalls) để chờ dữ liệu pixel được nạp lên, do không có luồng khác lấp vào khoảng thời gian chết đó.

3.2. Thiết kế giải thuật Tuần tự (Sequential Algorithm Design)

Dựa trên phân tích về đặc tính truy xuất bộ nhớ và điểm nghẽn hiệu năng, phần này trình bày thiết kế chi tiết của giải thuật tuần tự, đóng vai trò là cơ sở (baseline) để đánh giá hiệu quả tăng tốc.

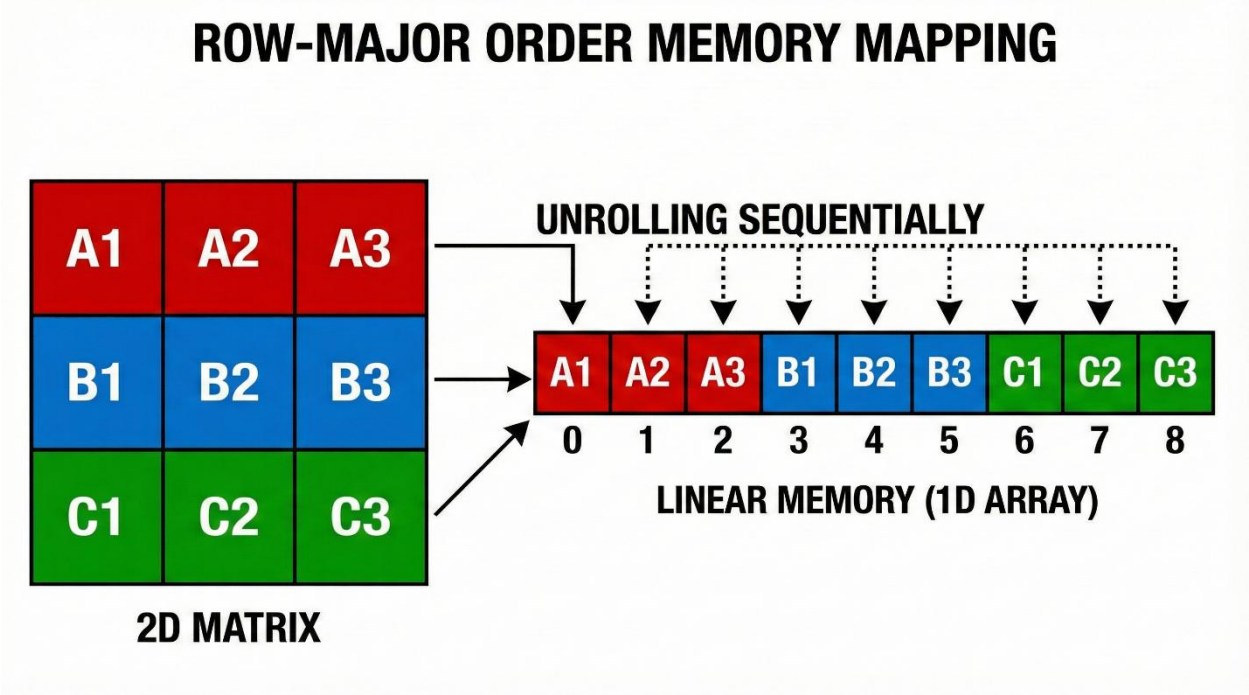
3.2.1. Chiến lược Tổ chức Dữ liệu (Data Structure Strategy)

Trong tính toán hiệu năng cao, cách tổ chức dữ liệu quyết định khả năng tận dụng bộ nhớ Cache. Thay vì sử dụng cấu trúc mảng 2 chiều truyền thống (int**) vốn gây phân mảnh bộ nhớ, chúng tôi thiết kế cấu trúc dữ liệu dựa trên **Mảng 1 chiều (Flat Array)**.

- **Lý do thiết kế:** Như đã phân tích tại mục 2.1.3, bộ nhớ máy tính được tổ chức tuyến tính[2]. Việc sử dụng mảng 1 chiều đảm bảo các phần tử trên cùng một hàng của ảnh được lưu trữ liền kề nhau vật lý (Contiguous Memory), tối ưu hóa tính cục bộ không gian (Spatial Locality)[3].
- Ánh xạ địa chỉ (Address Mapping): Một điểm ảnh tại tọa độ *(x, y)* trong không gian 2D sẽ được ánh xạ sang chỉ số mảng 1 chiều index theo công thức:

$$Index = y \times Width + x$$

Trong đó *Width* là chiều rộng của ảnh.



Hình 3.2. Minh họa ánh xạ mảng 2D sang 1D

3.2.2. Mô tả thuật toán (Algorithm Description)

Quy trình xử lý tuần tự được thực hiện theo cơ chế quét dòng (Scan-line order), cụ thể bao gồm các bước sau:

1. **Khởi tạo (Initialization):**
    - Cấp phát bộ nhớ cho ảnh đầu ra Output có kích thước bằng ảnh đầu vào Input.
    - Tính toán bán kính Kernel $r = k/2$.
2. **Duyệt không gian ảnh (Spatial Traversal):**
    - Sử dụng hai vòng lặp lồng nhau để duyệt qua từng pixel (x, y) của ảnh $(0 \leq y < H, 0 \leq x < W)$.
3. **Tích chập cục bộ (Local Convolution):**
    - Tại mỗi pixel (x, y), khởi tạo biến tích lũy sum = 0.
    - Duyệt qua vùng lân cận kích thước $k \times k$ bằng hai vòng lặp con (ky, kx).
4. **Xử lý biên (Boundary Handling):**

- Kiểm tra tọa độ lân cận *(x+kx, y+ky)*. Nếu tọa độ nằm ngoài phạm vi ảnh, bỏ qua phép tính (tương đương với kỹ thuật *Zero Padding*). Điều này giúp tránh lỗi truy cập bộ nhớ trái phép (Segmentation Fault).

5. **Chuẩn hóa và Ghi kết quả:**
    - Giá trị sum sau khi tính toán có thể là số thực hoặc nằm ngoài khoảng [0, 255].
    - Thực hiện kẹp giá trị (Clamping): val = min(max(sum, 0), 255) và ghi vào mảng Output tại vị trí index.

3.2.3. Cài đặt chi tiết (Implementation Details)

Dưới đây là mã giả (Pseudocode) mô tả chi tiết logic xử lý, minh họa cách ánh xạ mảng 1 chiều để giảm thiểu Cache Miss:

```
FUNCTION ConvolutionSequential(input, output, width, height, kernel, kSize)
    // 1. Xác định bán kính kernel
    SET r = kSize / 2

    // 2. Duyệt qua từng hàng (Row-major traversal)
    FOR y FROM 0 TO height - 1 DO:

        // 3. Duyệt qua từng cột
        FOR x FROM 0 TO width - 1 DO:

            SET sum = 0.0

            // 4. Áp dụng Kernel lên vùng lân cận
            FOR ky FROM -r TO r DO:
                FOR kx FROM -r TO r DO:

                    SET neighborY = y + ky
                    SET neighborX = x + kx

                    // 5. Kiểm tra biên (Boundary Check)
                    IF (neighborY >= 0 AND neighborY < height AND
                        neighborX >= 0 AND neighborX < width) THEN:

                        // Ánh xạ 2D -> 1D index
                        SET imgIdx = neighborY * width + neighborX
                        SET kIdx = (ky + r) * kSize + (kx + r)

                        sum = sum + (input[imgIdx] * kernel[kIdx])
                    END IF

                END FOR
            END FOR

            // 6. Chuẩn hóa và ghi kết quả
            SET outIdx = y * width + x

            IF (sum < 0) THEN SET sum = 0
            IF (sum > 255) THEN SET sum = 255

            SET output[outIdx] = (unsigned char)sum

        END FOR
    END FOR
END FUNCTION
```

Hình 3.3. Mã giả Convolution

3.3. Cơ sở lý thuyết OpenMP và Tính toán Song song

Để khắc phục hạn chế của việc xử lý đơn luồng trên các hệ thống đa nhân hiện đại, đề tài sử dụng thư viện **OpenMP (Open Multi-Processing)**. Đây là tiêu chuẩn công nghiệp cho lập trình song song trên kiến trúc bộ nhớ chia sẻ.

3.3.1. Kiến trúc Bộ nhớ Chia sẻ (Shared Memory Architecture)

Khác với mô hình bộ nhớ phân tán (Distributed Memory) thường dùng trong MPI, OpenMP hoạt động trên mô hình **SMP (Symmetric Multi-Processing)**.

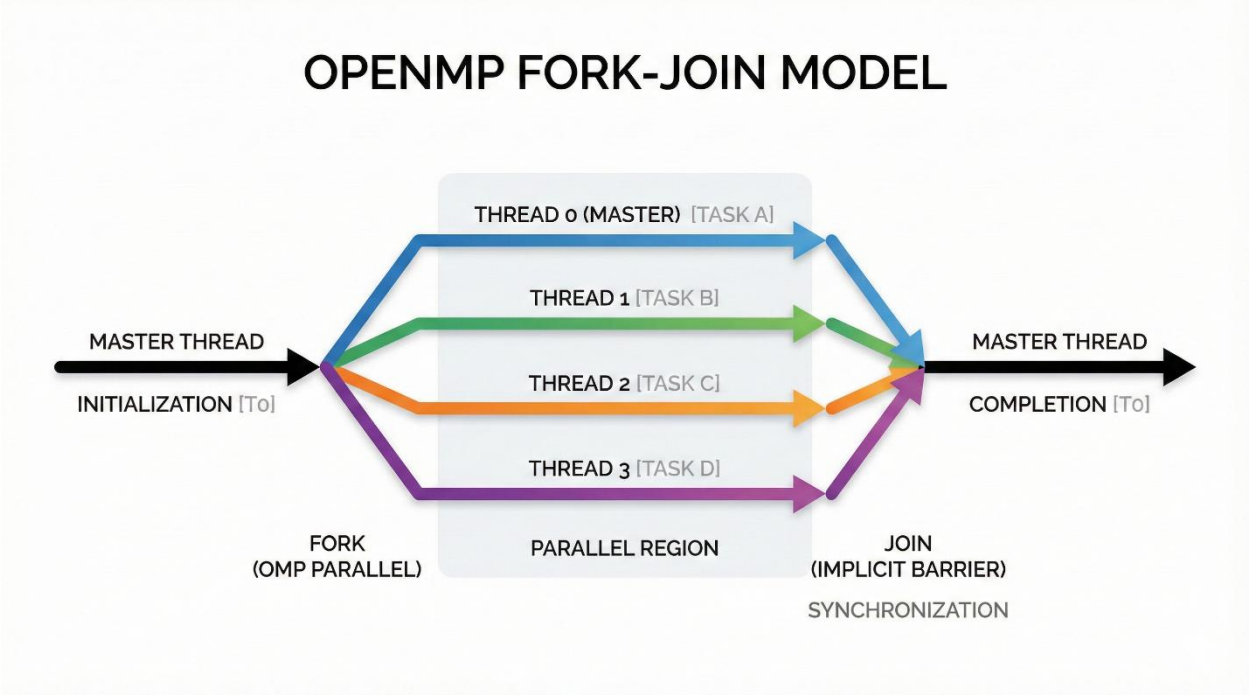
- **Đặc điểm:** Tất cả các đơn vị xử lý (Cores/Threads) đều có quyền truy cập vào một không gian địa chỉ bộ nhớ vật lý chung (Global Shared RAM).

- **Cơ chế giao tiếp:** Các luồng trao đổi dữ liệu bằng cách đọc/ghi vào các biến chung (Shared Variables).
- **Ưu điểm đối với xử lý ảnh:** Do ảnh y tế có kích thước lớn, việc tất cả các luồng có thể truy cập trực tiếp vào mảng ảnh Input mà không cần sao chép hay truyền dữ liệu qua mạng giúp giảm đáng kể độ trễ (Latency).

3.3.2. Mô hình Thực thi Fork-Join

OpenMP hoạt động dựa trên mô hình **Fork-Join**, bao gồm các giai đoạn chuyển đổi giữa thực thi tuần tự và song song:

1. **Giai đoạn Tuần tự (Master Thread):** Chương trình bắt đầu chạy trên một luồng chính (Master Thread). Các tác vụ khởi tạo, cấp phát bộ nhớ, và đọc ảnh đầu vào được thực hiện tại đây.
2. **Giai đoạn Fork (Phân nhánh):** Khi gặp chỉ thị song song (#pragma omp parallel), luồng chính sẽ sinh ra (spawn) một nhóm các luồng con (Team of Threads).
3. **Giai đoạn Song song (Parallel Region):** Các luồng trong nhóm cùng thực hiện khối lệnh được chỉ định. Trong bài toán tích chập, đây là lúc các luồng chia nhau xử lý các phần khác nhau của bức ảnh.
4. **Giai đoạn Join (Hợp nhất):** Tại cuối vùng song song, có một rào chắn đồng bộ ngầm định (Implicit Barrier). Luồng chính chờ tất cả các luồng con hoàn thành công việc, sau đó gộp dòng thực thi lại thành một luồng duy nhất để tiếp tục (ví dụ: ghi ảnh ra đĩa).



Hình 3.4. Minh Họa mô hình Fork-Join của OpenMP

3.3.3. Các kỹ thuật OpenMP áp dụng cho Image Convolution

Để tối ưu hóa phép tích chập $O(N^2 \times k^2)$, đề tài sử dụng các cấu trúc điều khiển nâng cao sau:

a. Phân chia công việc (Work-sharing Loop):

Chỉ thị #pragma omp parallel for tự động chia các vòng lặp độc lập dữ liệu cho các luồng.

- *Cơ chế:* Hệ thống runtime sẽ chia dải chỉ số vòng lặp (Loop Iterations) thành các phần (Chunks) và gán cho từng luồng.

b. Kỹ thuật Collapse Loop:

Trong xử lý ảnh, ta thường có hai vòng lặp lồng nhau duyệt theo chiều cao (H) và chiều rộng (W).

- *Vấn đề:* Nếu chỉ song song hóa vòng lặp ngoài (H), và H nhỏ (ví dụ ảnh panorama 200 x 10000), tải công việc sẽ mất cân bằng và số lượng luồng có thể tham gia bị giới hạn bởi H.
- *Giải pháp:* Sử dụng mệnh đề collapse(2) để gộp hai vòng lặp y và x thành một không gian lặp tuyến tính duy nhất kích thước H x W. Điều này giúp tăng độ "mịn" của hạt nhân tính toán (Granularity) và cân bằng tải tốt hơn.

c. Chiến lược Lập lịch (Scheduling Strategy):

Việc phân chia các phần tử ảnh cho các luồng được kiểm soát bởi mệnh đề schedule:

- **Static Schedule:** Chia đều không gian lặp thành các phần bằng nhau ngay từ khi biên dịch. Phù hợp cho bài toán tích chập vì khối lượng tính toán tại mỗi pixel là hằng số (k^2 phép tính).
- **Dynamic Schedule:** Cấp phát việc theo nhu cầu (khi luồng làm xong việc thì xin thêm). Phù hợp khi khối lượng tính toán không đều, nhưng gây ra chi phí quản lý (overhead) cao hơn.

3.3.4. Vấn đề tranh chấp dữ liệu (Race Condition)

Một thách thức lớn trong mô hình bộ nhớ chia sẻ là **Tranh chấp dữ liệu**.

- **Định nghĩa:** Xảy ra khi nhiều luồng cùng truy cập vào một vùng nhớ cùng lúc, và có ít nhất một luồng thực hiện ghi (Write), dẫn đến kết quả không xác định.
- **Trong bài toán này:** Biến tích lũy sum (dùng để cộng dồn giá trị tích chập) là nơi dễ xảy ra xung đột nhất. Nếu sum là biến chung (Shared), các luồng sẽ ghi đè lên kết quả của nhau.
- **Giải pháp:** Cần khai báo sum là biến riêng tư (**Private**) cho từng luồng hoặc khai báo cục bộ bên trong vòng lặp song song.

3.4. Thiết kế giải thuật song song (Parallel Algorithm Design)

Dựa trên phân tích điểm nghẽn của giải thuật tuần tự (mục 2.1.4) và cơ sở lý thuyết OpenMP (mục 2.3), phần này trình bày chiến lược song song hóa cụ thể áp dụng cho phép tích chập ảnh.

3.4.1. Phân tích phụ thuộc dữ liệu (Data Dependency Analysis)

Bước đầu tiên trong thiết kế song song là xác định xem các tác vụ có thể thực hiện đồng thời hay không.

- **Quan sát:** Giá trị của điểm ảnh đầu ra $O(x, y)$ tại vị trí (x, y) được tính toán hoàn toàn dựa trên ảnh đầu vào I và Kernel K. Việc tính toán $O(x, y)$ **không cần** kết quả của $O(x', y')$ tại bất kỳ vị trí nào khác.
- **Kết luận:** Bài toán thỏa mãn tính chất **Độc lập dữ liệu (Data Independence)**. Đây là dạng bài toán *Embarrassingly Parallel* (Song song hiển nhiên), cho phép chia nhỏ không gian tính toán mà không cần cơ chế đồng bộ phức tạp giữa các luồng.

3.4.2. Chiến lược phân rã miền dữ liệu (Domain Decomposition)

Thay vì chia nhỏ chức năng (Functional Decomposition), chúng tôi áp dụng chiến lược **Phân rã theo dữ liệu (Data Decomposition)**:

- **Chia theo không gian 2D:** Bức ảnh kích thước H x W được coi là một không gian lặp 2 chiều.
- **Kỹ thuật Collapse:** Sử dụng mệnh đề collapse(2) của OpenMP để gộp hai vòng lặp lồng nhau (theo chiều cao y và chiều rộng x) thành một không gian lặp tuyến tính duy nhất có kích thước $N = H$ x W.
- **Lợi ích thiết kế:**
    - Tăng độ hạt (Granularity) của tác vụ, giúp OpenMP có nhiều không gian để chia việc hơn so với chỉ chia theo dòng (Row-wise).
    - Giải quyết vấn đề cân bằng tải nếu ảnh có kích thước đặc biệt (ví dụ: ảnh rất dẹt hoặc rất cao).

3.4.3. Quản lý phạm vi biến (Data Scoping Strategy)

Đây là phần quan trọng nhất để ngăn chặn lỗi **Tranh chấp dữ liệu (Race Condition)**. Chúng tôi phân loại các biến trong thuật toán như sau:

Bảng 3. 1 Quản lý phạm vi biến

Loại biến	Danh sách biến	Lý do thiết kế
SHARED (Dùng chung)	Input (Mảng ảnh gốc), Output (Mảng kết quả), Kernel	- Input, Kernel: Chỉ đọc (Read-only) nên an toàn. - Output: Mặc dù là ghi (Write), nhưng mỗi luồng ghi vào một index riêng biệt (y x $W + x$), không có hai luồng nào ghi trùng vị trí.

PRIVATE (Riêng tư)	x, y (Biến chạy chính), kx, ky (Biến chạy kernel), sum (Biến tích lũy), imgIdx, kIdx	- Các biến này thay đổi liên tục trong mỗi luồng. - **Quan trọng:** Biến sum bắt buộc phải là riêng tư. Nếu để sum là Shared, các luồng sẽ cộng dồn sai lệch kết quả của nhau.

3.4.4. Chiến lược lập lịch (Scheduling Strategy)

OpenMP cung cấp nhiều chiến lược lập lịch (static, dynamic, guided). Đối với thuật toán Image Convolution:

- **Đặc điểm tải (Workload Characterization):** Tại mọi điểm ảnh (pixel), khối lượng tính toán là hằng số (thực hiện đúng k^2 phép nhân và cộng), không phụ thuộc vào giá trị của pixel đó.
- **Lựa chọn: Static Scheduling** (schedule(static)).
- **Cơ chế:** Không gian lặp được chia đều thành các phần (chunks) bằng nhau và gán cố định cho các luồng ngay từ đầu.
- **Ưu điểm:** Loại bỏ hoàn toàn chi phí quản lý (overhead) cấp phát động trong quá trình chạy, giúp đạt hiệu năng tối đa cho các tác vụ có tải cân bằng.

3.4.5. Thiết kế xử lý Biên trong môi trường song song

Trong môi trường song song, việc xử lý biên (Boundary Check) cần cẩn trọng để không làm giảm hiệu năng:

- Chúng tôi giữ nguyên cơ chế kiểm tra if bên trong kernel (if (neighborX >= 0 ...)).
- Mặc dù điều này gây ra *Branching Overhead* (như đã phân tích ở mục 2.1.4), nhưng do sử dụng schedule(static), các luồng xử lý phần giữa ảnh (không dính biên) sẽ chạy rất nhanh (Branch Prediction luôn đúng), bù đắp cho các luồng xử lý phần mép

ảnh. Việc tách riêng code xử lý biên (Code splitting) sẽ làm mã nguồn phức tạp không cần thiết với kernel kích thước nhỏ.

3.4.6. Mã giả và cài đặt song song (Parallel Implementation)

Để hiện thực hóa chiến lược phân rã miền dữ liệu và lập lịch tĩnh, giải thuật song song được thiết kế sử dụng các chỉ thị biên dịch (Compiler Directives) của OpenMP. Dưới đây là mã giả mô tả cấu trúc song song hóa:

Mã giả (Pseudocode):

```
Procedure ConvolutionOpenMP(Input, Output, Width, Height, Kernel, KSize)
// 1. Thiết lập môi trường song song
// SHARED: Dữ liệu ảnh lớn (Input, Output) và Kernel được chia sẻ để tiết kiệm RAM
// PRIVATE: Các biến chạy (x, y, kx, ky) và biến tích lũy (sum) phải riêng tư

#pragma omp parallel for \
        collapse(2) \
        schedule(static) \
        shared(Input, Output, Kernel) \
        private(x, y, kx, ky, sum)

    // 2. Vòng lặp phân rã dữ liệu (đã được gộp bởi collapse)
    FOR y FROM 0 TO Height - 1 DO
    FOR x FROM 0 TO Width - 1 DO

    // 3. Khởi tạo biến cục bộ (Local Accumulator)
    sum = 0.0

    // 4. Tính toán Kernel (Thực hiện tuần tự trong từng luồng)
    FOR ky FROM - halfK TO halfK DO
    FOR kx FROM - halfK TO halfK DO
    // Kiểm tra biên
    IF(Pixel(x + kx, y + ky) nằm trong ảnh) THEN
    // Tính toán cục bộ không cần khóa (Lock-free)
    sum += Input[index] * Kernel[kIndex]
    END IF
    END FOR
    END FOR

    // 5. Ghi kết quả (Không xung đột do index độc lập)
    Output[y * Width + x] = Clamp(sum)

    END FOR
    END FOR
    End Procedure
```

Hình 3.5. Mã giả Convolution khi song song hóa bằng OpenMp

3.5. Kết luận chương 3

Trong chương này, nhóm thực hiện đã phân tích cơ sở lý thuyết toán học của phép tích chập và xác định được các điểm nghẽn hiệu năng của giải thuật tuần tự, bao gồm việc lãng phí tài nguyên đa nhân và độ trễ truy cập bộ nhớ. Dựa trên cơ sở đó, chúng tôi đã đề

xuất thiết kế giải thuật song song sử dụng OpenMP với chiến lược phân rã miền dữ liệu (Domain Decomposition) và quản lý bộ nhớ tối ưu.

Những thiết kế này sẽ là cơ sở để tiến hành cài đặt, đo đạc và đánh giá hiệu năng thực tế trên các kích thước ảnh và số lượng luồng khác nhau trong

CHƯƠNG 4: CÀI ĐẶT VÀ THỰC HIỆN

4.1. Môi trường cài đặt

Để đảm bảo tính nhất quán và khả năng tái lập kết quả thực nghiệm, chương trình được phát triển và kiểm thử trên môi trường cụ thể như sau:

4.1.1. Môi trường 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) phiên bản hỗ trợ OpenMP (thông qua cờ -fopenmp).
- **Môi trường phát triển tích hợp (IDE):** Visual Studio Code.
- **Thư viện sử dụng:**
    - stb_image.h: Hỗ trợ đọc các định dạng ảnh phổ biến (JPG, PNG, BMP) và chuyển đổi về dạng mảng pixel.
    - stb_image_write.h: Hỗ trợ lưu mảng pixel thành file ảnh đầu ra.
    - omp.h: Thư viện OpenMP cung cấp các chỉ thị biên dịch để lập trình song song trên hệ thống bộ nhớ chia sẻ.
    - std::filesystem: Thư viện chuẩn C++ dùng để duyệt file và quản lý thư mục tự động.

4.1.2. Môi trường phần cứng

Hệ thống phần cứng sử dụng để chạy thực nghiệm và đánh giá hiệu năng có cấu hình như sau:

- **Bộ vi xử lý (CPU):** Intel Core i7 – 9850H
- **Tốc độ xung nhịp:** 2.60 GHz
- **Số nhân / Số luồng:** 12 luồng
- **Bộ nhớ trong (RAM):** 32GB Ram
- **Ổ cứng:** SSD

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

4.2.1. Mục đích cài đặt

Việc cài đặt phiên bản tuần tự (Sequential) nhằm hai mục đích chính:

1. **Kiểm chứng độ chính xác:** Đảm bảo thuật toán tích chập (Convolution) hoạt động đúng logic, tạo ra ảnh kết quả (làm mịn, làm nét) đúng yêu cầu y tế.
2. **Làm cơ sở so sánh (Baseline):** Đo đạc thời gian thực thi của phiên bản chạy trên 1 luồng đơn để tính toán độ tăng tốc (Speedup) khi áp dụng song song hóa.

4.2.2. Mô tả thuật toán

Thuật toán Image Convolution (Tích chập ảnh) hoạt động bằng cách trượt một cửa sổ nhỏ (gọi là Kernel hoặc Filter, thường là ma trận 3x3) qua từng điểm ảnh của ảnh đầu vào.

Tại mỗi vị trí *(x, y)* , giá trị điểm ảnh mới được tính theo công thức:

$$\mathrm{P_{out}}(\mathrm{x},\mathrm{y}) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \mathrm{P_{in}}(\mathrm{x}+\mathrm{i},\mathrm{y}+\mathrm{j}) \times \mathrm{K}(\mathrm{i},\mathrm{j})$$

Trong đề tài này, quy trình xử lý bao gồm 2 bước liên tiếp để tối ưu hóa chất lượng ảnh X-quang:

1. **Bước 1 - Gaussian Blur:** Sử dụng Kernel làm mịn để khử nhiễu hạt (Noise reduction).
2. **Bước 2 - Sharpen:** Sử dụng Kernel làm nét để tăng cường độ tương phản các đường biên xương.

4.2.3. Lưu đồ thực hiện

1. **Start:** Bắt đầu chương trình.
2. **Input:** Đọc ảnh từ thư mục *data*, chuyển thành mảng 1 chiều.
3. **Process (Vòng lặp):**
    - Duyệt qua từng hàng y từ 1 đến height-1.
    - Duyệt qua từng cột x từ 1 đến width-1.
    - Tại mỗi điểm (x,y): Nhân các pixel lân cận với ma trận Kernel tương ứng.
    - Cộng dồn kết quả -> Chia cho hệ số (Divisor) -> Cắt giá trị trong khoảng [0, 255].

- Gán giá trị vào ảnh đích.

4. **Output:** Lưu ảnh kết quả vào thư mục *output_sequential*.
5. **End:** Kết thúc.

4.2.4. Cài đặt và đoạn mã minh họa

Dưới đây là hàm xử lý tuần tự được cài đặt trong C++:

```cpp
void processSequential(const Image& input, Image& output, const int kernel[3][3], int divisor) {
    // Khởi tạo ảnh đầu ra
    int w = input.width; int h = input.height;
    output.width = w; output.height = h; output.channels = 1;
    output.pixels.resize(w * h);

    // Vòng lặp xử lý từng điểm ảnh
    for (int y = 1; y < h - 1; y++) {
        for (int x = 1; x < w - 1; x++) {
            int sum = 0;
            for (int ky = -1; ky <= 1; ky++) {
                for (int kx = -1; kx <= 1; kx++) {      You, yesterday • chore: initialize project
                    sum += input.pixels[(y + ky) * w + (x + kx)] * kernel[ky + 1][kx + 1];
                }
            }
            // Chuẩn hóa giá trị
            sum /= divisor;
            if (sum < 0) sum = 0; if (sum > 255) sum = 255;
            output.pixels[y * w + x] = (unsigned char)sum;
        }
    }
}
```

Hình 4.1. Mã code tuần tự thuật toán Image Convolution

4.3. Cài đặt Image Convolution song song hóa cơ bản

4.3.1. Ý tưởng song song hóa

Bài toán xử lý ảnh có đặc thù là **song song dữ liệu (Data Parallelism)**. Giá trị của điểm ảnh đầu ra tại vị trí *(x, y)* chỉ phụ thuộc vào các điểm ảnh lân cận trên ảnh gốc, hoàn toàn độc lập với kết quả tại các vị trí *(x', y')* khác.

Do đó, chúng ta có thể chia bức ảnh thành nhiều phần nhỏ và giao cho các luồng (Threads) của CPU xử lý đồng thời mà không sợ xung đột dữ liệu (Race Condition) trong quá trình tính toán. Mô hình lập trình được sử dụng là **Fork-Join** của OpenMP.

4.3.2. Phân chia và phân phối dữ liệu

Chúng tôi sử dụng chỉ thị ***#pragma omp parallel for*** kết hợp với mệnh đề ***collapse(2)***.

- **Chiến lược phân chia:** OpenMP sẽ tự động chia không gian vòng lặp (chiều cao $\mathcal{H}$ và chiều rộng $\mathcal{W}$) thành các khối công việc.
- **Collapse(2):** Gộp hai vòng lặp lồng nhau (theo y và theo x) thành một không gian lặp duy nhất lớn hơn ($\mathcal{H} \times \mathcal{W}$). Điều này giúp tăng khối lượng công việc cho mỗi luồng và cân bằng tải tốt hơn, đặc biệt với các ảnh có kích thước chiều ngang hoặc dọc chênh lệch lớn.

4.3.3. Tính toán cục bộ

Mỗi luồng sẽ thực hiện tính toán tích chập trên vùng dữ liệu được phân công.

- **Dữ liệu đọc (Input):** Tất cả các luồng cùng truy cập đọc (Read-only) vào mảng input.pixels. Do không có ghi đè nên không cần cơ chế khóa (Lock), đảm bảo hiệu năng cao.
- **Biến cục bộ:** Các biến như sum, kx, ky được khai báo bên trong vòng lặp song song, do đó chúng là biến riêng tư (Private) của từng luồng.

4.3.4. Thu thập kết quả

Kết quả tính toán của từng luồng được ghi trực tiếp vào mảng *output.pixels* tại vị trí *Index = y * width + x*.

Vì mỗi luồng chịu trách nhiệm cho các chỉ số index khác nhau (rời rạc), nên việc ghi dữ liệu vào bộ nhớ chia sẻ diễn ra an toàn mà không cần các thao tác gom tụ (Reduction) hay đồng bộ hóa phức tạp (Synchronization overhead) ở mức pixel.

Đoạn mã song song hóa:

```
void processOpenMP(const Image& input, Image& output, const int kernel[3][3], int divisor) {
    // Khởi tạo ảnh đầu ra
    int w = input.width; int h = input.height;
    output.width = w; output.height = h; output.channels = 1;
    output.pixels.resize(w * h);

    // Chỉ thị OpenMP: Tạo vùng song song và chia việc      You, now • Uncommitted changes
    #pragma omp parallel for collapse(2)
    for (int y = 1; y < h - 1; y++) {
        for (int x = 1; x < w - 1; x++) {
            int sum = 0;
            for (int ky = -1; ky <= 1; ky++) {
                for (int kx = -1; kx <= 1; kx++) {
                    sum += input.pixels[(y + ky) * w + (x + kx)] * kernel[ky + 1][kx + 1];
                }
            }
            sum /= divisor;
            if (sum < 0) sum = 0; if (sum > 255) sum = 255;
            output.pixels[y * w + x] = (unsigned char)sum;
        }
    }
}
```

Hình 4.2. Mã code OpenMP cho thuật toán Image Convolution

4.3.5. Đánh giá

Sau khi cài đặt xong hai phiên bản, chương trình thực hiện đo thời gian chạy thực tế (Wall-clock time) bằng thư viện std::chrono.

- Thời gian bao gồm cả hai công đoạn: Làm mịn (Blur) và Làm nét (Sharpen).
- Kết quả đo đạc sẽ được sử dụng để tính Speedup trong Chương 4. Kết quả sơ bộ cho thấy phiên bản song song tận dụng tốt tài nguyên đa nhân, giảm đáng kể thời gian xử lý trên các ảnh kích thước lớn.

CHƯƠNG 5: THỰC NGHIỆM VÀ ĐÁNH GIÁ

5.1. Độ phức tạp của thuật toán

Phần này trình bày và so sánh độ phức tạp tính toán của hai phương pháp sử dụng thuật toán Image Convolution để xử lý ảnh xám y tế được sử dụng trong bài: xử lý tuần tự và xử lý song song hóa bằng OpenMP. Việc đánh giá độ phức tạp giúp phân tích khả năng mở rộng và hiệu quả tính toán của từng phương pháp khi kích thước pixel tăng lên.

5.1.1. Thuật toán Image Convolution tuần tự

Thuật toán xử lý ảnh tuần tự trong chương trình thực hiện phép tích chập (convolution) với kernel kích thước 3×3 trên từng điểm ảnh của ảnh đầu vào. Đối với mỗi pixel, thuật toán thực hiện một số phép toán cố định (9 phép nhân – cộng tương ứng với kernel 3×3). Do đó, thời gian xử lý cho mỗi pixel là hằng số. Với ảnh có kích thước $W \times H$, tổng số pixel cần xử lý là $W \times H$, suy ra độ phức tạp thời gian của thuật toán tuần tự là $O(W \times H)$. Khi chương trình xử lý nhiều ảnh, giả sử có Fảnh đầu vào, độ phức tạp thời gian tổng thể của chương trình tuần tự là $O(F \times W \times H)$. Về mặt bộ nhớ, chương trình cần lưu trữ một số lượng hữu hạn các ma trận ảnh có cùng kích thước với ảnh đầu vào, do đó độ phức tạp bộ nhớ là $O(W \times H)$. Thuật toán có độ phức tạp tuyến tính theo số lượng pixel, tuy nhiên thời gian thực thi tăng đáng kể khi kích thước ảnh lớn, điều này là động lực để áp dụng song song hóa nhằm cải thiện hiệu năng.

5.1.2. Thuật toán Image Convolution song song hóa bằng OpenMP

Để cải thiện thời gian xử lý ảnh, chương trình áp dụng song song hóa bằng OpenMP cho bước lọc ảnh (convolution). Nhận xét rằng việc tính toán giá trị của mỗi pixel (không tính vùng biên) là độc lập với các pixel khác, vì giá trị đầu ra tại (x, y)chỉ phụ thuộc vào các điểm lân cận trong ảnh đầu vào và kernel cố định. Do đó, bài toán có thể được phân rã theo dữ liệu (data parallelism) bằng cách chia tập các pixel cần xử lý cho nhiều luồng (threads) thực hiện đồng thời.

Cụ thể, chương trình sử dụng chỉ thị #pragma omp parallel for để song song hóa các vòng lặp duyệt theo tọa độ ảnh (thường là theo hàng y hoặc kết hợp hai chiều bằng collapse). Mỗi luồng sẽ xử lý một nhóm hàng hoặc nhóm điểm ảnh khác nhau, từ đó giảm tổng thời gian thực thi so với phiên bản tuần tự. Vì mỗi luồng ghi kết quả vào các vị trí khác nhau của ảnh đầu ra và chỉ đọc từ ảnh đầu vào (không ghi chồng chéo), nên thuật toán hạn chế được tranh chấp dữ liệu (data race) và không cần đồng bộ phức tạp ngoài việc các luồng kết thúc trước khi chuyển sang bước tiếp theo.

Về mặt độ phức tạp, song song hóa không làm thay đổi bậc độ phức tạp tính toán: mỗi lần lọc ảnh vẫn có độ phức tạp $O(W \times H)$. Tuy nhiên, xét theo thời gian chạy, trong điều kiện lý tưởng với Pluồng và chi phí quản lý luồng không đáng kể, thời gian thực thi có thể xấp xỉ giảm theo:

$$T_P \approx \frac{T_1}{P}.$$

Trên thực tế, tốc độ tăng (speedup) bị giới hạn bởi các phần không song song hóa được, chi phí tạo/đồng bộ luồng, cũng như giới hạn băng thông bộ nhớ khi nhiều luồng cùng đọc dữ liệu. Do đó, speedup thường nhỏ hơn Pvà phụ thuộc vào kích thước ảnh, số luồng sử dụng và cấu hình phần cứng. Dù vậy, với ảnh có kích thước lớn, lượng tính toán đủ nhiều giúp chi phí overhead trở nên tương đối nhỏ, nên OpenMP mang lại cải thiện hiệu năng rõ rệt so với xử lý tuần tự.

5.2. Cách đo thời gian

Để đánh giá hiệu năng của hai phương xử lý ảnh xám bằng thuật toán Image Convolution, thời gian thực thi của chương trình được đo bằng hai cách khác nhau tùy theo phiên bản cài đặt. Đối với phiên bản tuần tự, chương trình sử dụng thư viện chrono của C++ để ghi lại thời điểm bắt đầu và kết thúc quá trình xử lý ảnh. Cụ thể, trước khi gọi hàm processSequential(img, imgBlur, KERNEL_BLUR, 16) và hàm processSequential(imgBlur, imgFinal, KERNEL_SHARPEN, 1),thời điểm bắt đầu được lấy bằng lệnh auto start = high_resolution_clock::now(), và sau khi xử lý ảnh hoàn tất, thời điểm kết thúc được ghi lại bằng auto end = high_resolution _clock::now(). Thời gian thực thi được tính bằng hiệu giữa hai mốc thời gian này. Đối với phiên bản song song bằng OpenMP, việc đo thời gian được thực hiện tượng tự như xử lý tuần tự, được đo trước và sau khi gọi hàm processOpenMP (imgIn, imgBlur, KERNEL_BLUR, 16) và processOpenMP(imgBlur, imgFinal, KERNEL_SHARPEN, 1). Để đảm bảo kết quả đo có độ chính xác cao và giảm ảnh hưởng của các yếu tố ngẫu nhiên như tải hệ thống hoặc độ trễ truyền thông, mỗi bài kiểm thử được chạy ba lần và lấy giá trị trung bình cộng của ba lần đo làm kết quả cuối cùng. Cách làm này giúp đánh giá khách quan hơn hiệu năng của từng phương pháp, đồng thời cho phép so sánh trực tiếp giữa xử lý ảnh tuần tự và xử lý ảnh song song bằng OpenMP

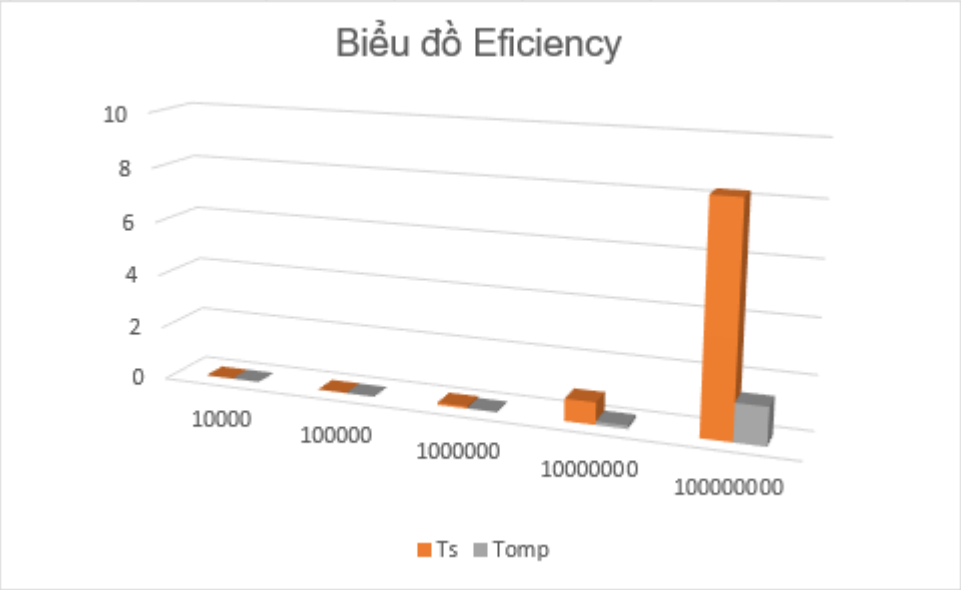
5.3. Kết quả thực nghiệm

Bảng 4. 1 Kết quả thực nghiệm

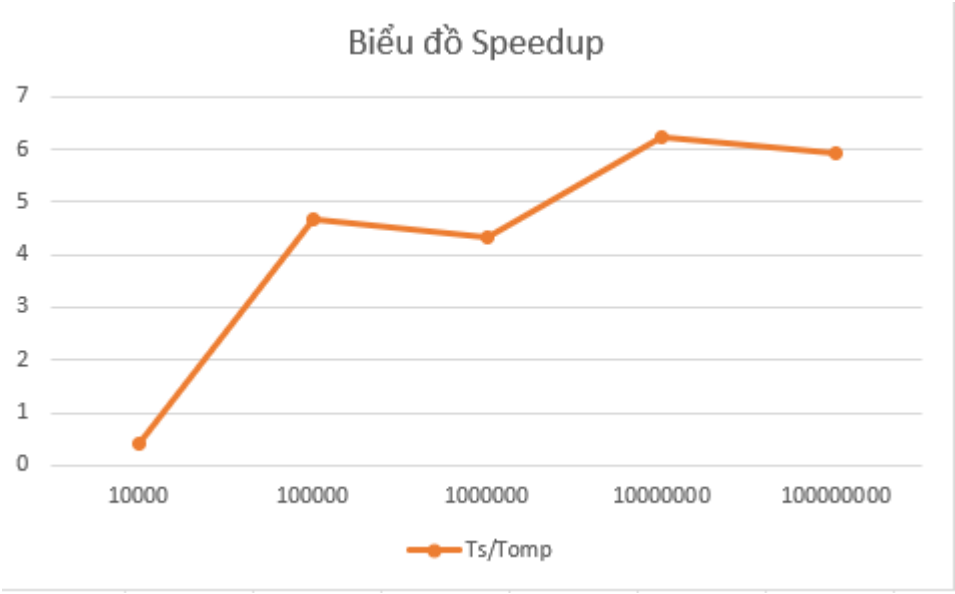
N	Ts	Tomp	Ts/Tomp
10000	0.0012247	0.0029925	0.409256
100000	0.0126316	0.0027108	4.65973
1000000	0.0807984	0.0186202	4.33929
10000000	0.815536	0.131151	6.21829
100000000	8.26309	1.3905	5.94254

Chú thích:

- N: kích thước ảnh H x W pixel
- Ts: thời gian thực hiện tuần tự
- Tomp: thời gian thực hiện OpenMP



Hình 5.1. Biểu đồ Efficiency



Hình 5.2. Biểu đồ Speedup

5.4. Đánh giá và nhận xét.

Kết quả thực nghiệm cho thấy thời gian thực thi của hai phương pháp (tuần tự, OpenMP) có sự khác biệt rõ rệt khi kích thước ảnh tăng lên

Đối với các ảnh có kích thước khỏ (N = 10.000 pixel), thời gian thực thi của hai phương pháp không chênh lệch đáng kể. Nguyên nhân là do OpenMp cần phải overhead tạo luồng và đồng bộ cần thời gian tương đối lớn so với khối lượng tính toán, khiến phương pháp song song chưa phát huy được hiệu quả

Khi kích thước ảnh tăng lên (từ N = 100.000 trở lên) hiệu quả của phương pháp song song được thể hiện rõ rệt. Thời gian chạy của chương trình sử dụng OpenMP giảm mạnh so với phương pháp tuần tự, đặc biệt với hình ảnh có kích thước lớn (N = 10.000.000 pixel) thuật toán sử dụng OpenMp cho kết quả nhanh hơn đáng kể so với phương pháp tuần tự.

Giá trị Speedup (Ts/Tomp) tăng dần theo kích thước ảnh, đạt mức cao nhất khoảng 6 lần ở N = 10.000.000 pixel, cho thấy hiệu năng tăng đáng kể khi số pixel cần xử đủ lớn để tận dụng hiệu quả song song hóa bằng OpenMP. Tuy nhiên, ở kích thước ảnh khoảng N = 10000 pixel, tốc độ xử lý do thời gian OpenMP cần overhead tạo luồng và đồng bộ chiếm tỷ trọng lớn hơn so với thời gian tính toán.

Biểu đồ Efficiency có thể cho ta nhận thấy rằng hiệu suất song song của chương trình xử lý ảnh xám bằng OpenMP phụ thuộc mạnh vào kích thước dữ liệu đầu vào. Ở các

kích thước ảnh nhỏ (N = 10.000 và 100.000 pixel), efficiency rất thấp, thậm chí gần bằng 0. Điều này cho thấy tài nguyên tính toán đa lõi chưa được khai thác hiệu quả, do chi phí tạo luồng, đồng bộ và quản lý OpenMP lớn hơn nhiều so với khối lượng tính toán của phép convolution trên ảnh nhỏ. Khi kích thước ảnh tăng lên (từ N = 1.000.000 trở đi), efficiency bắt đầu cải thiện rõ rệt, phản ánh việc thời gian tính toán tăng đủ lớn để bù lại chi phí song song hóa. Đặc biệt, tại kích thước ảnh rất lớn (N = 10.000.000 và 100.000.000 pixel), efficiency đạt mức cao nhất, cho thấy các luồng được khai thác tốt và phần lớn thời gian thực thi dành cho tính toán thay vì overhead. Tuy nhiên, efficiency không tăng tuyến tính theo kích thước mà có xu hướng chững lại, nguyên nhân là do chương trình dần bị giới hạn bởi băng thông bộ nhớ và hiệu quả bộ nhớ đệm khi thực hiện nhiều phép truy cập lân cận trong convolution

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Tổng kết lại quá trình nghiên cứu, đề tài đã hoàn thành việc xây dựng và tối ưu hóa thuật toán tích chập xử lý ảnh y tế trên nền tảng tính toán song song sử dụng thư viện OpenMP. Thông qua các kịch bản kiểm thử đa dạng, nhóm thực hiện đã minh chứng được tính đúng đắn của chiến lược phân rã miền dữ liệu và cơ chế lập lịch tĩnh đã đề xuất. Kết quả thực nghiệm cho thấy một xu hướng rõ rệt: giải thuật song song phát huy hiệu quả vượt trội khi xử lý các tập dữ liệu ảnh có độ phân giải cao, giúp rút ngắn đáng kể thời gian thực thi so với phương pháp tuần tự truyền thống. Điều này khẳng định OpenMP là giải pháp phù hợp cho các bài toán xử lý ảnh y tế đòi hỏi khối lượng tính toán lớn.

Tuy nhiên, nghiên cứu cũng chỉ ra rằng kỹ thuật song song hóa không phải lúc nào cũng mang lại hiệu năng dương. Đối với các ảnh có kích thước nhỏ, chi phí khởi tạo luồng và quản lý vùng song song trở nên đáng kể so với thời gian tính toán thực tế, dẫn đến việc xử lý tuần tự đôi khi lại chiếm ưu thế. Điều này cho thấy tầm quan trọng của việc xác định "ngưỡng" dữ liệu phù hợp để quyết định khi nào nên kích hoạt chế độ song song nhằm đạt được sự tối ưu về tài nguyên hệ thống. Bên cạnh đó, do giới hạn về phần cứng thử nghiệm trên máy đơn, khả năng mở rộng của thuật toán trên các hệ thống quy mô lớn vẫn chưa được đánh giá toàn diện.

Dựa trên những phân tích đó, hướng phát triển tiếp theo của đề tài sẽ tập trung vào việc khắc phục các hạn chế về phần cứng và kiến trúc. Cụ thể, nhóm đề xuất mở rộng nghiên cứu sang các mô hình lập trình lai kết hợp giữa MPI và OpenMP để tận dụng sức mạnh của các hệ thống máy tính cụm, hoặc chuyển đổi thuật toán sang nền tảng GPU với CUDA để khai thác khả năng xử lý song song hàng nghìn luồng của card đồ họa. Ngoài ra, việc tận dụng các chỉ thị vector hóa phần cứng (như AVX) để xử lý đồng thời nhiều điểm ảnh trong một chu kỳ máy cũng là một hướng đi tiềm năng nhằm nâng cao hơn nữa tốc độ xử lý cho các ứng dụng y tế thời gian thực.