Oblig3 IN3030: Prime numbers

**Introduction**

This report is about the different versions of Sieve of Eratosthenes, one parallel and one sequential. There is also a parallel and sequential version of factorization of a large number. The Sieve gives us all the prime numbers up to n, and then the factorization will give us all the prime factors for every number from n\*n-1 to n\*n-100. This report will tell a bit about how I have implemented the different parts of the program and look at what my results are.

**User guide**

To use the program, you would need Oblig3.java, FacPara.java, ParaSieve.java, Oblig3Precode.java and SieveOfEratosthenes.java in the same folder, the last two are files we were given.

* javac Oblig3.java
* java Oblig3 {N} {number of threads}

N must be greater than 16 and if you want max possible threads for your computer just put 0 as number of threads.

**Parallel Sieve of Eratosthenes**

The code implements a parallel Sieve of Eratosthenes algorithm to find prime numbers up to n. It splits the work among multiple threads, where each thread marks non-prime numbers within its assigned range. Synchronization points ensure that each thread has completed their work before starting to count the primes. Finally, the count of primes is collected, and the prime numbers are extracted from the marked range. This approach efficiently utilizes multiple CPU cores for faster prime number generation.

**Parallel factorization of a large number**

I made a class to start the factorization and a worker class to do the threaded work. Called the function factorize() so that it could find all the numbers to be factorized and distribute the primes from the parallel sieve evenly amongst the threads and the threads could work together to factorize the large number at hand. After each thread is done the factorization is going through and ensuring that every factor is a prime number, and the factors add up to the total.

**Implementation**

When running the program, it starts off with several checks that the user has put in the necessary data to be able to run the program. After each of the checks are passed, we then go to the sequential sieve then the parallel sieve and then the sequential factorization and lastly the parallel factorization. After the sieves I am checking if I get all the same primes in both and after the factorization, I have implemented a check to see if each of the precode objects is the same. For every part of the code that either uses the sieve or factorizes it is run 7 times in a loop to warmup the JVM. Each run is timed and stored in an array and after the part is done, I find the median value of and print it out for the user. After a sequential and a parallel is done I do a speed up comparison and calculate the speedup. For the sequential sieve I am just using the code given to us by the IN3030 GitHub, as of the parallel sieve I have explained above what it does. For the sequential factorization I am running a double for-loop with an inner while loop. The outer loop finds the number to be factorized and the inner loop factorizes the number using the primes from the sieve. The while loop is here to ensure that every factor is the smallest prime number it can be, if it is not the program will find the smallest and add it as a factor. Each time a factor is found it is stored using the addFactor() function from the object Oblig3Precode. When every number is factorized the program writes all the factors to a file. Lastly the parallel factorization which I have explained above.

**Measurements**

Cores used: 12

Sieve of Eratosthenes timing:

|  |  |  |
| --- | --- | --- |
|  | Sequential | Parallel |
| N = 2 000 000 | 7.3336 ms | 7.9061 ms |
| N = 20 000 000 | 74.682 ms | 79.2979 ms |
| N = 200 000 000 | 861.4804 ms | 865.1278 ms |
| N = 2 000 000 000 | 12073.4132 ms | 28175.7453 ms |

Factorization timing:

|  |  |  |
| --- | --- | --- |
|  | Sequential | Parallel |
| N = 2 000 000 | 108.1685 ms | 49.9605 ms |
| N = 20 000 000 | 939.1046 ms | 325.8895 ms |
| N = 200 000 000 | 7983.6226 ms | 3193.4301 ms |
| N = 2 000 000 000 | 70633.1589 ms | 25463.5909 ms |

Sieve of Eratosthenes speedup:

Et bilde som inneholder tekst, skjermbilde, line, diagram

Automatisk generert beskrivelse

As we can see on the graph and in the table the parallelization, I did from last time I delivered to this time has just made the program slower. I now get alle the correct primes but the runtime I abysmal. It might be because I am dividing chunks of n instead of the primes in initially created last time. The reason I changed approach is because I could not find a way to get all the primes with the old paraSieve.

Factorization speedup:

Et bilde som inneholder tekst, skjermbilde, line, diagram

Automatisk generert beskrivelse

Luckly I am getting a speed up for alle N in my factorization and this is because of the normal advantage of using several threads instead of just one.

**Conclusion**

In conclusion I now get correct primes and the correct factors, but my program is super slow. There might be an easy fix to this, but I could not find it. Overall I am satisfied with the result but a little sad about the runtime.

**Appendix**

Output of my program at N = 2 000 000 and max possible cores (which is 12):

**Et bilde som inneholder tekst, skjermbilde, Font, programvare

Automatisk generert beskrivelse**