Universiteit van Amsterdam

Assignment 1

POSIX Threads

November 7, 2014

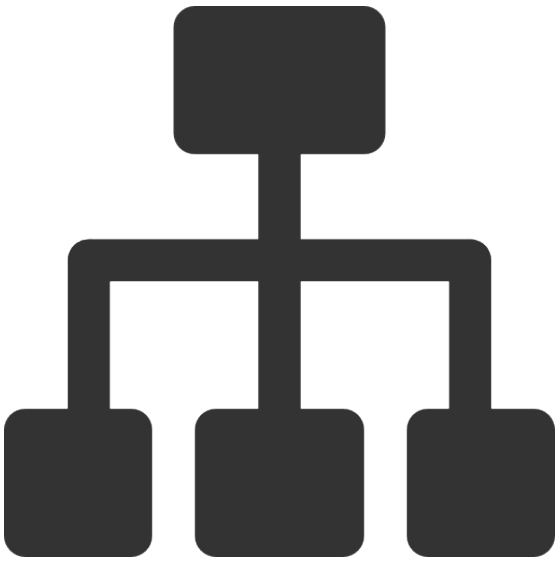
Student:
Robin Klusman
10675671

Maico Timmerman
10542590

Supervisor:
Dr. A. Pimentel

Course:
Concurrency and Parallel Programming

Course code:
5062COPP6Y



Contents

1 Introduction

For this assignment a parallel programming solution needs to be implemented for two problems, a wave equation simulation and the Sieve of Eratosthenes. For the wave simulation the user can specify the amount of wave amplitude points, the amount of steps it needs to simulate and the desired amount of threads. The program then calculates all the wave values until it has done the specified amount of steps.
For the Sieve of Eratosthenes assignment the program will use the Sieve of Eratosthenes algorithm to continuously produce prime numbers until a certain amount of primes have been found. It will do this using multiple threads, thus utilising parallelism.

2 Method

2.1 Wave Equation Simulation

First the specified amount of threads need to be created, these threads will then all start executing their routine. This routine gets a certain range of amplitude values that it is allowed to calculate, this range is determined by dividing the amount of amplitude points by the amount of threads, so that each thread has the same workload. When a thread finishes its calculations it locks and decrements the variable that is keeping track of the amount of busy threads. The lock makes sure no two threads try to decrement it at the same time, causing it to never reach 0. If this variable reaches 0, the thread that made it 0 will perform the buffer swap and broadcast a signal to all waiting threads that they can continue their work. When the specified amount of time steps has been completed, the threads will all terminate and the final wave is returned.

2.2 Sieve of Eratosthenes

To find prime numbers, the Sieve of Eratosthenes algorithm is used. This multi-threaded program uses this algorithm to find primes by filtering out non-primes from a constant flow of natural numbers, $n \in \mathbb{N}$, that is generated by the generator thread. In this implementation we actually opted to skip every even

number, since those are all multiples of 2 and would thus be filtered out anyway. This will improve the performance of the program a little because only half as many numbers need to be processed by the first filter and one less filter is needed. The generated numbers are put into a queue of a fixed size, from which the next filter will grab its input. If a number reached the end of the filter chain, that means it has to be a prime, since its not a multiple of any previously encountered number. This number will then be printed, and a new filter thread will be created to filter for multiples of this newly found prime.
The queues used to pass numbers between different threads are of a set size, so that no starvation can occur for threads further down the chain. Reading and writing to and from the queues is done mutually exclusive, so that no race conditions can exist. A thread will first lock the input buffer to see if there is anything in there. If there is, it will grab the first value and unlock the input. Then the value is processed and if it is not filtered out, the output buffer is locked to write this value to it. If there is no space in the output buffer, or if the input buffer is empty, the thread will wait for a signal before it can continue.

3 Results

3.1 Wave Equation Simulation

Running the Wave Simulation on the DAS4 machine with different values for the total amount of amplitude points and also varying the amount of threads used, but keeping the amount of time steps the same at 10000. In the graph below the results are shown, dividing the time it took to run that particular simulation on one thread by the time it took to run it on two, four, six and eight threads.

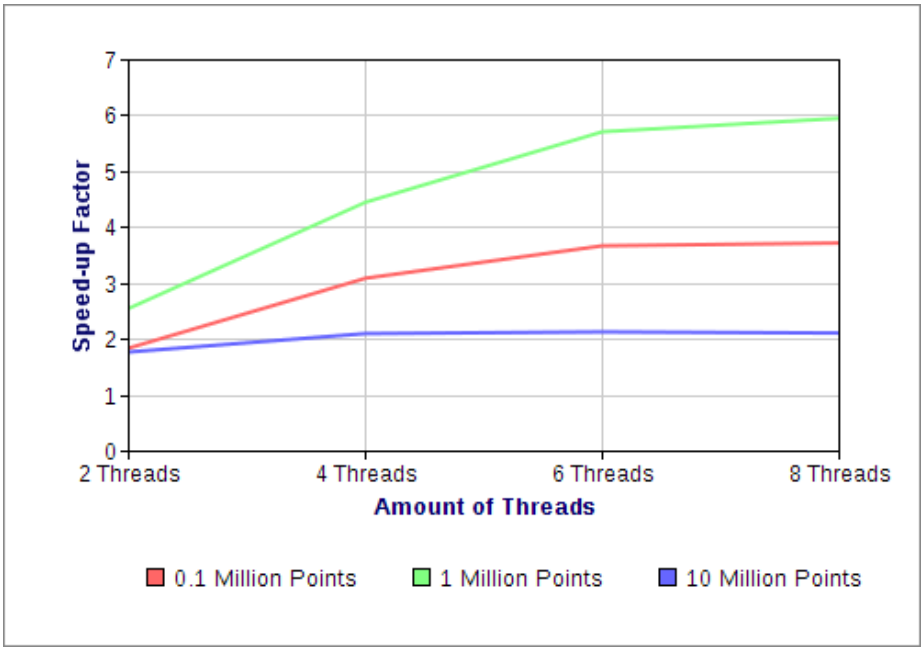


Figure 1: Speed-up factor of Wave Simulations with different amount of total amplitude points. Comparing multiple thread to single thread runtime. Running a total of 10000 time steps.

3.2 Sieve of Eratosthenes

4 Discussion

4.1 Wave Equation Simulation

It is interesting to see that when running the Wave Simulation with one million amplitude points gives a significantly higher speed-up than when it is run with a hundred thousand or ten million. In fact, when going from six to eight cores in the ten million points simulation, the speed-up factor actually slightly drops. This overhead has to be created then by the synchronization at the end of each time step. Since the program now has to deal with rather large quantities of data.
The reason for the lower amount of amplitude points to also have a lower speed-up factor can be explained by the fact that running it with one thread is relatively fast due to the smaller amount of computations needed. Thus the speed-up achieved is also lower.

4.2 Sieve of Eratosthenes