TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Lập trình song song thuật toán Merge Sort trên CPU và GPU

**Nhóm 7:**
Trần Hoài Nam - 20224412
Nguyễn Hoàng Tân - 20224426
Nguyễn Đăng Khánh - 20224440
Lương Hữu Phúc - 20224452

# NỘI DUNG

1. Đặt vấn đề
2. Khái niệm về Merge Sort
3. Song song hóa trên CPU bằng OpenMP
4. Song song hoá trên GPU bằng Cuda
5. Kết quả và đánh giá

# TẠI SAO LẠI CẦN SONG SONG HÓA

**Merge Sort** vốn là thuật toán đệ quy Merge Sort chia nhỏ bài toán → gọi đệ quy trái và phải → mỗi nhánh có thể xử lý độc lập.

Phù hợp với **chia để trị** (Divide & Conquer) Do chia thành 2 phần liên tục → có thể chia cho nhiều luồng xử lý cùng lúc.

**Tăng tốc độ** sắp xếp Với mảng lớn (triệu phần tử), nếu tuần tự sẽ rất lâu → song song giúp giảm đáng kể thời gian.

Tận dụng CPU đa lõi hiệu quả Mỗi phần của Merge Sort có thể chạy trên 1 core → **tận dụng toàn bộ** tài nguyên hệ thống.

**Tối ưu hiệu suất** xử lý dữ liệu lớn Merge Sort thường dùng cho dữ liệu lớn → song song hóa giúp xử lý nhanh hơn rất nhiều. chuyển bảng này thành ảnh đi

# PHÂN TÍCH TÍNH KHẢ THI

- **Bản chất thuật toán:**

Chia mảng thành 2 nhánh độc lập, dễ song song. **Khả thi cao**

- **Phân chia công việc:**

Nhánh đệ quy song song tốt, merge khó song song.**Khả thi một phần**

- **Hiệu quả thời gian:**

Nhanh hơn với dữ liệu lớn (triệu phần tử). **Khả thi**

- **Phần cứng:**

Cần CPU đa lõi, phổ biến hiện nay. **Khả thi**

- **Overhead:**

Dữ liệu nhỏ có thể chậm hơn do tạo luồng. **Hạn chế**

# Merge Sort là gì ?

**Sử dụng phương pháp chia để trị (divide to conquer):**

- Chia mảng thành 2 nửa nhỏ
- Sắp xếp đệ quy từng nửa
- Sau đó hợp nhất kết quả

Độ phức tạp *O(nlogn)* đảm bảo hiệu suất ổn định ngay cả với mảng lớn

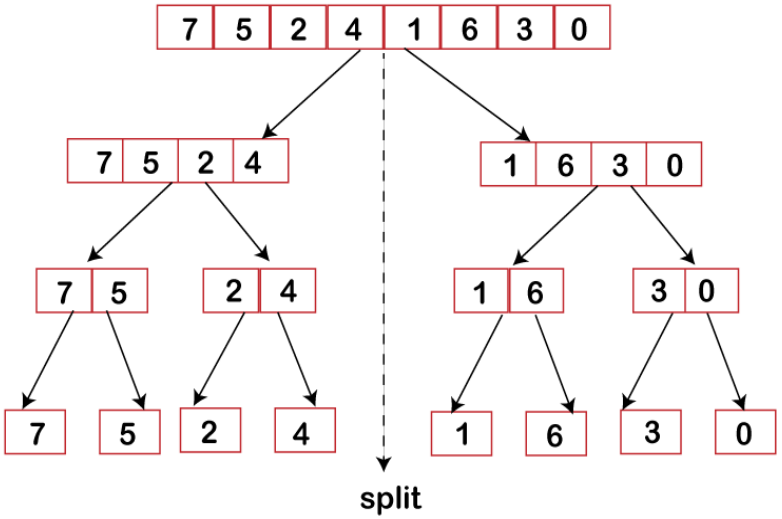
💡 **Nhờ tính chất chia đều khối lượng công việc thực thi nên có thể áp dụng được kỹ thuật song song**

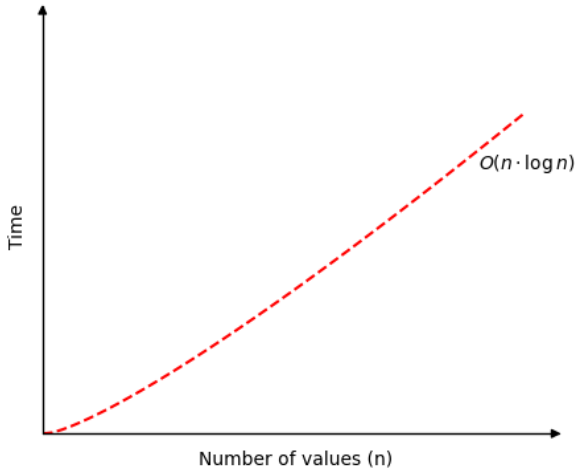
**Time Complexity**

O(n log n)

**Memory Complexity**

O(n)



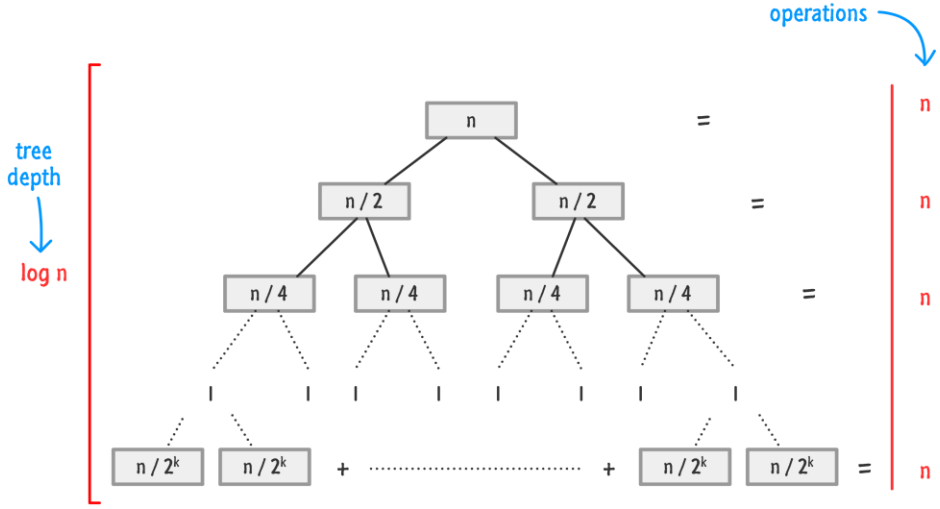






# SONG SONG HÓA SỬ DỤNG OPENMP (CPU)

# Song song hóa trên CPU bằng OpenMP

## Thực thi tuần tự:




***Khi số phần tử tăng, thời gian xử tăng theo cấp số nhân (gần tuyến tính với n, nhưng độ sâu đệ quy lại tăng $log_2(n)$)***

✗ Tuy nhiên chỉ chạy 1 CPU core

⇒ Sử dụng **OpenMP** sẽ tận dụng được sức mạnh của CPU đa nhân

⇒ **Phù hợp để tối ưu bằng song song hóa:**

- Song song trên các hàm đệ quy của thuật toán
- Thực hiện với mảng động n phần tử, các phần tử được sinh ngẫu nhiên
- Khi số phần tử < 10000, nên thực hiện merge sort tuần tự để tránh overhead, tạo ra thread không đáng có, gây chậm chương trình

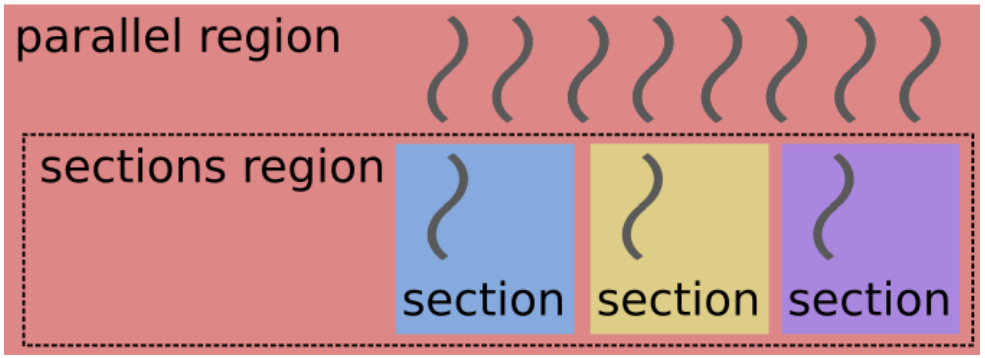
# Song song hóa bằng OpenMP Sections

**Ý tưởng:** Phân chia nhánh trái, nhánh phải thành các sections riêng biệt, sau đó merge lại khi các sections hoàn tất

**TH1**: Sử dụng chỉ thị: #pragma omp parallel sections

```
#pragma omp parallel sections //T
{
  #pragma omp section //Thread 1
  merge_sort(a, left, mid); //De

  #pragma omp section //Thread 2
  merge_sort(a, mid + 1, right);
}
```




- Đơn giản, dễ triển khai, phù hợp cho các bài toán chia nhánh rõ ràng
- Có thể tận dụng CPU đa nhân

- Hạn chế hiệu suất với số lượng phần tử lớn -> Đệ quy sâu
- Mỗi lần gọi đệ quy lại sinh thêm thread -> thread lồng thread
- Tiêu tốn bộ nhớ, không tận dụng tối đa CPU

# Song song hóa bằng OpenMP Task

**TH2**: Sử dụng chỉ thị: #pragma omp task

```
#pragma omp parallel
{
  #pragma omp single //Dam bao chi
  merge_sort(a, 0, num - 1);
}
```

*Hàm main*

- **#pragma omp parallel** + **#pragma omp single:** Tạo pool thread, đảm bảo chỉ 1 thread đầu tiên gọi merge_sort(), thread khác chờ để nhận task

```
#pragma omp task shared(a) firstprivate(left, mid)
merge_sort(a, left, mid);

#pragma omp task shared(a) firstprivate(right, mid)
merge_sort(a, mid + 1, right);

#pragma omp taskwait //Cho ca 2 task xong roi moi merge
```

*Hàm chứa thành phần đệ quy*

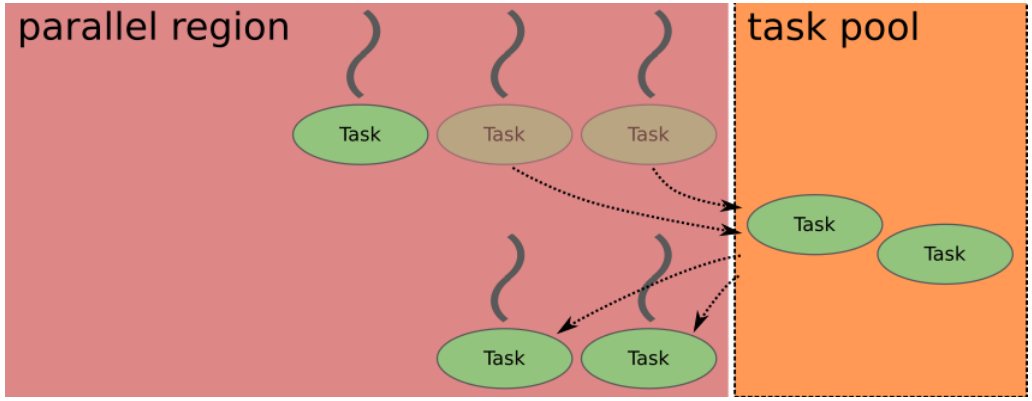
- **#pragma omp task:** tạo 1 nhiệm vụ con cho thread khác xử lý khi có sẵn tài nguyên
- **shared(a):** tất cả các task dùng chung mảng a
- **firstprivate(left, min):** sao chép giá trị left, mid vào biến cục bộ riêng của task
- **#pragma omp taskwait**: chờ cả 2 task hoàn thành rồi merge lại

# Song song hóa bằng OpenMP Task

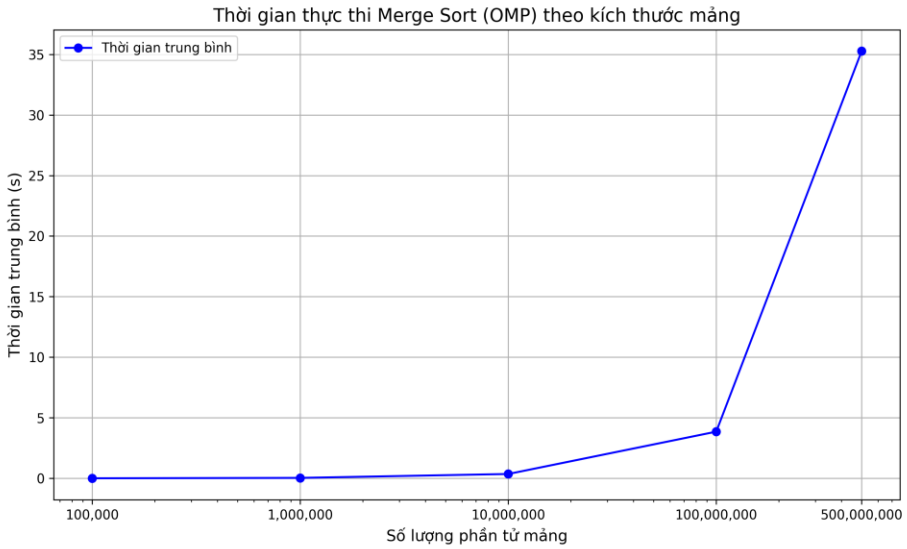
**TH2**: Sử dụng chỉ thị: #pragma omp task

**Quy trình trên máy tính**

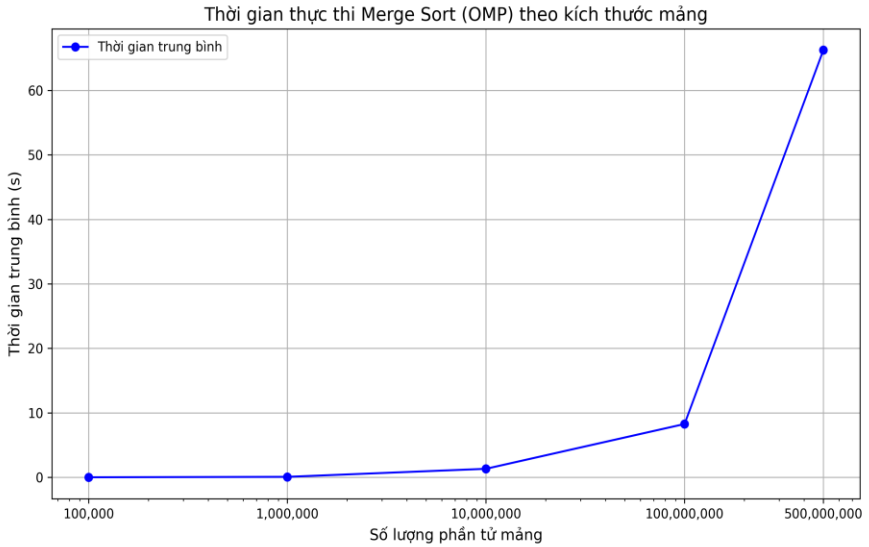
| CPU thread | Task |
|---|---|
| Thread 0 | Khởi động merge_sort và tạo task bằng #pragma omp task |
| Thread 1 | Lấy task từ hàng đợi (nếu có) và chạy merge_sort() |
| Thread 2 | Lấy task khác và tiếp tục đệ quy |
| Thread n | Tiếp tục chia nhỏ công việc |



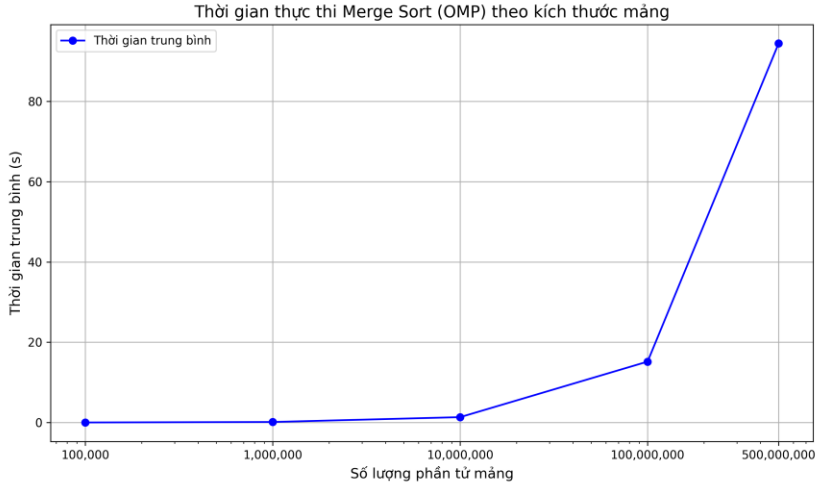
# Đánh giá kết quả




**Average_time_omp_task**

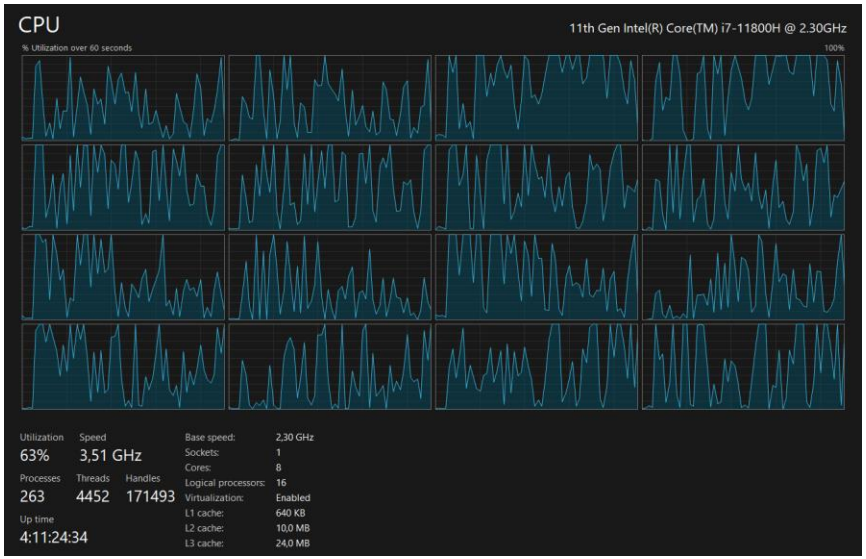



**Average_time_sections**

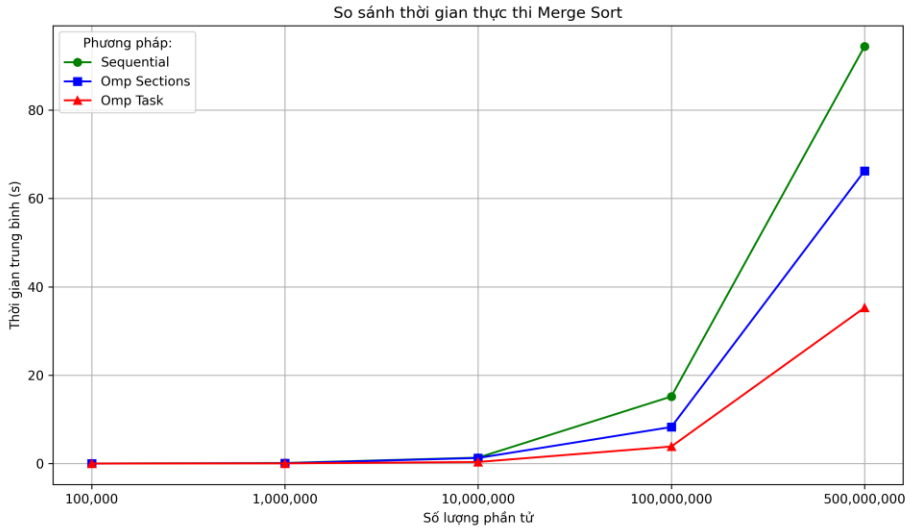



**Average_time_sequential**

# Đánh giá kết quả




*CPU khi xử lý song song*




*So sánh thời gian thực thi*

- Đối với mảng từ 100k -> 10 triệu phần tử: sự khác biệt không quá rõ
- Khi mảng từ 100 triệu trở lên:
  - **Omp task**: cho tốc độ xử lý nhanh nhất nhờ khả năng chia nhỏ đệ quy một các linh hoạt
  - **Omp sections**: Cho hiệu quả cao hơn tuần tự nhưng bị giới hạn bởi số section đặt ra
  - **Sequential**: Mất nhiều thời gian nhất do xử lý tuần tự trên 1 thread duy nhất

⇒ ***Sử dụng omp task cho hiệu quả cao nhất do CPU được chia nhiều trên nhiều cores, thể hiện qua biểu đồ sử dụng tài nguyên***

# Tính toán kết quả (Phân tích hiệu suất song song)

**Tăng tốc (Speedup - S)**: Tỷ lệ giữa thời gian thực thi tuần tự và thời gian thực thi song song.

$$S = \frac{T_{\text{tuần tự}}}{T_{\text{song song}}}$$

| Kích thước đầu vào | Merge_sort_sequential | omp task | omp sections |
|---|---|---|---|
| 100K | 0.0123999914 | 0.0033333460 | 0.0081333160 |
| 1M | 0.1331333478 | 0.0344666481 | 0.0733333270 |
| 10M | 1.3413999716 | 0.3562666893 | 1.2609333515 |
| 100M | 15.1702000300 | 3.8511333307 | 8.2645333290 |
| 500M | 94.4135333538 | 35.2743333181 | 66.2394666513 |

**Kết quả sau khi tính toán Speedup:**

| Kích thước đầu vào | S_omp task | S_omp_sections |
|---|---|---|
| 100K | 3.72 | 1.52 |
| 1M | 3.86 | 1.82 |
| 10M | 3.76 | 1.06 |
| 100M | 3.94 | 1.84 |
| 500M | 2.68 | 1.43 |

# Tính toán kết quả

**Hiệu suất** ()

Hiệu suất được tính bằng tăng tốc chia cho số luồng:

$$E = \frac{S}{P}$$

| Kích thước đầu vào | E_omp task | E_omp sections |
|---|---|---|
| 100K | 0.47 | 0.19 |
| 1M | 0.48 | 0.23 |
| 10M | 0.47 | 0.13 |
| 100M | 0.49 | 0.23 |
| 500M | 0.34 | 0.18 |

# Tính toán kết quả

**Khả năng mở rộng (Scalability)**
**Xu hướng tăng tốc**:

- omp task: Tăng tốc đạt đỉnh ở 100M phần tử (3.94) nhưng giảm xuống 2.68 ở 500M, cho thấy chi phí song song hoặc tranh chấp tài nguyên (như băng thông bộ nhớ, tranh chấp cache) tăng lên với đầu vào lớn.
- omp sections: Tăng tốc thấp hơn, đạt đỉnh ở 1M và 100M (1.82–1.84), nhưng giảm mạnh xuống 1.06 ở 10M và 1.43 ở 500M, cho thấy phân phối công việc không hiệu quả.

**Xu hướng hiệu suất**:

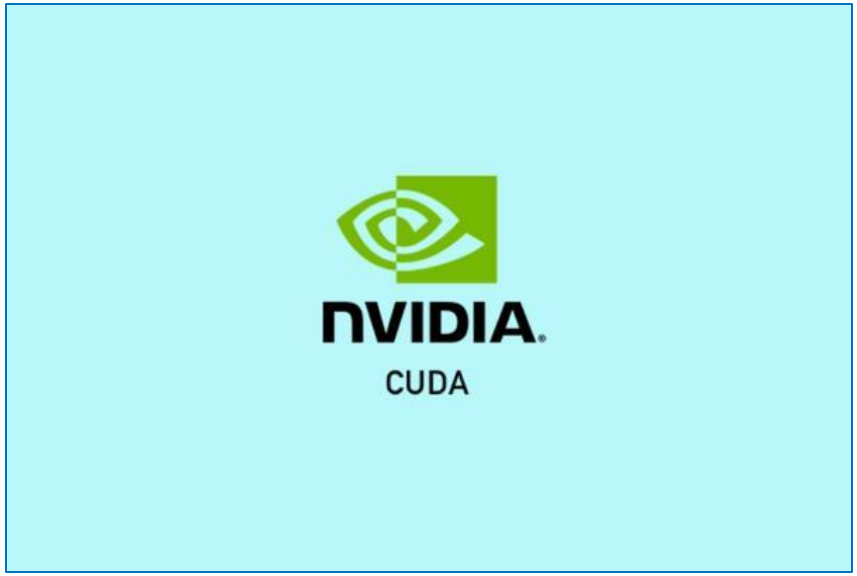
- omp task: Hiệu suất ổn định (~0.47–0.49) đến 100M, nhưng giảm xuống 0.34 ở 500M, phản ánh hạn chế về khả năng mở rộng.
- omp sections: Hiệu suất rất thấp (0.13–0.23), do sử dụng 16 luồng không hiệu quả, có thể vì phân phối công việc không đều hoặc chi phí đồng bộ.

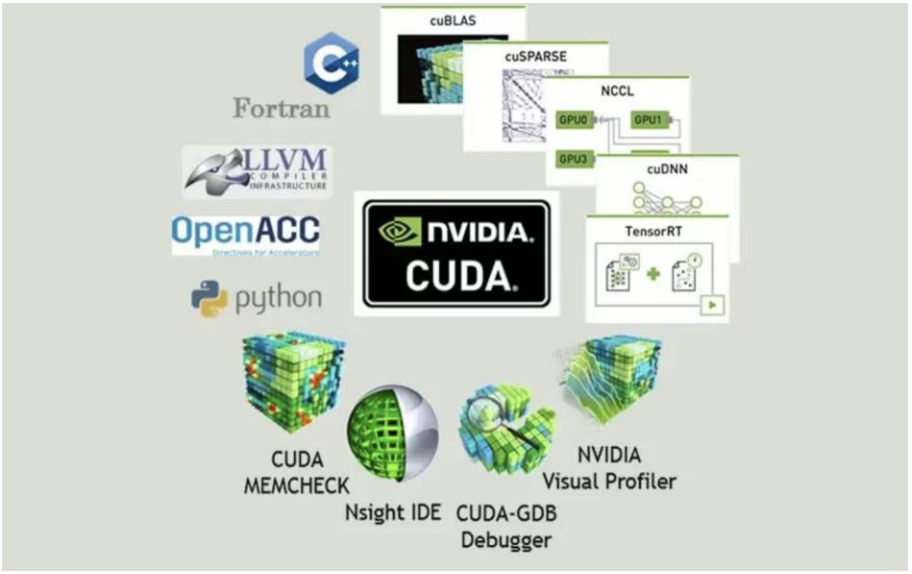

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

# SONG SONG HÓA SỬ DỤNG CUDA (GPU)

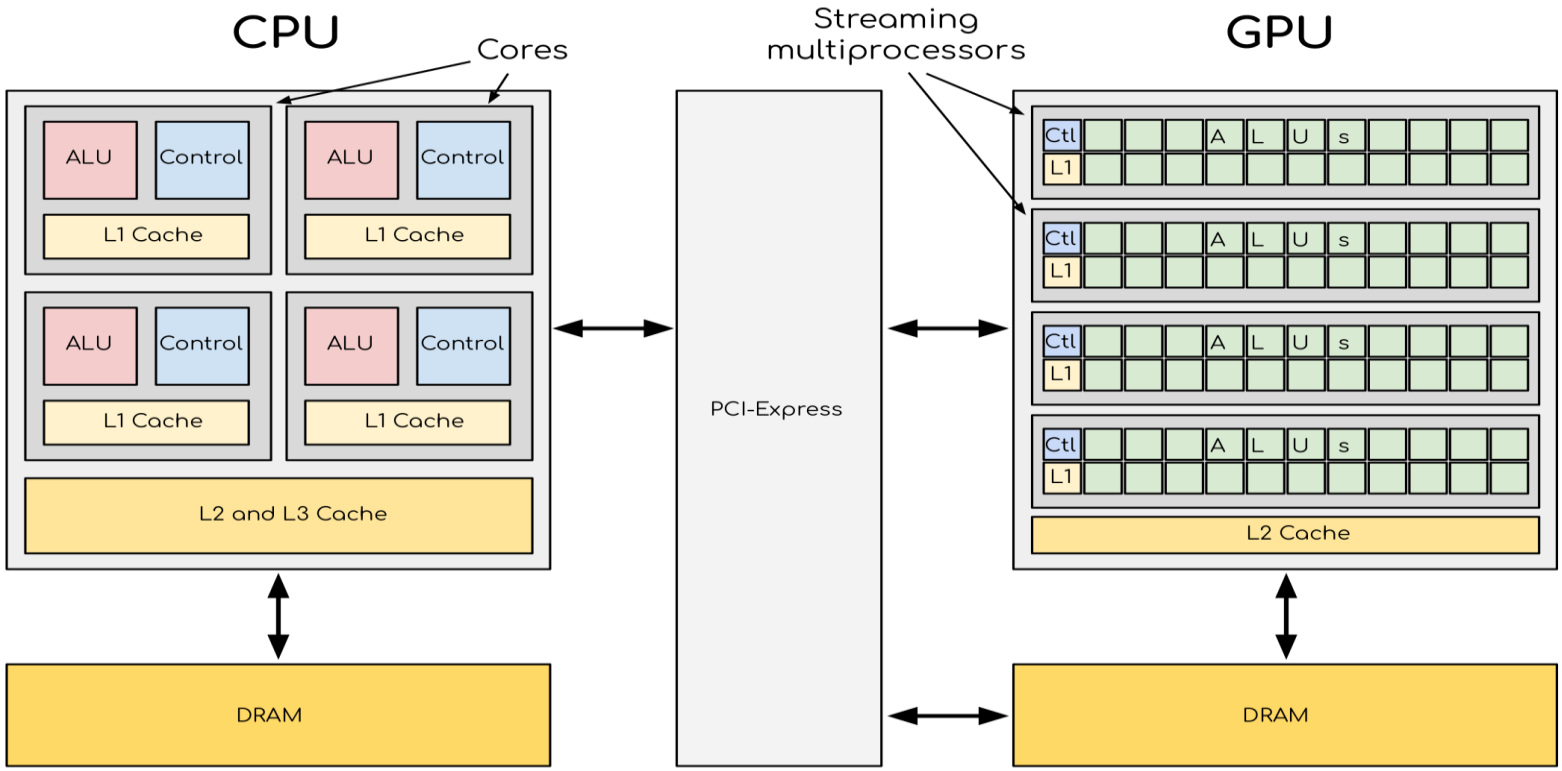
# CUDA LÀ GÌ ?

- **CUDA (Compute Unified Device Architecture)** là kiến trúc và mô hình lập trình do **NVIDIA** phát triển, cho phép lập trình **GPU** để **tăng tốc hiệu suất tính toán** cho các ứng dụng.
- CUDA hỗ trợ các ngôn ngữ như **C/C++, Python, Java, Fortran, MATLAB** và sử dụng trình biên dịch **NVCC** để chạy trực tiếp trên GPU.
- CUDA giúp GPU không chỉ xử lý đồ họa mà còn thực hiện các tác vụ như **AI, xử lý ảnh, mô phỏng vật lý, và tính toán khoa học** bằng cách khai thác hàng **nghìn lõi xử lý song song**.



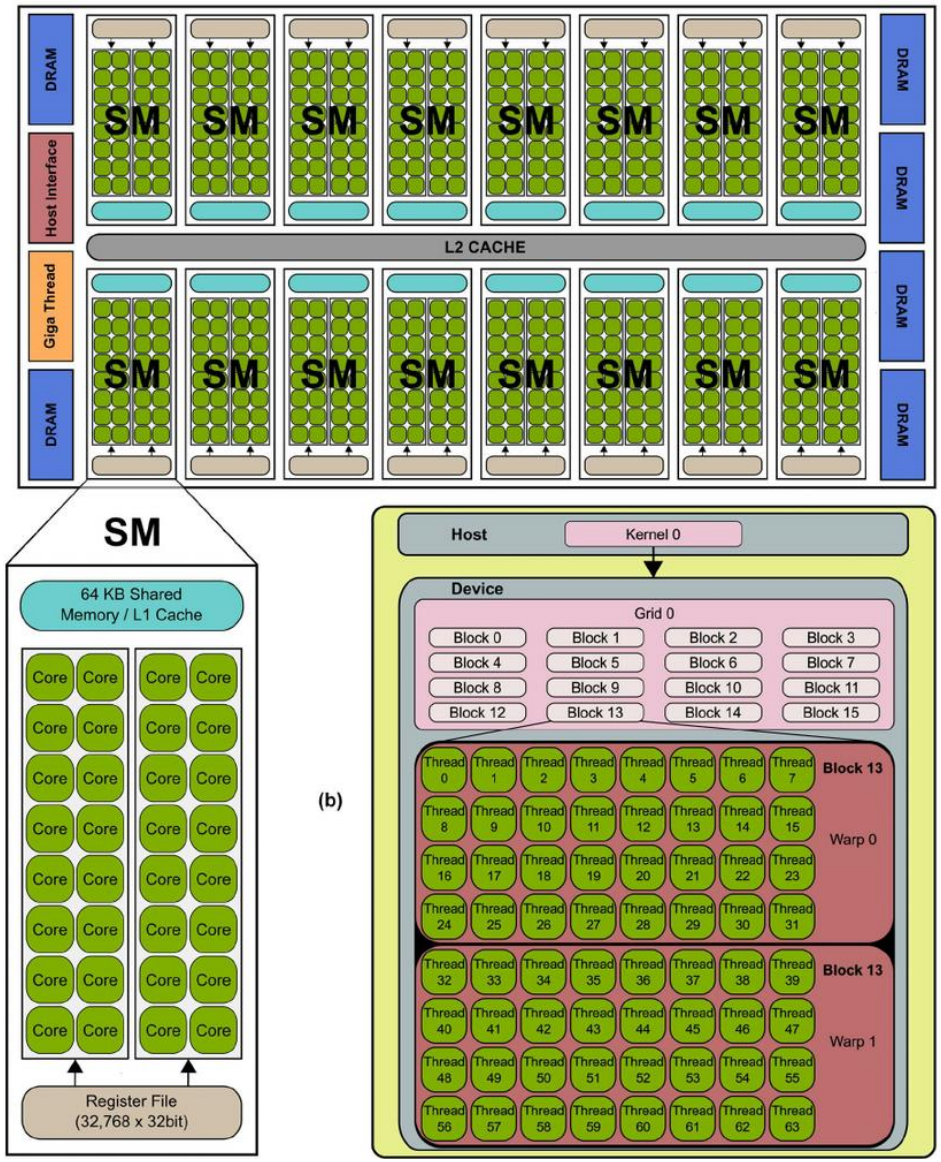
# KHÁC BIỆT GIỮA GPU VÀ CPU




- CPU thường được dùng đa tác vụ, tuần tự, phức tạp
- Ít lõi nhưng rất mạnh (4-16 cores), hoạt động độc lập tốt
- Mô hình xử lý tính toán **MIMT**
- Số lượng stack/core lớn

- GPU tập trung vào xử lý song song khối dữ liệu lớn
- Chứa hàng trăm - nghìn lõi, hoạt động tốt khi không branching
- Mô hình xử lý tính toán **SIMT**
- Băng thông lớn nhưng stack/core nhỏ

# KIẾN TRÚC GPU




- **Kiến trúc phần cứng GPU**

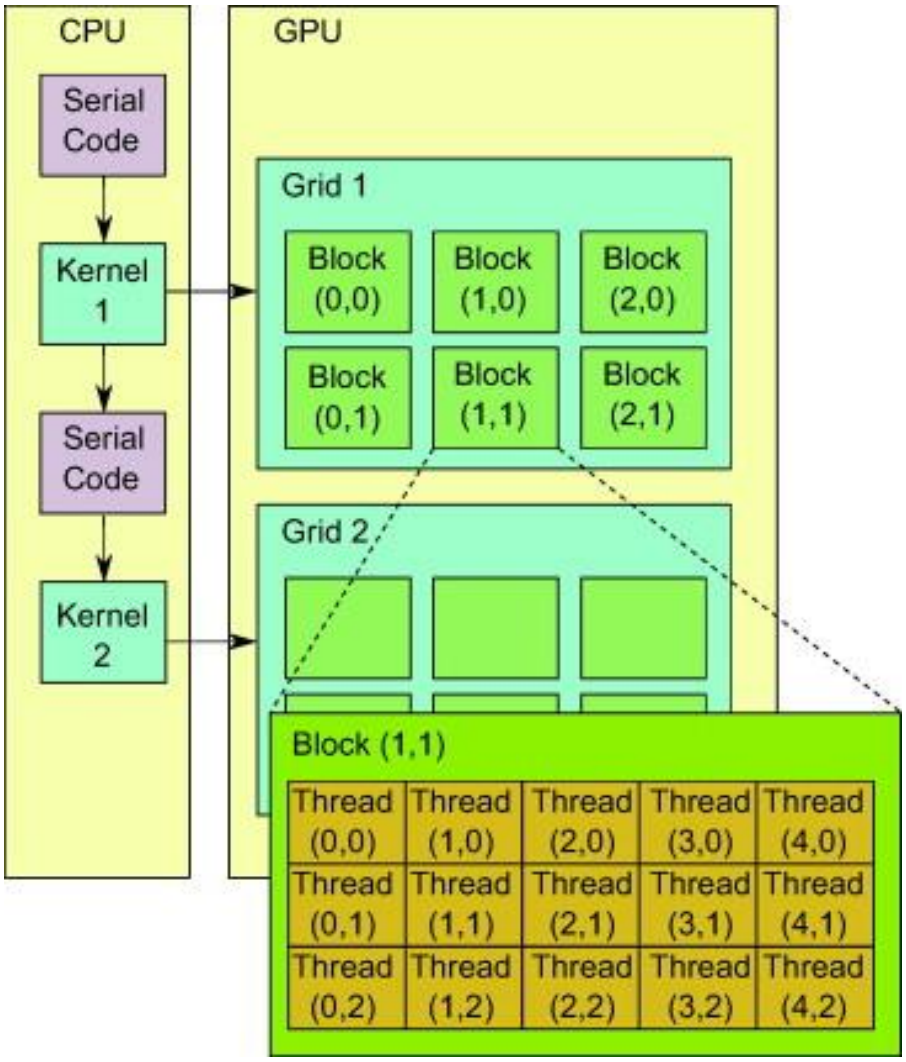
- GPU thường được "phân lô" bởi các **SM (Streaming Multiprocessor)**

- Mỗi SM là 1 đơn vị phần cứng xử lý tính toán nhưng **dùng 1 không gian memory** với nhau.

- Trong SM, chứa hàng chục đến trăm **cores** nhỏ hơn (32, 64, 128,..), mỗi **core** đảm nhiệm tác vụ tính toán. Ngoài ra còn các thành phần khác như **Register File, Shared Memory, Bộ Warp Scheduler,**...

- Các **core** này được thiết kế để chạy đồng thời và thực hiện 1 thao tác trên nhiều dữ liệu ! (Mô hình SIMT)

# KIẾN TRÚC GPU




- **Kiến trúc phần mềm CUDA (để ánh xạ lên phần cứng)**

➢ *Để ánh xạ lên phần cứng, cần chia công việc thành các khối và trong khối đó chứa các thread để xử lý !*

- **Grid:** là một lưới chứa các **blocks**

- **Blocks**: là các "bể" trong đó chứa các threads (1 block tối đa chứa 1024 threads). Một SM có thể carry tối đa 32 blocks cùng lúc (tùy kiến trúc), số thread trong block có thể thay đổi.

- **Warp**: là đơn vị nhỏ hơn bên trong block dùng để phân cụm các threads theo nhóm (**1 warp = 32 threads**). Một block có thể chứa tối đa 32 warps

# KIẾN TRÚC GPU

Việc ánh xạ xảy ra như sau:

| Phần mềm (CUDA) ⟶ | Phần cứng (GPU) |
|---|---|
| Threads | CUDA cores |
| Warp (32 threads) | Warp Scheduler trên SM |
| Blocks | Gán lên các SM |
| Grid | Phân phối toàn GPU |

- Mỗi khi chạy kernel, CUDA sẽ chia đều tài nguyên các block vào các SM. Sau đó, tùy vào số threads mỗi blocks, các threads được ánh xạ vào các cores và warp scheduler sẽ chia thành từng warps (32 threads)/block để xử lý. GPU sẽ chỉ xử lý tác vụ theo mỗi đơn vị warp. Một SM có thể chứa tối đa lên đến 2048 threads (tùy kiến trúc)
- Mỗi SM đảm nhiệm 1 ánh xạ của block từ phần mềm, nếu có nhiều block được xử lý thì sẽ luân phiên nhau "xếp hàng" chờ xử lý trên SM.
- Nếu SM có nhiều cores, thì có thể thực hiện song song nhiều warp cùng lúc hiệu quả hơn !
- Tài nguyên của SM bao gồm **Registers, Shared Memory,** số lượng **block/thread** tối đa

# KIẾN TRÚC GPU

TUY NHIÊN, GPU có 1 nhược điểm:

**Các lõi xử lý song song theo SIMT (Single Instruction Multiple Threads)**

- Nghĩa là 1 lệnh được thực hiện cùng lúc trên nhiều threads khác nhau
- Một warp chạy 1 lệnh giống nhau trên 32 threads (giống như 32 công nhân cùng làm 1 thao tác trên 32 sản phẩm)

**Số lượng Stack hạn chế trên mỗi core**

- GPU có **stack per thread**, nhưng rất nhỏ và giới hạn về dung lượng (vài KB), chỉ thích hợp làm tác vụ tính toán nhẹ
- Với đệ quy sâu, dễ bị tràn bộ nhớ (**Stack-Over-Flow**)

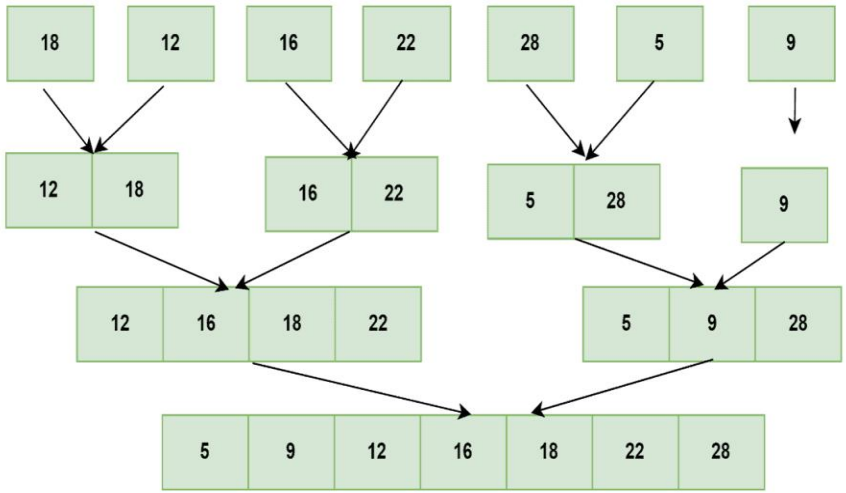
→ Nếu **đệ quy sâu** hoặc **rẽ nhánh if/else** khác nhau giữa các threads, thì các thread trong 1 warp sẽ bị **divergence** (phân kỳ), gây giảm hiệu năng !
→ Vì thế thuật toán Merge Sort truyền thống sử dụng đệ quy sẽ cần thay đổi thanh kiểu tuần tự mà nhiều thread có thể chạy cùng lúc mà không bị **race-condition**
→ **Đề xuất thuật toán Bottom-Up iterative Merge Sort**
*(Nguồn tham khảo: iterative-merge-sort)*

# Ý TƯỞNG TRIỂN KHAI

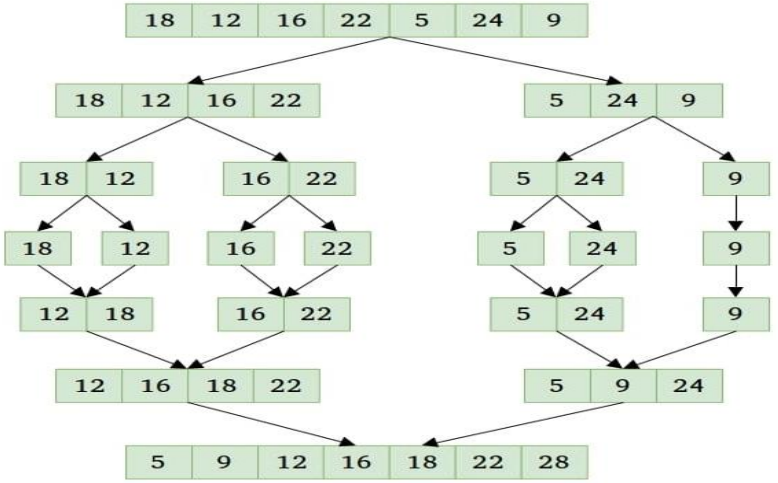
→ Thay vì chia mảng ra làm 2 đến khi mảng còn 1 phần tử rồi merge lại như truyền thống (Top-Bottom), **thuật toán này làm ngược lại** !

**Bottom-Up Merge Sort (từ đoạn nhỏ ghép dần lên):**

- Bắt đầu từ dưới lên với các phần tử riêng lẻ (1 phần tử/mảng)
- Mỗi lần chạy, hợp nhất các cặp liền kề của các mảng con đã được sắp xếp để tạo ra các mảng con có kích thước lớn hơn theo thứ tứ tăng dần.
- Sau mỗi vòng lặp, kích thước mảng con sẽ gấp đôi so với trước đó (1, 2, 4, 8,..)




*Bottom-Up Merge Sort*




*Top-Bottom Merge Sort*

# Ý TƯỞNG TRIỂN KHAI

**Hardware: NVIDIA GeForce RTX 3050 (desktop):**

- Kiến trúc Ampere
- **16 SMs** (Streaming Multiprocessor)
- **128 cores** mỗi SM → Tổng **2048 cores CUDA**
- Giới hạn **16 blocks/SM**
- Tối đa **1536 threads/SM** → Tối đa **48 warps/SM**
- Tối đa **1024 threads/block** → Tổng **32 warps/block**

*(Nếu có 1 block tối đa 1024 threads thì SM chỉ giữ được 1 block như vậy)*

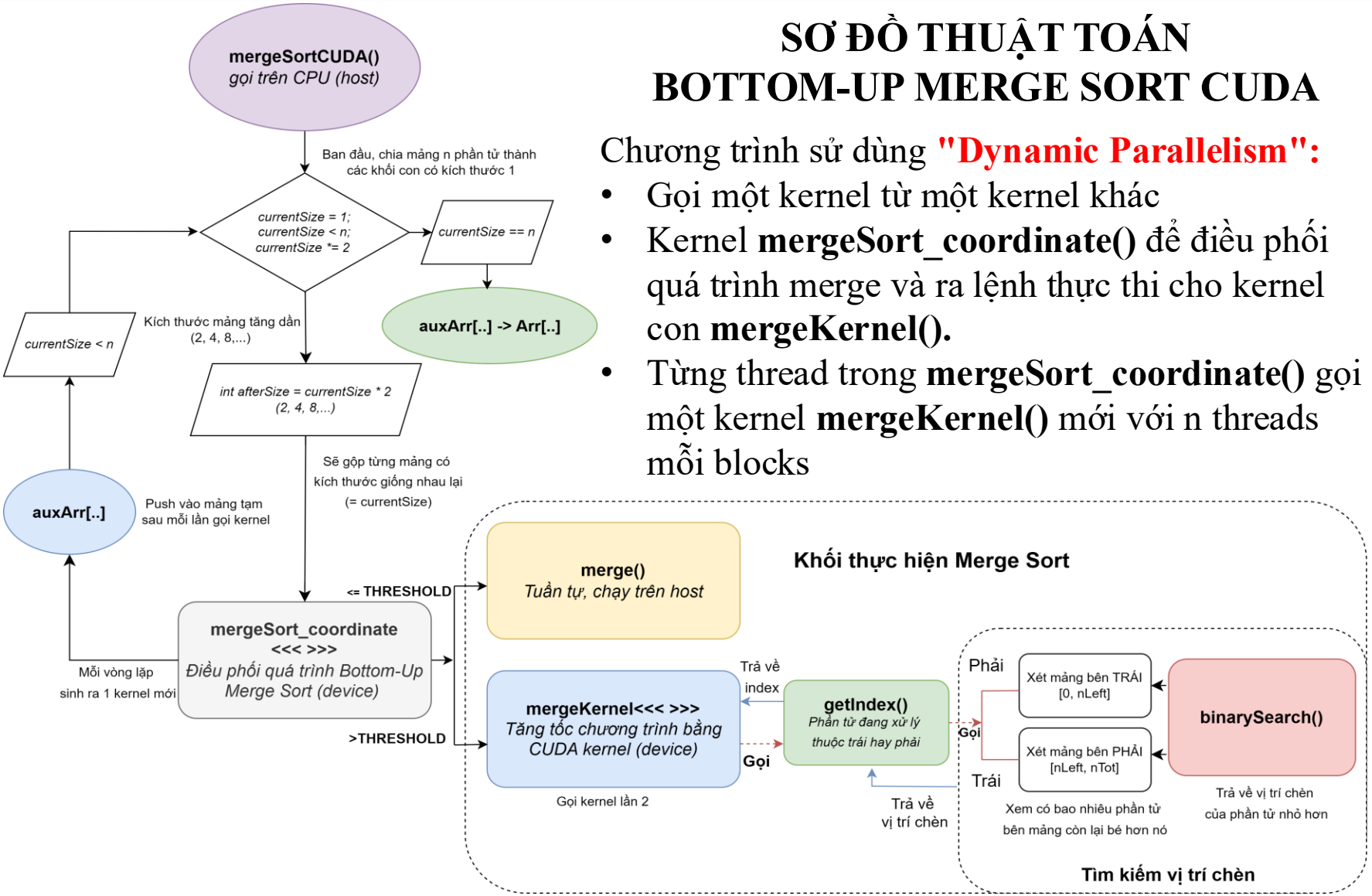
**Software: CUDA version 12.8**

```
Device 0: NVIDIA GeForce RTX 3050 Laptop GPU
1. Multiprocessor (SMs): 16
2. Maximum blocks per SM: 16
3. Maximum threads per SM: 1536
4. Maximum threads per Block: 1024
 -> x: 1024, y: 1024, z: 64
5. Registers per block: 65536
6. Shared Memory per Block: 49152 bytes
7. CUDA Cores per SM: 128 cores/SM
 -> Total CUDA cores: 2048
```

**Kịch bản kiểm thử:**

- Chạy và so sánh hiệu suất khi sử dụng Bottom-Up Merge Sort cho 100 triệu phần tử với số lượng blocks và threads khác nhau
- Đánh giá hiệu suất và khả năng tăng tốc

# Ý TƯỞNG TRIỂN KHAI




## SƠ ĐỒ THUẬT TOÁN BOTTOM-UP MERGE SORT CUDA

Chương trình sử dùng **"Dynamic Parallelism":**

- Gọi một kernel từ một kernel khác
- Kernel **mergeSort_coordinate()** để điều phối quá trình merge và ra lệnh thực thi cho kernel con **mergeKernel().**
- Từng thread trong **mergeSort_coordinate()** gọi một kernel **mergeKernel()** mới với n threads mỗi blocks

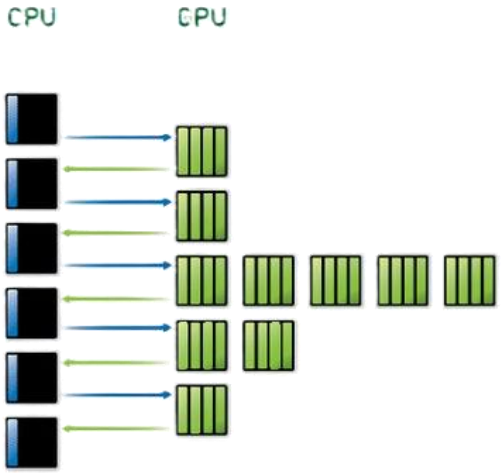
# Ý TƯỞNG TRIỂN KHAI

**Lý do chọn Dynamic Parallism (Kernel lồng kernel) để thực thi:**

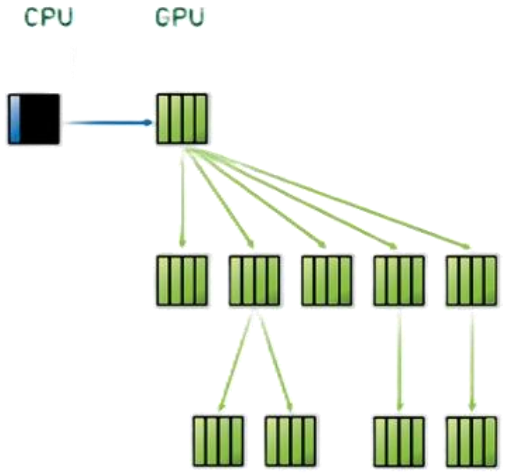
- Là tính năng của CUDA cho phép 1 **kernel** có thể launch 1 **kernel** khác trực tiếp trên **device (GPU)** mà không cần quay lại **host (CPU)** để yêu cầu lại.
- Các thread sinh ra từ lần gọi **kernel cha** sẽ gọi tiếp các **kernel con** ngay tại **device**

→ Tránh phải quay về **host** để launch kernel mới, tiết kiệm được thời gian và overhead

- **Bottom-Up Merge Sort** tận dùng điều này để điều phối việc chia nhỏ mảng thành các phần nhỏ hơn và trong đó lại tiếp tục so sánh và hợp nhất các mảng con đó lại thành mảng lớn ! Vì bản chất của **Bottom-Up** là từ các đoạn nhỏ ghép dần lên



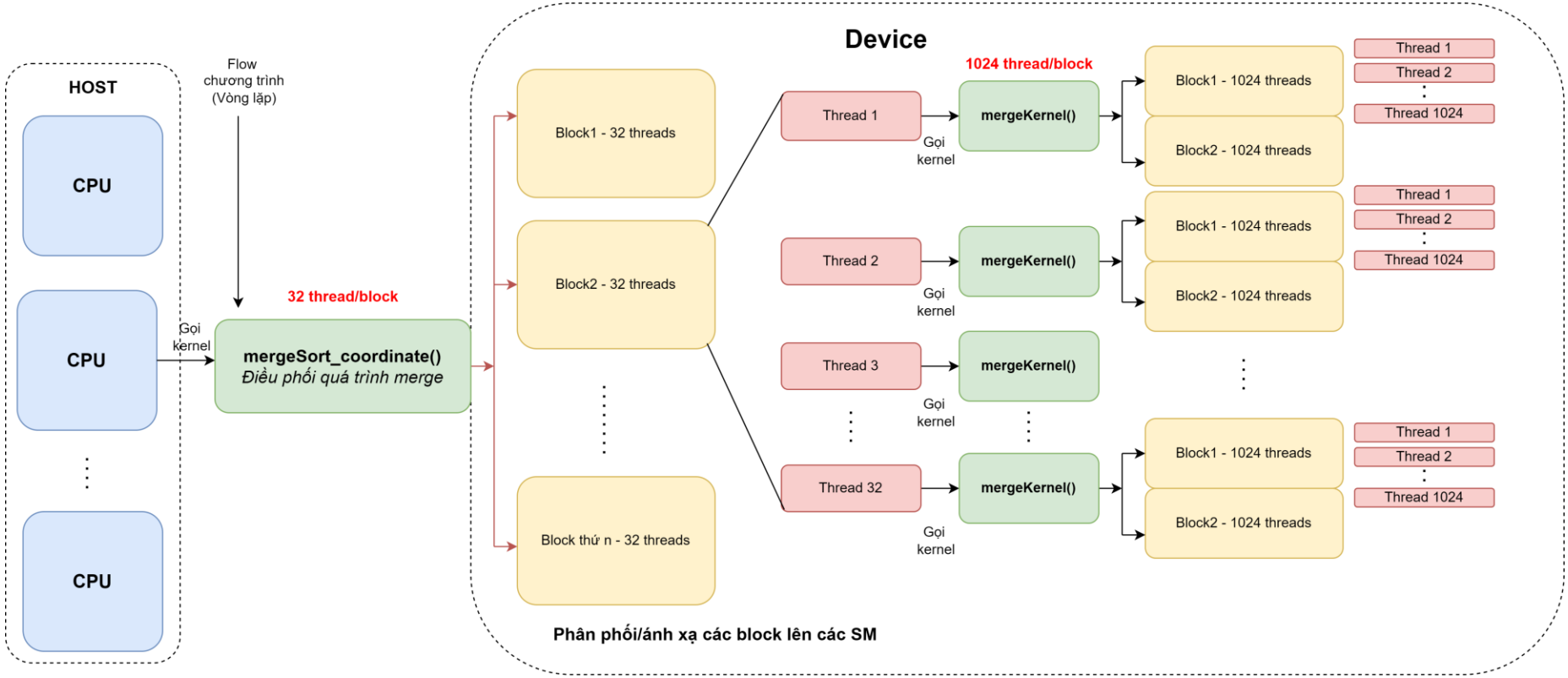

*CPU launched kernel (truyền thống)*




*Dynamic Parallelism*

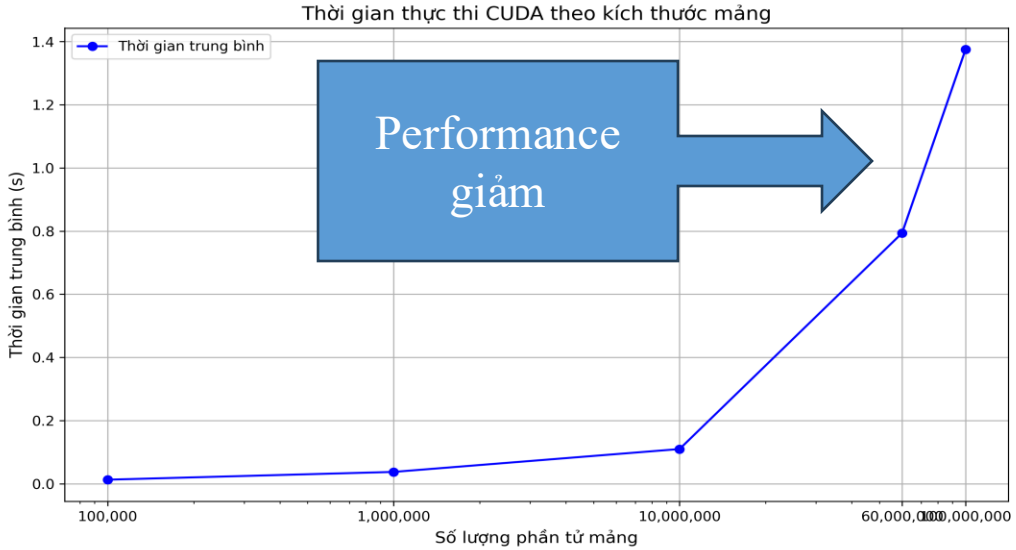
# Ý TƯỞNG TRIỂN KHAI

## Flow chương trình Bottom-Up Merge Sort sủ dụng Dynamic Parallelism



# THỰC THI VÀ SO SÁNH

## TH1: Thử nghiệm 1024 threads/block cho kernel mergeKernel() + 32 threads/block cho kernel mergeSort_coordinate()




| Số phần tử | Thời gian chạy(s) |
|---|---|
| 100_000 | 0.0129380842 |
| 1_000_000 | 0.0375562347 |
| 10_000_000 | 0.1099893951 |
| 60_000_000 | 0.7936757568 |
| 100_000_000 | 1.3747836507 |

- *Với **32 threads**, 100k phần tử → BottomUp **lần 1** cần merge **50k** lần → Sinh ra **1563 blocks**, chia đều lên **16 SMs** → Mỗi SM gánh **98 blocks** → Luôn phiên xếp hàng xử lý*

→ Gửi vào kernel con **mergeKernel()** để tiếp tục chia nhỏ việc

- Set tối đa số threads mỗi block trong **mergeKernel()** = **1024 threads/block**

→ Số warps mỗi blocks tăng (**32 warps**) nhưng số blocks mỗi SM bị giới hạn (**1 blocks/SM**)

→ Không tận dụng được tối đa tài nguyên phần cứng (các SM), tuy nhiều warps/block nhưng ít blocks được active thì cũng ít warps được active cùng lúc, khó che đi latency (độ trễ)

# THỰC THI VÀ SO SÁNH

## Thời gian thực thi các hàm của chương trình (100 triệu phần tử)

| Time (%) | Total Time (ns) | Num Calls | Avg (ns) | Med (ns) | Min (ns) | Max (ns) | StdDev (ns) | Name |
|---|---|---|---|---|---|---|---|---|
| 77,0 | 1228439505 | 54 | 22748879,0 | 4345622,0 | 788 | 103151966 | 31737503,0 | cudaDeviceSynchronize |
| 11,0 | 187287883 | 29 | 6458202,0 | 33043,0 | 12501 | 112826786 | 24547043,0 | cudaMemcpy |
| 8,0 | 134112879 | 2 | 67056439,0 | 67056439,0 | 1453 | 134111426 | 94830071,0 | cudaEventCreate |
| 1,0 | 16450111 | 27 | 609263,0 | 13994,0 | 8671 | 16059248 | 3087717,0 | cudaLaunchKernel |
| 0,0 | 5107860 | 2 | 2553930,0 | 2553930,0 | 2398008 | 2709852 | 220507,0 | cudaFree |
| 0,0 | 3502093 | 2 | 1751046,0 | 1751046,0 | 1551688 | 1950405 | 281935,0 | cudaMalloc |
| 0,0 | 2397765 | 1 | 2397765,0 | 2397765,0 | 2397765 | 2397765 | 0,0 | cuLibraryUnload |
| 0,0 | 927053 | 2 | 463526,0 | 463526,0 | 39944 | 887109 | 599036,0 | cudaEventRecord |
| 0,0 | 907386 | 1 | 907386,0 | 907386,0 | 907386 | 907386 | 0,0 | cuModuleGetLoadingMode |
| 0,0 | 6963 | 1 | 6963,0 | 6963,0 | 6963 | 6963 | 0,0 | cudaEventSynchronize |
| 0,0 | 3535 | 1 | 3535,0 | 3535,0 | 3535 | 3535 | 0,0 | cuCtxSynchronize |
| 0,0 | 1904 | 2 | 952,0 | 952,0 | 401 | 1503 | 779,0 | cudaEventDestroy |
| 0,0 | 505 | 1 | 505,0 | 505,0 | 505 | 505 | 0,0 | cuDeviceGetLuid |

[6/8] Executing 'cuda_gpu_kern_sum' stats report

| Time (%) | Total Time (ns) | Instances | Avg (ns) | Med (ns) | Min (ns) | Max (ns) | StdDev (ns) | Name |
|---|---|---|---|---|---|---|---|---|
| 100,0 | 819502883 | 27 | 30351958,0 | 8578544,0 | 10752 | 98407127 | 38264450,0 | mergeSort_coordinate |

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

| Time (%) | Total Time (ns) | Count | Avg (ns) | Med (ns) | Min (ns) | Max (ns) | StdDev (ns) | Operation |
|---|---|---|---|---|---|---|---|---|
| 45,0 | 128517170 | 27 | 4759895,0 | 4754853,0 | 4751846 | 4782341 | 11369,0 | [CUDA memcpy Device-to-Device] |
| 28,0 | 80402536 | 1 | 80402536,0 | 80402536,0 | 80402536 | 80402536 | 0,0 | [CUDA memcpy Host-to-Device] |
| 25,0 | 72875238 | 1 | 72875238,0 | 72875238,0 | 72875238 | 72875238 | 0,0 | [CUDA memcpy Device-to-Host] |

- ✓ Mất **27 lần** chạy kernel để merge 100 triệu phần tử (~**819.5ms**) mỗi lần ~**30.35ms.**
- ✓ Thời gian đồng bộ giữa CPU và GPU nhiều nhất, tốn **1228.42ms** (~**77%**), mất **54 lần** gọi.
- ✓ Thời gian sao chép dữ liệu từ Host ↔ Device tốn **187.28ms** (~**11%**), mất **29** lần gọi. Thực tế tổng thời gian mất **221ms.** **(Mũi tên xanh)**
- ✓ Thời gian gọi kernel từ CPU và đẩy sang GPU tốn **16.45ms** (~**1%**)

# THỰC THI VÀ SO SÁNH

**Hiệu suất một kernel: Speed of Light – So sánh hiệu suất thực tế và lý thuyết**

```
mergeSort_coordinate (1, 1, 1)x(32, 1, 1), Context 1, Stream 7, Device 0, CC 8.6
  Section: GPU Speed Of Light Throughput
  ---------------------- ----------- ------------
  Metric Name            Metric Unit  Metric Value
  ---------------------- ----------- ------------
  DRAM Frequency                 Ghz         5,50
  SM Frequency                   Mhz       712,50
  Elapsed Cycles               cycle   38.091.786
  Memory Throughput                %        30,53
  DRAM Throughput                  %        12,09
  Duration                        ms        53,46
  L1/TEX Cache Throughput          %        31,37
  L2 Cache Throughput              %         8,13
  SM Active Cycles             cycle 37.047.719,06
  Compute (SM) Throughput          %        69,39
  ---------------------- ----------- ------------
```

- Hiệu suất kernel **mergeSort_coordinate()** tính toán chỉ đạt ~ **69.39 %**, vẫn chưa tận dụng hết sức mạnh của SM
- **Memory Throughput** chỉ **~30.53 %, DRAM ~12.09 %** → Chưa tối ưu khả năng truy cập bộ nhớ do chiếm nhiều tài nguyên trên mỗi SM , ít warp hoạt động đồng thời
- Bộ nhớ **Cache L1 (~31.37 %)** ưu tiên hơn **L2(~8.13%)** (do sử dụng hàm trong code) nên được tận dụng tốt nhưng vẫn cần phân bổ lại để tối ưu hơn.

→ **Compute SM Throughput** ổn nhưng **Memory** và **Cache** vẫn chưa được khai thác hiệu quả

# THỰC THI VÀ SO SÁNH

## Hiệu suất một kernel – Occupancy (Độ chiếm dụng)

```
Section: Occupancy
------------------------------ ----------- ------------
Metric Name                    Metric Unit Metric Value
------------------------------ ----------- ------------
Block Limit SM                       block           16
Block Limit Registers                block           64
Block Limit Shared Mem               block           16
Block Limit Warps                    block           48
Theoretical Active Warps per SM       warp           16
Theoretical Occupancy                    %        33,33
Achieved Occupancy                       %        63,90
Achieved Active Warps Per SM          warp        30,67
------------------------------ ----------- ------------
```

- **Occupancy** là tỷ lệ giữa số lượng warp đang active thực tế trên mỗi SM so với lượng warp tối đa mà phần cứng hỗ trợ → Biết được mức độ khai thác tài nguyên phần cứng
- **Theoretical Occupancy (lý thuyết) ~ 33.33 %** do bị giới hạn lượng block tối đa của chương trình **(Block Limit SM = 16)** và **shared memory**

**Nguyên nhân giới hạn lý thuyết :**

- Do chương trình dùng nhiều **shared memory/block**, mỗi **block** xin 1 ít bộ nhớ

→ Giới hạn lượng lượng **block** tổng thể

- **Achieved Occupany (Độ chiếm dụng thực tế)** lại đạt ~ **63.90 %,** tương đối ổn.
- **Block Limits Warps → 48 warps** có thể đồng thời cư trú trên 1 SM (*thể hiện đúng phần cứng*)

→ Đạt được **30.67 warp** active/**48 warps** phần cứng → Khai thác tài nguyên gần mức tối đa

→ Kích thước **1024 threads** khiến giới hạn số block có thể chạy cùng lúc trên SM (**1 block**), tối đa hóa được băng thông xử lý nhưng tiêu hao tài nguyên bộ nhớ, khó giấu latency !

→ Vẫn tối ưu và tăng Occupancy thêm được !

# THỰC THI VÀ SO SÁNH

## Hiệu suất kernel: Launch Statistics (Thống kê khởi chạy từng kernel)

| ID | Estimated Speedup [%] | Function Name | Demangled Name | Duration [ms] (1.735,61 ms) | Runtime Improvement [ms] (525,88 ms) | # Registers [register/thread | Grid Size | Block Size [block] |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 8,07 | 0,00 | 32 | 781250, 1, ... | 32, 1, ... |
| 2 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 23,42 | 0,00 | 32 | 390625, 1, ... | 32, 1, ... |
| 3 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 27,78 | 0,00 | 32 | 195313, 1, ... | 32, 1, ... |
| 4 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 42,07 | 0,00 | 32 | 97657, 1, ... | 32, 1, ... |
| 5 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 62,98 | 0,00 | 32 | 48829, 1, ... | 32, 1, ... |
| 6 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 82,40 | 0,00 | 32 | 24415, 1, ... | 32, 1, ... |
| 7 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 119,80 | 0,00 | 32 | 12208, 1, ... | 32, 1, ... |
| 8 | 0.00 | mergeSort_coordi... | mergeSort_coordi... | 133,83 | 0,00 | 32 | 6104, 1, ... | 32, 1, ... |
| 9 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 133,66 | 0,00 | 32 | 3052, 1, ... | 32, 1, ... |
| 10 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 134,66 | 0,00 | 32 | 1526, 1, ... | 32, 1, ... |
| 11 | 33.33 | mergeSort_coordin... | mergeSort_coordin... | 130,32 | 43,44 | 32 | 763, 1, ... | 32, 1, ... |
| 12 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 116,96 | 58,48 | 32 | 382, 1, ... | 32, 1, ... |
| 13 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 83,79 | 0,00 | 32 | 191, 1, ... | 32, 1, ... |
| 14 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 42,59 | 0,00 | 32 | 96, 1, ... | 32, 1, ... |
| 15 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 41,67 | 0,00 | 32 | 48, 1, ... | 32, 1, ... |
| 16 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 38,35 | 0,00 | 32 | 24, 1, ... | 32, 1, ... |
| 17 | 25.00 | mergeSort_coordin... | mergeSort_coordin... | 36,95 | 9,24 | 32 | 12, 1, ... | 32, 1, ... |
| 18 | 62.50 | mergeSort_coordin... | mergeSort_coordin... | 36,99 | 23,12 | 32 | 6, 1, ... | 32, 1, ... |
| 19 | 81.25 | mergeSort_coordin... | mergeSort_coordin... | 37,57 | 30,52 | 32 | 3, 1, ... | 32, 1, ... |
| 20 | 87.50 | mergeSort_coordin... | mergeSort_coordin... | 38,88 | 34,02 | 32 | 2, 1, ... | 32, 1, ... |
| 21 | 93.75 | mergeSort_coordin... | mergeSort_coordin... | 57,19 | 53,62 | 32 | 1, 1, ... | 32, 1, ... |

- **Runtime Improvement** = **Runtime lần trước - Runtime lần này (ms).** Là mức giảm thời gian thực thi của 1 kernel giữa các lần gọi, càng lớn thì kernel sau chạy hiệu quả hơn, thể hiện mức độ cải thiện thời gian
- **Estimated SpeedUp (%) = (Runtime Improvement – Runtime lần trước) x 100**. Là thời gian tăng tốc ước lượng giữa kernel lần này chạy nhanh hơn bao nhiêu lần số với trước đó. Nếu số này càng lớn thì khả năng tăng tốc càng nhiều

# THỰC THI VÀ SO SÁNH

## Hiệu suất kernel: Launch Statistics (Thống kê khởi chạy từng kernel)

| ID | Estimated Speedup [%] | Function Name | Demangled Name | Duration [ms] (1.735,61 ms) | Runtime Improvement [ms] (525,88 ms) | # Registers [register/thread | Grid Size | Block Size [block] |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 8,07 | 0,00 | 32 | 781250, 1, ... | 32, 1, ... |
| 2 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 23,42 | 0,00 | 32 | 390625, 1, ... | 32, 1, ... |
| 3 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 27,78 | 0,00 | 32 | 195313, 1, ... | 32, 1, ... |
| 4 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 42,07 | 0,00 | 32 | 97657, 1, ... | 32, 1, ... |
| 5 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 62,98 | 0,00 | 32 | 48829, 1, ... | 32, 1, ... |
| 6 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 82,40 | 0,00 | 32 | 24415, 1, ... | 32, 1, ... |
| 7 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 119,80 | 0,00 | 32 | 12208, 1, ... | 32, 1, ... |
| 8 | 0.00 | mergeSort_coordi... | mergeSort_coordi... | 133,83 | 0,00 | 32 | 6104, 1, ... | 32, 1, ... |
| 9 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 133,66 | 0,00 | 32 | 3052, 1, ... | 32, 1, ... |
| 10 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 134,66 | 0,00 | 32 | 1526, 1, ... | 32, 1, ... |
| 11 | 33.33 | mergeSort_coordin... | mergeSort_coordin... | 130,32 | 43,44 | 32 | 763, 1, ... | 32, 1, ... |
| 12 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 116,96 | 58,48 | 32 | 382, 1, ... | 32, 1, ... |
| 13 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 83,79 | 0,00 | 32 | 191, 1, ... | 32, 1, ... |
| 14 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 42,59 | 0,00 | 32 | 96, 1, ... | 32, 1, ... |
| 15 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 41,67 | 0,00 | 32 | 48, 1, ... | 32, 1, ... |
| 16 | 0.00 | mergeSort_coordin... | mergeSort_coordin... | 38,35 | 0,00 | 32 | 24, 1, ... | 32, 1, ... |
| 17 | 25.00 | mergeSort_coordin... | mergeSort_coordin... | 36,95 | 9,24 | 32 | 12, 1, ... | 32, 1, ... |
| 18 | 62.50 | mergeSort_coordin... | mergeSort_coordin... | 36,99 | 23,12 | 32 | 6, 1, ... | 32, 1, ... |
| 19 | 81.25 | mergeSort_coordin... | mergeSort_coordin... | 37,57 | 30,52 | 32 | 3, 1, ... | 32, 1, ... |
| 20 | 87.50 | mergeSort_coordin... | mergeSort_coordin... | 38,88 | 34,02 | 32 | 2, 1, ... | 32, 1, ... |
| 21 | 93.75 | mergeSort_coordin... | mergeSort_coordin... | 57,19 | 53,62 | 32 | 1, 1, ... | 32, 1, ... |

- Các lần gọi kernel từ 0-10 đều không có cải thiện hiệu suất (**Runtime Improvement ~ 0 ms, Estimated Speedup ~ 0 %**)
- Bắt đầu từ kernel thứ 11, có sự cải thiện đáng kể (**SpeedUp ~ 33 %, Rumtime Improvement ~ 43,44 ms**) và đến kernel thứ 12 thì tăng được **50 %** so với trước đó
- Đặc biệt từ kernel 21 trở đi, tăng tốc ước tính đạt **93.77 %**
- Các lần gọi có Duration ngắn (< 40ms), các mức tăng tốc đáng kể lại xuất hiện, chứng tỏ một số kernel con có thể đang bị **nghẽn hiệu suất**

# THỰC THI VÀ SO SÁNH

## Hiệu suất kernel: Launch Statistics (Thống kê khởi chạy từng kernel)

| | | | | |
|---|---|---|---|---|
| Grid Size | Số block sinh ra trong lần merge đầu | 1.562.500 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 32 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | Số threads sinh ra trong lần merge đầu | 50.000.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 6.103,52 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

| | | | | |
|---|---|---|---|---|
| Grid Size | Số block sinh ra trong lầnmerge thứ 2 | 781.250 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 32 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | Số threads sinh ra trong lần merge thứ 2 | 25.000.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 3.051,76 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

| | | | | |
|---|---|---|---|---|
| Grid Size | Lần gọi thứ 3 | 390.625 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 32 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | | 12.500.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 1.525,88 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

| | | | | |
|---|---|---|---|---|
| Grid Size | Lần gọi thứ 4 | 195.313 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 32 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | | 6.250.016 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 762,94 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

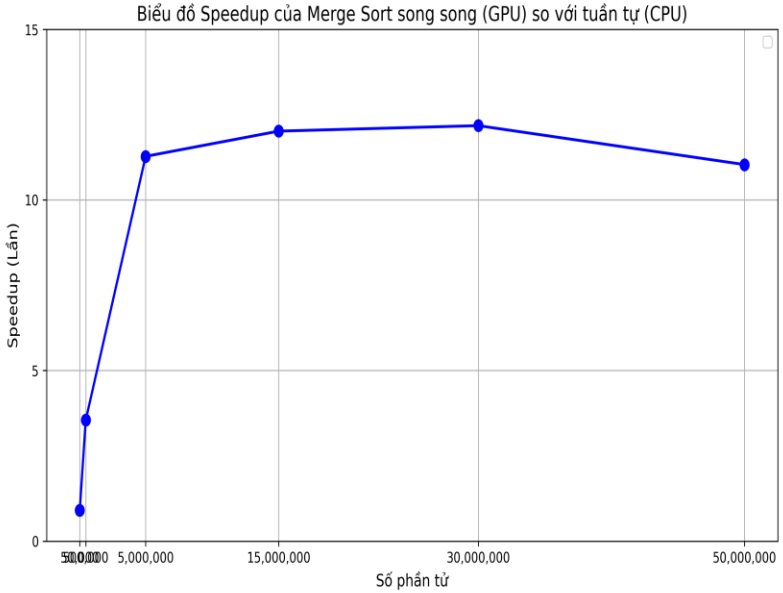
- Số **threads** và số **block** tỷ lệ nghịch với số lần gọi **kernel** là do sau mỗi lần merge thì kích thước các mảng con tăng dần đồng nghĩa số lượng mảng con cũng giảm đi

# THỰC THI VÀ SO SÁNH

## So sánh kết quả tăng tốc với tuần tự

- Tốc độ Speedup: $$\text{Speedup} = \frac{\text{Thời gian chạy trên CPU (tuần tự)}}{\text{Thời gian chạy trên GPU (song song)}}$$

| Phần tử | Merge Sort CPU (Tuần tự) | Bottom-Up Merge Sort (CUDA) | Số lần Speedup | Lượng threads (lớn nhất) |
|---|---|---|---|---|
| 100 000 | **0.0123999914** | **0.0136775061** | **0.906** | **50 000** |
| 1 000 000 | **0.1331333478** | **0.0375562347** | **3.546** | **500 000** |
| 10 000 000 | **1.3413999716** | **0.1189789790** | **11.274** | **5 000 000** |
| 30 000 000 | **4.4540000121** | **0.3707434998** | **12.013** | **15 000 000** |
| 60 000 000 | **9.6621333440** | **0.7936757568** | **12.174** | **30 000 000** |
| 100 000 000 | **15.170200300** | **1.3747836507** | **11.033** | **50 000 000** |



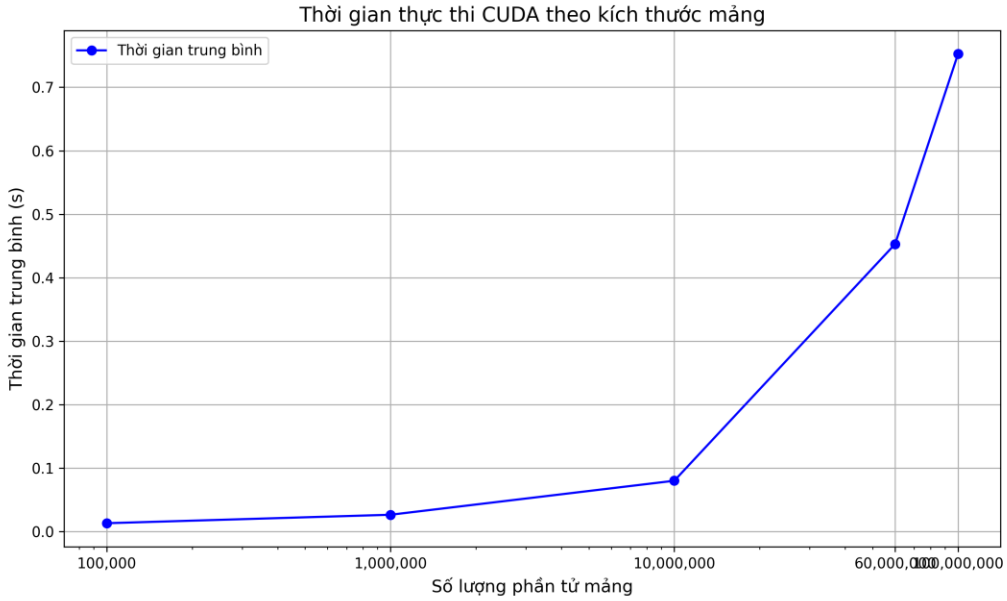

- Khả năng tăng tốc lớn nhất là **12.174** lần với **60 triệu phần tử**
- Đồ thị tăng tốc có dấu hiệu bão hòa khi đến ngưỡng 60 triệu phần tử và có dấu hiệu giảm dần
- Hiệu năng giảm dần với số lượng lớn phần tử do băng thông bộ nhớ bắt đầu bị giới hạn. Với 1024 threads/block, tài nguyên mỗi SM bị chiếm dụng nhiều, gây giảm Occupancy
- Dễ bị nghẽn khi dữ liệu tăng cao

# THỰC THI VÀ SO SÁNH


**Biểu đồ so sánh giữa tuần tự (CPU), OpenMP và song song trên CUDA**

# THỰC THI VÀ SO SÁNH

**TH2: Thử nghiệm với 256 threads/block cho kernel mergeKernel() + 16 threads/block cho kernel mergeSort_coordinate()**




| Số phần tử | Thời gian chạy(s) |
|---|---|
| 100_000 | 0.0128979390 |
| 1_000_000 | 0.0262354093 |
| 10_000_000 | 0.0798213343 |
| 60_000_000 | 0.4528704224 |
| 100_000_000 | 0.7526189901 |

- Tốc độ khi để **256 threads** nhanh hơn so với **1024 threads/block**. **Lí do:**

→ Do ít threads mà với số lần merge cố định thì cần nhiều blocks/SM hơn (4 blocks/SM)
→ Nhiều blocks thì tận dụng tối đa tài nguyên lên nhiều SM, khiến GPU luôn bận !
→ Tạo ra nhiều warps trên các block **được active cùng lúc**, giúp **che giấu đi latency** dễ hơn do khi có 1 warp đang đọc memory thì có thể switch sang warp khác để tiếp tục thực thi, tránh lãng phí tài nguyên.

# THỰC THI VÀ SO SÁNH

## Thời gian thực thi các hàm của chương trình (100 triệu phần tử)

| Time (%) | Total Time (ns) | Num Calls | Avg (ns) | Med (ns) | Min (ns) | Max (ns) | StdDev (ns) | Name |
|---|---|---|---|---|---|---|---|---|
| 62,0 | 606345616 | 54 | 11228622,0 | 6531135,0 | 1097 | 28576462 | 11687614,0 | cudaDeviceSynchronize |
| 21,0 | 204099587 | 29 | 7037916,0 | 59705,0 | 23962 | 112957305 | 26230830,0 | cudaMemcpy |
| 12,0 | 125020174 | 2 | 62510087,0 | 62510087,0 | 1628 | 125018546 | 88400310,0 | cudaEventCreate |
| 1,0 | 17091256 | 27 | 633009,0 | 18220,0 | 11013 | 16619702 | 3194978,0 | cudaLaunchKernel |
| 0,0 | 6806226 | 2 | 3403113,0 | 3403113,0 | 2161483 | 4644743 | 1755930,0 | cudaFree |
| 0,0 | 3140714 | 2 | 1570357,0 | 1570357,0 | 1387479 | 1753235 | 258628,0 | cudaMalloc |
| 0,0 | 1800100 | 1 | 1800100,0 | 1800100,0 | 1800100 | 1800100 | 0,0 | cuLibraryUnload |
| 0,0 | 1269471 | 2 | 634735,0 | 634735,0 | 44422 | 1225049 | 834829,0 | cudaEventRecord |
| 0,0 | 595550 | 1 | 595550,0 | 595550,0 | 595550 | 595550 | 0,0 | cuModuleGetLoadingMode |
| 0,0 | 8798 | 1 | 8798,0 | 8798,0 | 8798 | 8798 | 0,0 | cudaEventSynchronize |
| 0,0 | 3509 | 1 | 3509,0 | 3509,0 | 3509 | 3509 | 0,0 | cuCtxSynchronize |
| 0,0 | 2490 | 2 | 1245,0 | 1245,0 | 390 | 2100 | 1209,0 | cudaEventDestroy |
| 0,0 | 432 | 1 | 432,0 | 432,0 | 432 | 432 | 0,0 | cuDeviceGetLuid |

[6/8] Executing 'cuda_gpu_kern_sum' stats report

| Time (%) | Total Time (ns) | Instances | Avg (ns) | Med (ns) | Min (ns) | Max (ns) | StdDev (ns) | Name |
|---|---|---|---|---|---|---|---|---|
| 100,0 | 272217215 | 27 | 10082119,0 | 12518428,0 | 11520 | 23834142 | 9599281,0 | mergeSort_coordinate |

[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

| Time (%) | Total Time (ns) | Count | Avg (ns) | Med (ns) | Min (ns) | Max (ns) | StdDev (ns) | Operation |
|---|---|---|---|---|---|---|---|---|
| 43,0 | 128368711 | 27 | 4754396,0 | 4754503,0 | 4751942 | 4756582 | 1386,0 | [CUDA memcpy Device-to-Device] |
| 29,0 | 88868766 | 1 | 88868766,0 | 88868766,0 | 88868766 | 88868766 | 0,0 | [CUDA memcpy Device-to-Host] |
| 27,0 | 80846953 | 1 | 80846953,0 | 80846953,0 | 80846953 | 80846953 | 0,0 | [CUDA memcpy Host-to-Device] |

- ✓ Mất **27 lần** chạy kernel để merge 100 triệu phần tử (~**272.21ms**), mỗi lần ~**10.08ms.**
- ✓ Thời gian đồng bộ giữa CPU và GPU nhiều nhất, tốn **606.34ms** (~**62%**), mất **54 lần** gọi.
- ✓ Thời gian sao chép dữ liệu từ Host ↔ Device tốn **204.1ms** (~**11%**), mất **29 lần** gọi. Thực tế tổng thời gian lại tốn **298ms** **(Mũi tên xanh)**
- ✓ Thời gian gọi kernel từ CPU và đẩy sang GPU tốn **17.1ms** (~**1%**)

# THỰC THI VÀ SO SÁNH

**Hiệu suất một Kernel: Speed of Light – So sánh thời gian thực tế và lý**

```
mergeSort_coordinate (1, 1, 1)x(16, 1, 1), Context 1, Stream 7, Device 0, CC 8.6
  Section: GPU Speed Of Light Throughput
  ---------------------- ----------- -------------
  Metric Name            Metric Unit  Metric Value
  ---------------------- ----------- -------------
  DRAM Frequency                 Ghz          5,50
  SM Frequency                   Mhz        712,50
  Elapsed Cycles               cycle    29.268.887
  Memory Throughput                %         39,70
  DRAM Throughput                  %         15,72
  Duration                        ms         41,08
  L1/TEX Cache Throughput          %         39,76
  L2 Cache Throughput              %         11,02
  SM Active Cycles             cycle 29.227.131,81
  Compute (SM) Throughput          %         90,24
  ---------------------- ----------- -------------

  INF   This workload is utilizing greater than 80.0% of the available compute or memory performance of the device.
        To further improve performance, work will likely need to be shifted from the most utilized to another unit.
        Start by analyzing workloads in the Compute Workload Analysis section.
```

- Hiệu suất kernel **mergeSort_coordinate()** tính toán đạt **~90.24 %,** cho thấy tận dụng tốt sức mạnh của GPU (**Compute (SM) Throughput**)
- **Memory Throughput ~39.70 %**, **DRAM ~15.72 %** → Chưa khai thác tối đa băng thông bộ nhớ nhưng tốt hơn so với 1024 threads.
- **CacheL1 ~39.76%** cao do ưu tiên bằng hàm trong code, dẫn đến lượng truy cập L2 thấp (**~11.02 %**). Cân nhắc phân bổ lại **Cache L1** và **shared memory** để tối ưu hơn

# THỰC THI VÀ SO SÁNH

## Hiệu suất một kernel: Occupancy (Độ chiếm dụng)

```
Section: Occupancy
------------------------------ ----------- ------------
Metric Name                    Metric Unit Metric Value
------------------------------ ----------- ------------
Block Limit SM                       block           16
Block Limit Registers                block           64
Block Limit Shared Mem               block           16
Block Limit Warps                    block           48
Theoretical Active Warps per SM       warp           16
Theoretical Occupancy                    %        33,33
Achieved Occupancy                       %        93,41
Achieved Active Warps Per SM          warp        44,84
------------------------------ ----------- ------------

OPT   Est. Local Speedup: 66.67%
      The 4.00 theoretical warps per scheduler this kernel can issue according to its occupancy are below the
      hardware maximum of 12. This kernel's theoretical occupancy (33.3%) is limited by the number of blocks that
      can fit on the SM, and the required amount of shared memory.
```

- **Achieved Occupancy (Độ chiếm dụng thực tế) ~ 93.41 %**

→ Thực tế lại rất cao, cho thấy phần lớn tài nguyên SM được khai thác hiệu quả. Tăng khả năng che giấu độ trễ (latency) do truy cập bộ nhớ

- **Achieved Active Warps Per SM** đạt ~ **44.84 warps** trên giới hạn là **48 warps/SM**. So với TH1, khả năng tận dụng tài nguyên tốt hơn nhiều

# THỰC THI VÀ SO SÁNH

## Hiệu suất kernel: Launch Statistics (Thống kê khởi chạy từng kernel)

| ID | Estimated Speedup [%] | Function Name | Demangled Name | Duration [ms] (964,64 ms) | Runtime Improvement [ms] (596,46 ms) | # Registers [register/thread | Grid Size | Block Size [block] |
|---|---|---|---|---|---|---|---|---|
| 0 | 50.00 | mergeSort_coordi... | mergeSort_coordi... | 27,77 | 13,89 | 32 | 3125000, 1, ... | 16, 1, ... |
| 1 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 13,99 | 6,99 | 32 | 1562500, 1, ... | 16, 1, ... |
| 2 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 22,05 | 11,02 | 32 | 781250, 1, ... | 16, 1, ... |
| 3 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 27,72 | 13,86 | 32 | 390625, 1, ... | 16, 1, ... |
| 4 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 38,13 | 19,06 | 32 | 195313, 1, ... | 16, 1, ... |
| 5 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 39,39 | 19,69 | 32 | 97657, 1, ... | 16, 1, ... |
| 6 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 39,88 | 19,94 | 32 | 48829, 1, ... | 16, 1, ... |
| 7 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 39,89 | 19,95 | 32 | 24415, 1, ... | 16, 1, ... |
| 8 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,30 | 20,65 | 32 | 12208, 1, ... | 16, 1, ... |
| 9 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,52 | 20,76 | 32 | 6104, 1, ... | 16, 1, ... |
| 10 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 42,13 | 21,07 | 32 | 3052, 1, ... | 16, 1, ... |
| 11 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,60 | 20,80 | 32 | 1526, 1, ... | 16, 1, ... |
| 12 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,94 | 20,97 | 32 | 763, 1, ... | 16, 1, ... |
| 13 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 47,82 | 23,91 | 32 | 382, 1, ... | 16, 1, ... |
| 14 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 43,32 | 21,66 | 32 | 191, 1, ... | 16, 1, ... |
| 15 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 28,29 | 14,15 | 32 | 96, 1, ... | 16, 1, ... |
| 16 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 28,81 | 14,40 | 32 | 48, 1, ... | 16, 1, ... |
| 17 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 29,56 | 14,78 | 32 | 24, 1, ... | 16, 1, ... |
| 18 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 30,74 | 15,37 | 32 | 12, 1, ... | 16, 1, ... |
| 19 | 62.50 | mergeSort_coordin... | mergeSort_coordin... | 31,93 | 19,96 | 32 | 6, 1, ... | 16, 1, ... |
| 20 | 81.25 | mergeSort_coordin... | mergeSort_coordin... | 33,45 | 27,18 | 32 | 3, 1, ... | 16, 1, ... |
| 21 | 87.50 | mergeSort_coordin... | mergeSort_coordin... | 38,67 | 33,83 | 32 | 2, 1, ... | 16, 1, ... |
| 22 | 93.75 | mergeSort_coordin... | mergeSort_coordin... | 40,94 | 38,38 | 32 | 1, 1, ... | 16, 1, ... |

- Có sự khác biệt rõ ràng so với TH1, các lần gọi kernel đều được tăng tốc và cải thiện thời gian tốt
- Từ kernel 0 → 18, khả năng **Speed up** tăng **50 %** sau mỗi kernel (đều nhau), **Runtime Improvement** tăng dần chứng tỏ các kernel được tối ưu tốt hơn, tận dụng tốt tài nguyên trên SM, không có hiện tượng bị nghẽn hiệu suất do giới hạn hay chia tài nguyên không đều
- Từ kernel 18 trở đi, tăng tốc nhanh hơn nữa (~**93.75 %** đối với kernel 22 trở đi)

# THỰC THI VÀ SO SÁNH

## Hiệu suất kernel: Launch Statistics (Thống kê khởi chạy từng kernel)

| ID | Estimated Speedup [%] | Function Name | Demangled Name | Duration [ms] (964,64 ms) | Runtime Improvement [ms] (596,46 ms) | # Registers [register/thread | Grid Size | Block Size [block] |
|---|---|---|---|---|---|---|---|---|
| 0 | 50.00 | mergeSort_coordi... | mergeSort_coordi... | 27,77 | 13,89 | 32 | 3125000, 1, ... | 16, 1, ... |
| 1 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 13,99 | 6,99 | 32 | 1562500, 1, ... | 16, 1, ... |
| 2 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 22,05 | 11,02 | 32 | 781250, 1, ... | 16, 1, ... |
| 3 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 27,72 | 13,86 | 32 | 390625, 1, ... | 16, 1, ... |
| 4 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 38,13 | 19,06 | 32 | 195313, 1, ... | 16, 1, ... |
| 5 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 39,39 | 19,69 | 32 | 97657, 1, ... | 16, 1, ... |
| 6 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 39,88 | 19,94 | 32 | 48829, 1, ... | 16, 1, ... |
| 7 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 39,89 | 19,95 | 32 | 24415, 1, ... | 16, 1, ... |
| 8 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,30 | 20,65 | 32 | 12208, 1, ... | 16, 1, ... |
| 9 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,52 | 20,76 | 32 | 6104, 1, ... | 16, 1, ... |
| 10 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 42,13 | 21,07 | 32 | 3052, 1, ... | 16, 1, ... |
| 11 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,60 | 20,80 | 32 | 1526, 1, ... | 16, 1, ... |
| 12 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 41,94 | 20,97 | 32 | 763, 1, ... | 16, 1, ... |
| 13 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 47,82 | 23,91 | 32 | 382, 1, ... | 16, 1, ... |
| 14 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 43,32 | 21,66 | 32 | 191, 1, ... | 16, 1, ... |
| 15 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 28,29 | 14,15 | 32 | 96, 1, ... | 16, 1, ... |
| 16 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 28,81 | 14,40 | 32 | 48, 1, ... | 16, 1, ... |
| 17 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 29,56 | 14,78 | 32 | 24, 1, ... | 16, 1, ... |
| 18 | 50.00 | mergeSort_coordin... | mergeSort_coordin... | 30,74 | 15,37 | 32 | 12, 1, ... | 16, 1, ... |
| 19 | 62.50 | mergeSort_coordin... | mergeSort_coordin... | 31,93 | 19,96 | 32 | 6, 1, ... | 16, 1, ... |
| 20 | 81.25 | mergeSort_coordin... | mergeSort_coordin... | 33,45 | 27,18 | 32 | 3, 1, ... | 16, 1, ... |
| 21 | 87.50 | mergeSort_coordin... | mergeSort_coordin... | 38,67 | 33,83 | 32 | 2, 1, ... | 16, 1, ... |
| 22 | 93.75 | mergeSort_coordin... | mergeSort_coordin... | 40,94 | 38,38 | 32 | 1, 1, ... | 16, 1, ... |

**Launch Statistics**

Summary of the configuration used to launch the kern the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

| | | | |
|---|---|---|---|
| Grid Size | 3.125.000 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | 16 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | 50.000.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | 12.207,03 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | 0 | Stack Size | 1.024 |
| # SMs [SM] | 16 | # TPCs | 8 |
| Enabled TPC IDs | all | - | - |

Số blocks sinh ra từ lần gọi kernel đầu tiên

Số threads sinh ra từ lần gọi kernel đầu tiên

# THỰC THI VÀ SO SÁNH

## Hiệu suất kernel: Launch Statistics (Thống kê khởi chạy từng kernel)

| | | | | |
|---|---|---|---|---|
| Grid Size | Số block sinh ra trong lầnmerge thứ 2 | 1.562.500 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 16 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | Số threads sinh ra trong lần merge thứ 2 | 25.000.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 6.103,52 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

| | | | | |
|---|---|---|---|---|
| Grid Size | Lần gọi thứ 3 | 781.250 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 16 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | | 12.500.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 3.051,76 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

| | | | | |
|---|---|---|---|---|
| Grid Size | Lần gọi thứ 4 | 390.625 | Function Cache Configuration | CachePreferNone |
| Registers Per Thread [register/thread] | | 32 | Static Shared Memory Per Block [byte/block] | 0 |
| Block Size | | 16 | Dynamic Shared Memory Per Block [byte/block] | 0 |
| Threads [thread] | | 6.250.000 | Driver Shared Memory Per Block [Kbyte/block] | 1,02 |
| Waves Per SM | | 1.525,88 | Shared Memory Configuration Size [Kbyte] | 16,38 |
| Uses Green Context | | 0 | Stack Size | 1.024 |
| # SMs [SM] | | 16 | # TPCs | 8 |
| Enabled TPC IDs | | all | - | - |

- Số threads sinh ra là giống nhau so với TH1 do số lần merge là giống nhau nhưng khác nhau về số block trong mỗi grid, điều này ảnh hưởng lớn đến hiệu suất tổng thể.

Đánh giá hiệu suất tổng thể cho thấy:

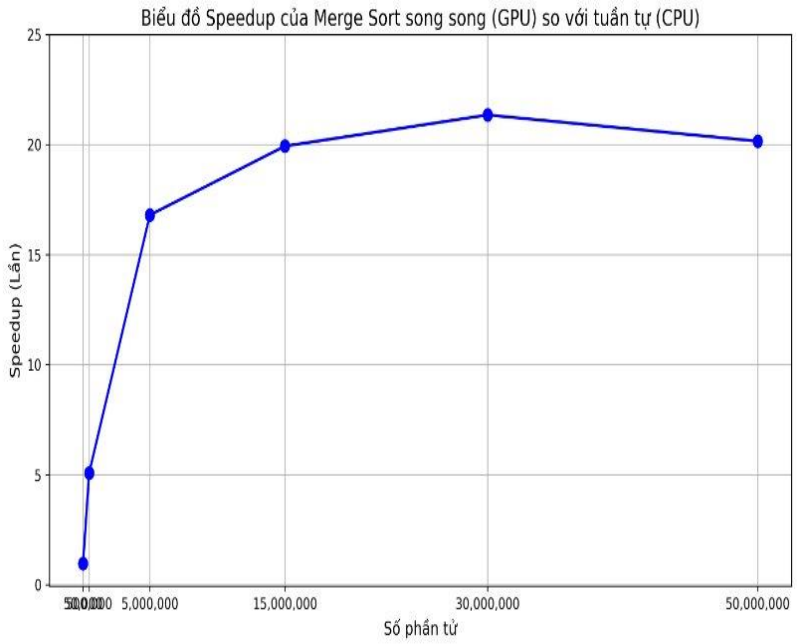
- Bằng việc giảm số thread mỗi block, ngoài việc tăng tài nguyên cho mỗi thread, ta active được lượng lớn block trên 1 SM, ép cho GPU active tối đa phần cứng đem lại kết quả tốt

# THỰC THI VÀ SO SÁNH

**Nhận xét kết quả:**

- Tốc độ Speedup: $$\text{Speedup} = \frac{\text{Thời gian chạy trên CPU (tuần tự)}}{\text{Thời gian chạy trên GPU (song song)}}$$

| Phần tử | Merge Sort CPU (Tuần tự) | Bottom-Up Merge Sort (CUDA) | Số lần Speedup | Lượng threads (lớn nhất) |
|---|---|---|---|---|
| 100 000 | **0.0123999914** | **0.0128979390** | **0.961** | **50 000** |
| 1 000 000 | **0.1331333478** | **0.0262354093** | **5.075** | **500 000** |
| 10 000 000 | **1.3413999716** | **0.0798213343** | **16.800** | **5 000 000** |
| 30 000 000 | **4.4540000121** | **0.2235130941** | **19.927** | **15 000 000** |
| 60 000 000 | **9.6621333440** | **0.4528704224** | **21.343** | **30 000 000** |
| 100 000 000 | **15.170200300** | **0.7526189901** | **20.150** | **50 000 000** |




- Speedup lên đến **21.343** lần, điều này chứng tỏ sự phân bố thread tối ưu hơn TH1
- Cấu hình này tạo ra nhiều block hơn nhưng nhẹ hơn, dẫn đến chi phí tạo kernel con thấp hơn, tăng hiệu quả xử lý.
- Ổn định hơn đến 60 triệu phần tử, chỉ giảm nhẹ với 100 triệu phần tử

# THỰC THI VÀ SO SÁNH


**Biểu đồ so sánh giữa tuần tự (CPU), song song bằng Omp Task và song song trên CUDA**

# ĐÁNH GIÁ & KẾT LUẬN

Lý thuyết cho rằng nên để số threads theo bội số của 32 để dễ dàng chia theo **warps** (do 32 threads = 1 warps) nhưng thực tế chương trình khi chạy với **16 threads** mỗi **block** cho **kernel cha** thì tốc độ kernel và tốc độ chương trình lại nhanh hơn !

Lí do nghiệm được:

- Do kernel **mergeSort_coordinate()** chỉ là kernel nhỏ, có nhiệm vụ điều phối quá trình merge, chủ yếu truy cập memory, nên việc để thread nhỏ giúp tăng số block có thể hoạt động đồng thời, tiết kiệm chi phí/block, giảm **workload** (tải công việc) để tăng khả năng ẩn trễ khi nhập xuất memory.

Cùng với **1024 threads/block** (quá cao) ở **kernel con** cũng không tối ưu tốt bằng việc để **256 threads/blocks**

Lí do nghiệm được:

- Với mergeSort song song, mỗi thread phải xử lý nhiều việc (như gọi kernel con), dùng số lượng nhỏ thread nhưng làm việc nặng sẽ hiệu quả hơn → Giảm overhead khi đồng bộ giữa các threads sau mỗi lần lặp, mỗi thread có nhiều tài nguyên, cho phép GPU chạy nhiều block cùng lúc, giúp ẩn độ trễ tốt
- Nếu càng nhiều threads/block thì nhiều threads phải đồng bộ thường xuyên, gây ra chi phí đồng bộ cao, chậm chương trình,
- Mỗi SM có lượng tài nguyên giới hạn. Nếu tăng số lượng thread lớn (như 1024 threads/block ) thì mỗi thread lại được cấp ít tài nguyên hơn, dẫn đến giảm hiệu năng.

# EM XIN CẢM ƠN !