Smarandache-Wellin Numbers

Nagy Lilla

06.05.2019

1 Introduction

A Smarandache-Wellin number is the concatenation of the first n prime numbers. A Smarandache-Wellin prime number is a Smarandache-Wellin number that is also a prime.
The Smarandache-Wellin numbers look like this: 2, 23, 235, 2357, ... Out of these numbers, the Smarandache-Wellin primes are 2, 23, 2357 and the fourth one has 355 digits, the result of concatenating the first 128 prime numbers, ending with 719.

2 Solution

The algorithm is pretty simple and fast, because the only operation we need before checking that the number is prime is a simple concatenation, therefore it reaches big numbers fast. This induces a new problem: big numbers. The data type long long was not enough, because the program would end in seconds, and the search for a big number library had to take into consideration the fact that this library has to be compatible with C and C++ as well, the other language I chose, otherwise the measurements would not have been consistent. For Prolog, there is no such problem, because it handles big numbers without any issues.
After lots of trial and error, I decided to use the GMP library for big numbers, because it's compatible with C and C++ as well, and it is one of the most popular big number libraries, so the documentation was pretty good.
To make the solution compatible with multithreading, I first generate the prime numbers until 2000 (or 10000 for Prolog), then I concatenate them and I execute a for loop for every thread, which checks whether these numbers are primes or not. If so, they are printed to the screen.
I measured the time after the numbers have been concatenated, because it works very fast, so I thought I could even have a look-up table or anything similar, and the real power of calculation lays in the prime number algorithm. The primes are actually obtained once in C, C++ and OpenMP, and are used for every run with a different number of threads.
For checking whether these big numbers are primes, I have used built-in probabilistic prime number algorithms that check whether a number is prime or not

with a certain probability. These are pretty accurate and are much faster than the average prime number algorithm, therefore are suitable for big numbers and give results with a pretty good approximation. The probabilistic feature can be seen the most in Prolog, as it doesn't always give the correct numbers, which is avoided in the GMP library with the return values being different for definitely prime and probably prime numbers. This way, 2, 23 and 2357 are definitely prime in GMP, but Prolog's algorithm doesn't always consider them like that.
When I used OpenMP, I used a parallel for to do the multithreading, which solved the problem of creating threads.In Prolog, the algorithm was much faster, so I needed more numbers to reach a similar time interval as I needed for C or C++. This might be also because in Prolog there is no need for an extra library for big numbers.
The parallelization is done by concatenating all the numbers separately before checking whether they are prime or not. This is valid for any solution. Then, each thread takes every i-th number from the array, where i is the index of the thread, starting from 0 to n - 1, where n is the number of threads, except for Prolog, where I send the solutions with sending the message in a message queue and load balancing is done automatically. In C and C++, this results in load balancing, whereas in OpenMP this is done by simply using a parallel for.

3 Results

The C++ and OpenMP solutions don't have a speedup, they actually become slower with every new thread. However, the solution written with OpenMP performs better than the one using C, because there is a parallel for which is already better optimized than the one done manually.
The C++ solution performs much better, it does have a speedup, but the best result is given by Prolog, which was also expected, because it is very well optimized and it doesn't require a big number library like all the other solutions do.
In conclusion, the best speedup is given by Prolog, with a maximum speedup of 1.88, at 2 threads, because my laptop has two physical CPUs, so almost reaching a 2 times speedup is a pretty good result.