Report – Project 2 (Performance Analysis of Sorting Algorithms)

1. Introduction

This report looks at how two sorting methods (Merge Sort and Quick Sort) perform when run on three types of processors: a regular CPU, a parallel CPU, and a GPU. It tests them using three types of data (already sorted, completely random, and sorted in reverse) and with different amounts of data (2 million, 4 million, and 8 million items). The report also compares how these tests perform on both Windows and macOS to see how the operating system affects the results.

2. Device Information

i. Windows

- CPU: Intel Core i5-8300H
- RAM: 32 GB
- OS: Windows 11
- Cores: 4
- Logical Cores: 8
- GPU: Nvidia GTX 1050

ii. MacBook Pro

- M3 Pro Chip
- RAM: 36GB Unified Memory
- OS: Sequoia
- Cores: 12 (6 performance and 6 efficiency)

3. Methodology

The sorting tests were done using these settings:

- **Algorithms:** Merge Sort and Quick Sort
- **Processors:** Regular CPU, Parallel CPU, and GPU
- **Data Types:** Arrays that were already sorted, completely random, or sorted in reverse
- **Array Sizes:** 2 million, 4 million, and 8 million elements
- **Operating Systems:** Windows and macOS (using OpenMP for parallel processing)

All the sorting tests worked correctly, and the arrays ended up properly sorted, as shown in the "isSorted" column

Note: The **helper_functions.h** need to be within the current directory, as compilation errors will occur. OpenMP program will work for even up to 90M, but Cuda will not work till 40M.

© 2025 Sagar Patel. All rights reserved. Do not copy or redistribute without permission.

4. Key Findings

Processing Unit Performance

i. CPU vs GPU Performance:

- The CPU (both regular and parallel) was always faster than the GPU in every test.
- The parallel CPU was the fastest overall on both Windows and macOS.

ii. Parallel vs Sequential CPU:

- The parallel CPU was much faster than the regular one—between 2 to 5 times faster, depending on the data and the platform.
- On macOS, the speed boost from parallel processing was even better in some cases compared to Windows

iii. GPU Performance:

- The GPU was always slower than both CPU types. This might be because the sorting algorithms don't work well on GPUs, or because moving data between the CPU and GPU takes too much time for arrays of this size.

iv. Windows vs MacBook

- The macOS version using OpenMP was especially efficient, particularly when using Quick Sort on already sorted data.
- Windows showed similar performance patterns, but the exact times were different.

4. Detailed Results

A. 2,000,000 Elements

WINDOWS

Algorithm	Dataset	Sequential CPU (s)	Parallel CPU (s)	GPU (s)
Merge Sort	Sorted	0.397	0.194	0.877
Quick Sort	Sorted	0.044	0.016	1.362
Merge Sort	Random	0.596	0.193	0.847
Quick Sort	Random	0.858	0.381	1.396
Merge Sort	Reverse-Sorted	0.421	0.154	0.706
Quick Sort	Reverse-Sorted	0.083	0.040	1.358

MacOS

Algorithm	Dataset	Sequential CPU (s)	Parallel CPU (s)
Merge Sort	Sorted	0.209	0.042
Quick Sort	Sorted	0.016	0.004
Merge Sort	Random	0.357	0.076

© 2025 Sagar Patel. All rights reserved. Do not copy or redistribute without permission.

Quick Sort	Random	0.495	0.141
Merge Sort	Reverse-Sorted	0.203	0.043
Quick Sort	Reverse-Sorted	0.034	0.014

B. 4,000,000 Elements

WINDOWS

Algorithm	Dataset	Sequential CPU (s)	Parallel CPU (s)	GPU (s)
Merge Sort	Sorted	0.826	0.301	1.436
Quick Sort	Sorted	0.093	0.032	2.570
Merge Sort	Random	1.150	0.408	1.735
Quick Sort	Random	1.550	0.894	2.429
Merge Sort	Reverse-Sorted	0.834	0.312	1.421
Quick Sort	Reverse-Sorted	0.177	0.079	2.598

MacOS

Algorithm	Dataset	Sequential CPU (s)	Parallel CPU (s)
Merge Sort	Sorted	0.416	0.081
Quick Sort	Sorted	0.037	0.009
Merge Sort	Random	0.787	0.155
Quick Sort	Random	1.036	0.244
Merge Sort	Reverse-Sorted	0.423	0.083
Quick Sort	Reverse-Sorted	0.076	0.028

C. 8,000,000 Elements

WINDOWS

Algorithm	Dataset	Sequential CPU (s)	Parallel CPU (s)	GPU (s)
Merge Sort	Sorted	1.689	0.664	2.916
Quick Sort	Sorted	0.207	0.171	6.164
Merge Sort	Random	2.413	0.805	3.486
Quick Sort	Random	2.820	1.221	5.277
Merge Sort	Reverse-Sorted	1.697	0.658	2.905
Quick Sort	Reverse-Sorted	0.387	0.171	6.113

MacOS

© 2025 Sagar Patel. All rights reserved. Do not copy or redistribute without permission.

Algorithm	Dataset	Sequential CPU (s)	Parallel CPU (s)
Merge Sort	Sorted	0.839	0.169
Quick Sort	Sorted	0.079	0.020
Merge Sort	Random	1.622	0.326
Quick Sort	Random	2.075	0.744
Merge Sort	Reverse-Sorted	0.835	0.170
Quick Sort	Reverse-Sorted	0.166	0.059

5. Performance Ratios

Windows - Speed Improvement: Parallel CPU vs Sequential CPU

- Merge Sort: 2.0x - 3.1x faster
- Quick Sort: 2.1x - 2.8x faster

macOS - Speed Improvement: Parallel CPU vs Sequential CPU

- Merge Sort: 4.0x - 5.0x faster
- Quick Sort: 2.8x - 4.0x faster

Windows - Speed Comparison: GPU vs Fastest CPU (Parallel)

- Merge Sort: 3.6x - 4.8x slower
- Quick Sort: 4.3x - 85.1x slower

Cross-Platform Comparison: macOS Parallel vs Windows Parallel

- Merge Sort: 2.0x - 4.6x faster on macOS
- Quick Sort: 1.6x - 8.6x faster on macOS

6. Conclusion

The results show that using a parallel CPU is the best option for sorting in all test cases on both Windows and macOS. Quick Sort with parallel CPU was the fastest for sorted and reverse-sorted data, while Merge Sort gave more steady results across all types of data. The GPU did not perform better than the CPUs for these tests. This means that, for sorting tasks like these, it's better to improve CPU parallel processing rather than using a GPU. macOS showed better parallel processing than Windows, with faster speeds and higher performance boosts. This means that the operating system can also make a big difference in how well parallel sorting works.

According to the implemented algorithm, assume someone wants to perform sorting for data preprocessing using these implementations, I would prefer or suggest going as:

- Quick Sort, for pre-sorted or reverse-sorted using OpenMP
- Merge Sort, for randomly sorted using OpenMP

Still, there the Cuda code needs to be optimized for both algorithms. Therefore, I will take into consideration using such a scenario.

© 2025 Sagar Patel. All rights reserved. Do not copy or redistribute without permission.