Parallel Computing Assignment

Professor: Ferrandi Fabrizio
Teaching Assistant: Curzel Serena

Students

Capodanno Mario

Grillo Valerio

Lovino Emanuele

A. Y. 2024-2025

Chapter 1

Introduction

1.1 Challenge Description

The main goal of the challenge was to parallelize the given mergesort implementation, using the openMP APIs described during the lectures.

1.2 Experimental Setup

The experimental setup consists of a machine equipped with an AMD Ryzen 5 5600U, a processor with 6 physical cores and 12 threads. The openMP-based Merge Sort algorithm was compiled with GCC with the -03 optimization flag and hosted on an Ubuntu 24.01 Linux system.

1.3 Performance measurements

All runs were conducted on a maximum of 12 threads, matching the total number of hardware threads available. The parallel implementation was tested on arrays ranging from 1 million to 100 million elements as these sizes exploit better the parallel paradigm. Tests were performed also on other sizes, whose results were necessary to better evaluate correct and optimal cut-off parameters.

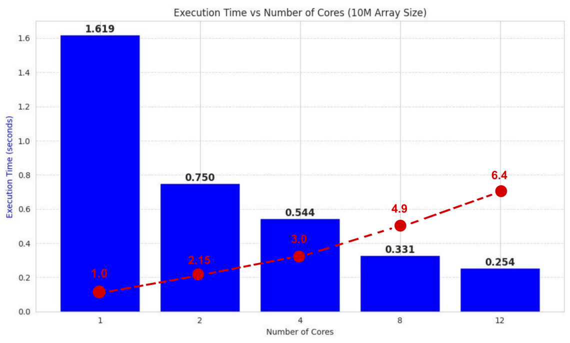

Figure 1.1: threadcore

Threads	**Execution Time (sec)**	**Speedup**
Array Size: 100,000,000		
1	12.504	1.00
2	7.112	1.75
4	6.149	2.03
8	3.198	3.90
12	2.630	4.57
Array Size: 10,000,000		
1	1.619	1.00
2	0.750	2.15
4	0.544	3.00
8	0.331	4.90
12	0.254	6.37
Array Size: 1,000,000		
1	0.141	1.00
2	0.065	2.10
4	0.055	2.56
8	0.031	4.54
12	0.0223	6.32

Table 1.1: Execution Time and Speedup for Different Thread Counts and Array Sizes

1.4 Design choices

This section analyzes the improvements made to the recursive division and merging processes within the sorting algorithm, exploiting OpenMP for parallel execution.

1.4.1 MsPartition()

The recursive division phase has been enhanced to achieve parallelism through the use of OpenMP's `taskwait` clause. By introducing a sort cut-off mechanism, the algorithm dynamically switches to a sequential partition when the array size is significantly smaller than the original input. This approach minimizes the overhead associated with task creation and improving performance. Additionally, transitioning from sequential Merge-Sort to `std::sort` for very small arrays has resulted in a notable 25% reduction in execution time.

1.4.2 MsMergeParallel()

Similarly, the merging phase has been optimized by implementing parallel recursive calls. Load balancing and binary search techniques have been employed to further enhance performance, achieving a 40% reduction in execution time. A merge cut-off mechanism ensures that the algorithm goes back to the sequential case when appropriate, mirroring the strategy used in the division phase.