High Performance and Parallell Computing
Assignment 3

Björn Mosten

Contents

Chapter 1

1.2 Introduction

The goal of this project was to create a well-performing quick-sort process that scales well with multiple threads. It was written in C and utilizes the OpenMP API for parallelization.

1.2.1 Hardware

CPU: AMD Ryzen 7 3700X 8-Core Processor, with a maximum clock speed of 3.6 GHz. It has 256 KB of L1i-cache, 256 KB of L1d-cache, 4 MB of L2-cache and 32 MB of L3-cache. It has 8 cores and 2 threads per core. RAM: 32 GB of DDR4 RAM.

1.3 Solution

1.3.1 Data Structures

Each thread is given a struct containing the following data:

- `int * data` - The data to be sorted.
- `int * data_free` - A pointer to the start of the data array. This is often, but not always, the same as *data*.
- `int * new_data` - Data array to be used when merging. Combines the data from *tmp_data* from another thread and *data* from the current thread.
- `int * tmp_data` - Temporary data array containing the data that is to be merged into another thread.
- `int data_n` - The number of elements in the data array.
- `int tmp_data_n` - The number of elements in the tmp_data array.

1.3.2 Algorithm

Each thread is given an equal amount of data to sort. Initially, each thread sorts its own data using the default C qsort function. All threads are put into a single group and common pivot point is calculated by taking the average of the median values of all threads in each group.

1.4 Optimization Techniques

1.4.1 Parallelization

Parallelization was done with OpenMP.
Each thread is given 1/N of the data to sort, where N is the number of threads. Initially, each thread sorts its own data using the default C qsort function. A group of N threads is created and each thread is given a 'partner' in said group. Each pair consists of one 'lower' part and one 'upper' part. Each thread calculates the average of the medians in its given group. This calculation may not be necessary, but by calculating it in each threads

allows us to remove the overhead of having to communicate the median values between threads. Each thread then partitions its data into two parts, one part containing all values lower than the average of the medians and one part containing all values higher than the average of the medians. In each pair of threads, the thread with a higher thread id is given the upper part part of its partner thread, and the thread with a lower thread id is given the lower part of its partner thread.

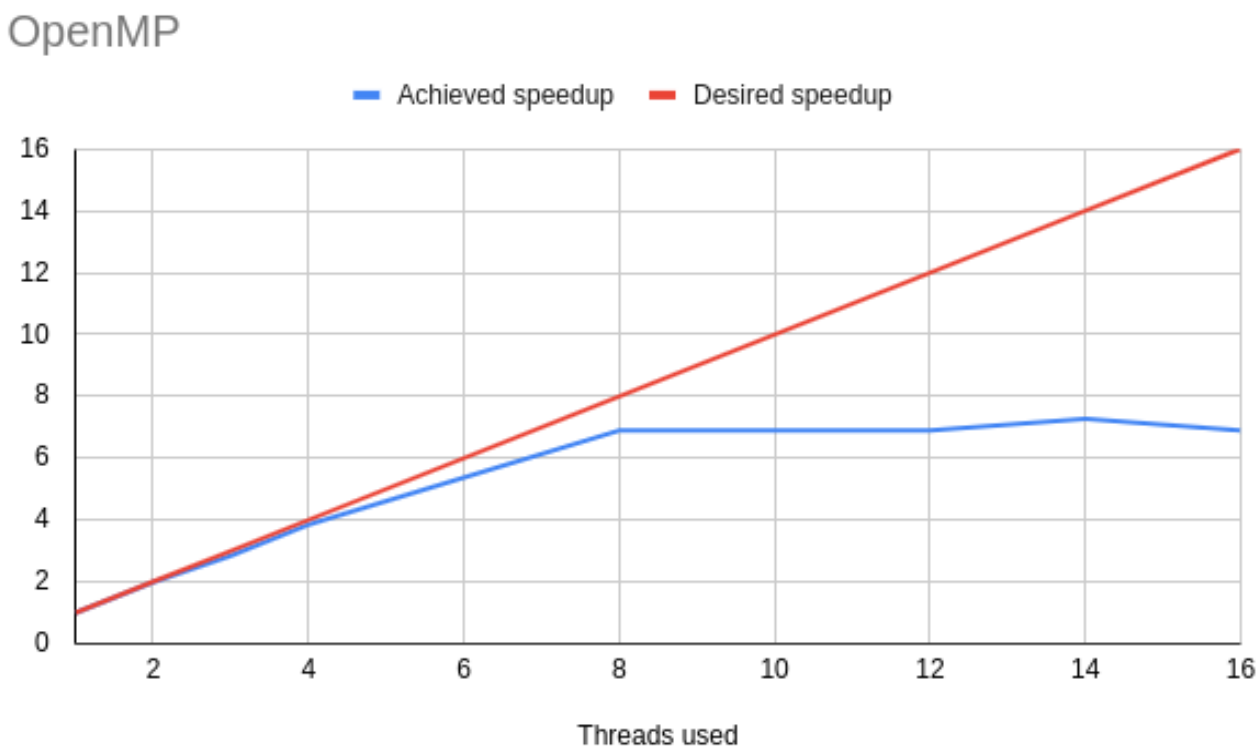
1.4.2 Loop Unrolling

1.5 Compilator

GCC 11 was used for compilation with the following flags:

```
-Ofast
-flto
-ftree-vectorize
-march=native
-mtune=native
-fopenmp
-funroll-loops
-mno-vzeroupper
-fno-trapping-math
-fno-signaling-nans
-funsafe-math-optimizations
```

1.6 End Results



Bibliography