ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Parallel Computing

Báo cáo thực hành

Heat Diffusion Simulation

GVHD: Diệp Thanh Đăng
La Quốc Nhựt Huân

Sinh viên: Hà Chí Trung - 2213682

TP HỒ CHÍ MINH, THÁNG 12 2025

1 Mở đầu

1.1 Dẫn nhập

Vào ngày 3 tháng 9 năm 2025, Trung Quốc đã tổ chức một cuộc diễu hành Ngày Chiến thắng đánh dấu kỷ niệm 80 năm chiến thắng Nhật Bản và kết thúc Thế chiến II. Đáng chú ý, Trung Quốc lần đầu tiên công bố tên lửa đạn đạo xuyên lục địa (ICBM) DF-61 - được mô tả là một trong những hệ thống tiên tiến nhất của họ - với tầm bắn ước tính từ 12.000 đến 15.000 km. Có tin đồn cho rằng DF-61 được trang bị bảy đầu đạn hạt nhân, mỗi đầu đạn có sức công phá ước tính khoảng 650 kiloton - mạnh hơn gấp 300 lần quả bom "Little Boy" thả xuống Hiroshima.

1.2 Yêu cầu

Trong bài tập này, sinh viên được yêu cầu ước tính quá trình khuếch tán nhiệt (heat diffusion) được tạo ra từ một vụ nổ hạt nhân. Quá trình khuếch tán nhiệt không gì khác hơn chính là một phép tích chập 2D (2D convolution) với một kernel K:

$$K = \begin{pmatrix} 0.05 & 0.1 & 0.05 \\ 0.1 & 0.4 & 0.1 \\ 0.05 & 0.1 & 0.05 \end{pmatrix}$$

để lan truyền nhiệt sang các ô lân cận.

Sinh viên được cung cấp một ma trận 4000×4000, đại diện cho cường độ nhiệt của một thành phố (khoảng 1600 km^2), trong đó mỗi ô tương ứng với diện tích 100 m^2 (chiều dài 10m và chiều rộng 10m). Ma trận này mô phỏng hậu quả của một thảm họa hạt nhân, với nhiệt độ đỉnh ở trung tâm và giảm dần về phía bề mặt, trong khi các khu vực xung quanh duy trì nhiệt độ cơ bản là $30°C$ (sử dụng 30 cho padding thay vì 0).

Cụ thể, sinh viên được yêu cầu:

1. Thực hiện tích chập tuần tự (sequential convolution) trong ít nhất 100 lần lặp.
2. Triển khai ít nhất hai thuật toán tích chập song song (parallel convolution) khác nhau bằng OpenMP, mỗi thuật toán chạy ít nhất 100 lần lặp.
3. Tiến hành so sánh thời gian (time comparison) giữa các triển khai tuần tự và song song.

1.3 Mục tiêu Bài tập

Dựa trên các yêu cầu kỹ thuật đã nêu, mục tiêu của bài tập này là áp dụng thư viện OpenMP để song song hóa và tối ưu hóa hiệu suất của thuật toán tích chập 2D lặp lại.

Báo cáo này sẽ tập trung vào các mục tiêu cụ thể sau:

- **Trình bày triển khai:** Mô tả chi tiết ba phương pháp triển khai thuật toán: (a) Phiên bản Tuần tự (Sequential), (b) Phiên bản Song song OpenMP với Static Schedule, và (c) Phiên bản Song song OpenMP với Dynamic Schedule.
- **Thu thập dữ liệu:** Ghi lại thời gian thực thi (Execution Time) của cả ba phiên bản trong cùng một điều kiện môi trường thử nghiệm.
- **Phân tích hiệu suất:** Tiến hành so sánh và phân tích dữ liệu đã thu thập. Các chỉ số hiệu suất chính, bao gồm **Tăng tốc (Speedup)** và **Hiệu quả (Efficiency)**, sẽ được tính toán để đánh giá mức độ hiệu quả của việc song song hóa và xác định chiến lược lập lịch (scheduling) tối ưu nhất cho bài toán này.

2 Hiện thực

2.1 Giải thuật tuần tự

Luồng thực thi của thuật toán mô phỏng truyền nhiệt được thiết kế để lặp lại chính xác quá trình tích chập 3×3 và cập nhật trạng thái dữ liệu (Ma trận `GRID`) qua mỗi bước thời gian.

1. **Khởi tạo Trạng thái Tạm thời:** Đầu tiên, tạo một ma trận đệm mới (`new_grid`) cùng kích thước với ma trận gốc (`grid`) để lưu kết quả của mỗi bước tính toán.
2. **Tạo Vòng lặp Thời gian:**

```
        for (int s = 0; s < steps; ++s) {
```

   Tạo một vòng lặp lớn (`for s`) chạy từ 1 đến N (ví dụ: $N = 100$), điều khiển số lần mô phỏng truyền nhiệt. Toàn bộ quá trình tính toán và cập nhật trạng thái sẽ diễn ra bên trong vòng lặp này.

3. **Duyệt Vị trí và Tích chập:**

```
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
```

   - Sử dụng hai vòng lặp lồng nhau (`for i`, `for j`) để duyệt tới tất cả các ô (i, j) của ma trận gốc.
   - Tại mỗi ô, thực hiện tính tích chập bằng hai vòng lặp nhỏ hơn (`for ki`, `for kj`).

4. **Phép tính tích chập, kiểm tra biên:**

```
        for (int ki = -1; ki <= 1; ++ki) {
                    for (int kj = -1; kj <= 1; ++kj) {
                        int ni = i + ki;
                        int nj = j + kj;
                        double current_temp;

                        if (ni < 0 || ni >= rows || nj < 0 || nj >= cols) {
                            current_temp = PADDING_VALUE;
                        } else {
                            current_temp = grid[ni][nj];
                        }
                        new_val += current_temp * K[ki + 1][kj + 1];
                    }
                }
                new_grid[i][j] = new_val;
```

   - Trong quá trình tích chập, **kiểm tra biên** được thực hiện: Nếu tọa độ lân cận (ni, nj) nằm ngoài ma trận, nhiệt độ lân cận sẽ là 30.0 (yêu cầu padding). Ngược lại, lấy giá trị từ ma trận gốc.
   - Giá trị nhiệt độ mới được tính bằng tổng của các phép nhân (nhiệt độ $\times$ trọng số kernel).
   - Kết quả được ghi vào vị trí tương ứng trong `new_grid`.

5. **Cập nhật và Lặp lại:**

```
1                    grid.swap(new_grid);
```

Sau khi tính xong toàn bộ ma trận (vòng lặp i, j kết thúc), ma trận gốc được cập nhật bằng kết quả mới:

$$\text{Ma trận gốc} = \text{Ma trận đệm}$$

Thao tác này được thực hiện bằng lệnh `grid.swap(new_grid)`, làm cho kết quả vừa tính toán trở thành trạng thái nhiệt độ ban đầu cho lần lặp tiếp theo. Quá trình tiếp tục lặp lại cho đến khi đạt N bước.

2.2 Song song hóa với OpenMP và Static Scheduling

Cơ chế song song hóa được sử dụng là **Phân chia Dữ liệu theo Hàng (Row-wise Data Decomposition)** kết hợp với **Lập lịch Tĩnh (Static Scheduling)**. Chỉ thị `#pragma omp parallel for schedule(static)` được áp dụng trên vòng lặp duyệt hàng (`for i`) để phân chia công việc:

- **Đơn vị Công việc:** Mỗi hàng (`i`) của ma trận 4000×4000 được coi là một đơn vị công việc độc lập vì giá trị nhiệt mới của một hàng không phụ thuộc vào kết quả tính toán của các hàng khác trong cùng một bước thời gian.
- **Cơ chế Phân chia:** Lập lịch Tĩnh chia tổng số hàng (4000) thành N khối có kích thước bằng nhau (ví dụ: 500 hàng cho mỗi luồng nếu $N = 8$). Sự phân công này là **cố định** và được thực hiện một lần duy nhất khi vòng lặp bắt đầu.
- **Luồng Thực thi (Fork-Join):** Quá trình thực thi diễn ra theo mô hình Fork-Join (Tách-Hợp):
  1. **Tách (Fork):** Khi gặp chỉ thị `parallel for`, luồng chính tạo ra N luồng làm việc.
  2. **Thực thi:** N luồng đồng thời chạy trên các lõi CPU, mỗi luồng chỉ xử lý các hàng đã được gán (ví dụ: Luồng 0 xử lý hàng 0 đến 499).
  3. **Hợp (Join):** Sau khi tất cả các luồng hoàn thành, chúng đồng bộ hóa (tại một rào cản ngầm) và luồng thực thi hợp nhất trở lại thành luồng chính để thực hiện thao tác cập nhật trạng thái (`grid.swap`).

Lợi ích của Lập lịch Tĩnh (Static)

Chế độ `schedule(static)` tối ưu cho bài toán này vì:

1. **Tính Đồng nhất (Uniformity):** Công việc tính tích chập cho mọi hàng là **đồng đều** (mất cùng một lượng thời gian).
2. **Overhead Thấp:** Lập lịch tĩnh loại bỏ nhu cầu giao tiếp giữa các luồng để yêu cầu công việc mới, giúp giảm thiểu **chi phí quản lý luồng (overhead)** xuống mức thấp nhất.

Đây là cơ chế được dự đoán sẽ mang lại mức tăng tốc (speedup) tốt nhất cho bài toán tích chập.

2.3 Song song hóa với `schedule(dynamic)`

Cơ chế này được áp dụng tương tự trên vòng lặp duyệt hàng (`for i`) bằng chỉ thị: `#pragma omp parallel for schedule(dynamic)`.

- **Đơn vị Công việc:** Tương tự như chế độ Static, công việc được chia thành các hàng (`i`), nhưng được gộp thành các khối (chunks) nhỏ hơn.
- **Cơ chế Phân chia:** Phân chia công việc là **linh hoạt** và diễn ra trong thời gian chạy (runtime):
  1. Ban đầu, mỗi luồng chỉ được gán một khối công việc nhỏ.
  2. Khi một luồng hoàn thành khối công việc hiện tại của mình, nó sẽ **yêu cầu** khối công việc tiếp theo từ một hàng đợi chung.
  3. Quá trình này tiếp diễn cho đến khi tất cả các lần lặp được xử lý.

Dự đoán iệu suất

Mặc dù chế độ Dynamic rất hiệu quả cho các bài toán mà khối lượng công việc của mỗi lần lặp là **không đồng nhất** (ví dụ: một số hàng tốn nhiều thời gian hơn hàng khác), nhưng đối với bài toán Tích chập 2D:

1. **Tải Đồng nhất:** Vì thời gian tính toán cho mỗi hàng trong tích chập là gần như nhau, chế độ Dynamic không mang lại lợi ích cân bằng tải.
2. **Chi phí (Overhead) Cao hơn:** Việc phải liên tục giao tiếp với hệ thống để yêu cầu các khối công việc mới (runtime management) sẽ làm tăng đáng kể **chi phí lập lịch (scheduling overhead)**.

Do đó, chế độ Dynamic được dự đoán là sẽ có thời gian thực thi **lâu hơn** so với chế độ Static cho bài tập này. Việc so sánh T_{dynamic} với T_{static} sẽ là một điểm mấu chốt trong phần phân tích hiệu suất của báo cáo.

3 Kết quả

Cấu hình CPU chạy chương trình:

```
Architecture:                    x86_64
CPU(s):                          12
  Model name:                    11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz
    CPU max MHz:                 4500.0000
    CPU min MHz:                 800.0000
```

Kết quả mô phỏng dạng bản đồ nhiệt tăng dần số lần lặp:

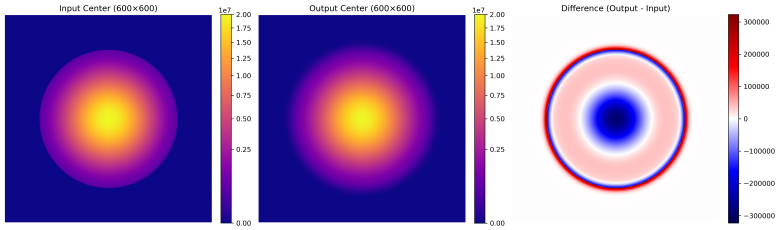

Figure 1: 200 iterations

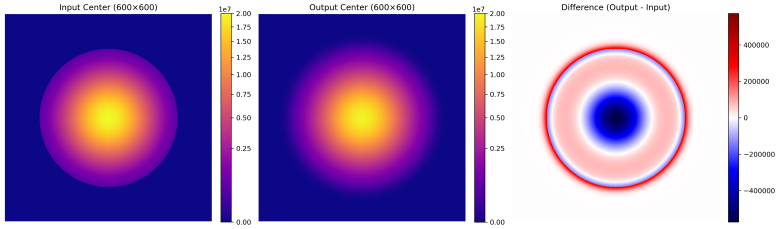

Figure 2: 400 iterations

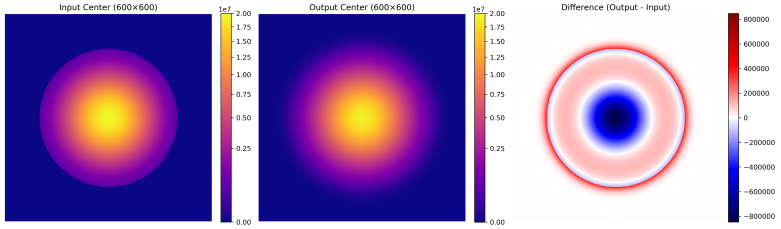

Figure 3: 600 iterations

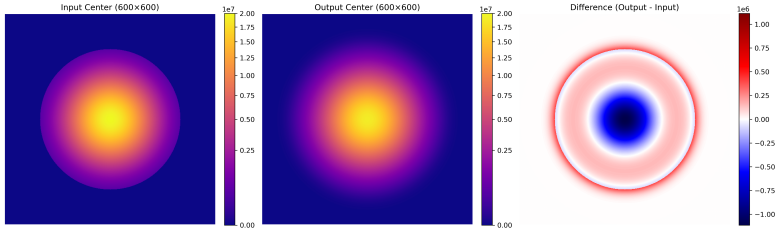

Figure 4: 800 iterations

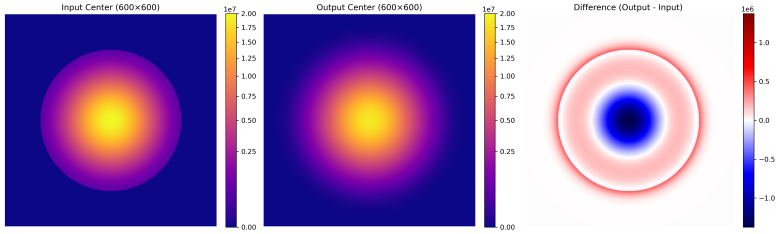

Figure 5: 1000 iterations

Iterations	Sequential	Parallel Static	Parallel Dynamic
100	13,45	3,01	2,64
200	26,98	4,92	5,60
300	39,75	7,79	8,11
400	54,50	11,03	10,57
500	67,17	13,24	13,03
600	80,63	16,45	16,41
700	95,53	19,36	19,25
800	109,97	21,39	21,87
900	128,90	25,11	25,96
1000	141,06	26,91	29,08

Figure 6: Bảng kết quả thời gian thực thi theo lần lặp

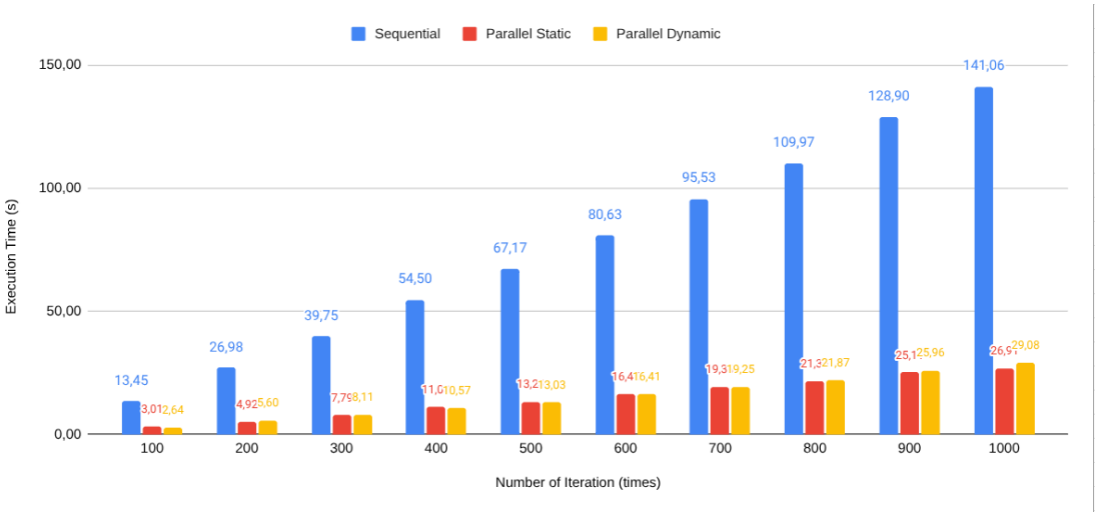

Figure 7: Biểu đồ thời gian thực thi theo lần lặp

3.1 Phân tích Tăng tốc (Speedup) và Hiệu quả (Efficiency)

Sử dụng khối lượng công việc lớn nhất (1000 Iterations) và $N = \mathbf{12}$ luồng để tính toán các chỉ số hiệu suất chính. Công thức tính Tăng tốc là $S = T_{seq}/T_{par}$ và Hiệu quả là $E = S/N$.

Table 1: Tăng tốc và Hiệu quả tại 1000 Iterations ($N = 12$ Threads).

Phiên bản	**Thời gian** (T)	**Tăng tốc** (S)	**Hiệu quả** (E)
Sequential	141.06 s	1.00	8.33%
Parallel Static	26.91 s	$141.06/26.91 = \mathbf{5.24}$	$5.24/12 = \mathbf{43.67}\%$
Parallel Dynamic	29.08 s	$141.06/29.08 = \mathbf{4.85}$	$4.85/12 = 40.42\%$

3.2 Kết luận

1. **Hiệu quả Song song:** Cả hai phiên bản OpenMP đều giảm đáng kể thời gian thực thi xuống khoảng 1/5 thời gian tuần tự. Điều này xác nhận rằng tích chập 2D là một bài toán **data parallelism** lý tưởng.

2. **Xu hướng tại Khối lượng Công việc Lớn:** Ban đầu (100-500 Iterations), Dynamic Schedule có lợi thế nhỏ. Tuy nhiên, khi khối lượng công việc tăng lên 1000 Iterations, **Static Schedule** trở nên vượt trội (26.91s so với 29.08s của Dynamic).

3. **Phân tích Schedule Tĩnh (`static`):** Chế độ Static đạt mức Tăng tốc cao nhất (5.24 lần). Tuy nhiên, với 12 luồng, Hiệu quả (E) đạt **43.67**%, thấp hơn đáng kể so với mức lý tưởng. Điều này cho thấy có thể **12** luồng là quá nhiều (vượt quá số core vật lý hiệu quả) hoặc Overhead (chi phí) OpenMP đã tăng cao khi sử dụng số lượng luồng lớn hơn. Dù vậy, Static vẫn là chiến lược tốt nhất cho bài toán đồng nhất.

4. **Phân tích Schedule Động (`dynamic`):** Mặc dù vẫn đạt tăng tốc đáng kể, chế độ Dynamic bị ảnh hưởng bởi **chi phí lập lịch runtime**. Việc phải liên tục yêu cầu các khối công việc mới tích lũy qua 1000 bước lặp, khiến nó chậm hơn Static. Điều này cho thấy Dynamic là không tối ưu cho các bài toán đồng nhất.