IN4330 Oblig 3 report

Specifications

Computing Specification used for testing

Processor is an Intel(R) Core(TM) i5-9400F , 2,90 GHZ with 6 cores and 6 logical processors

Usage

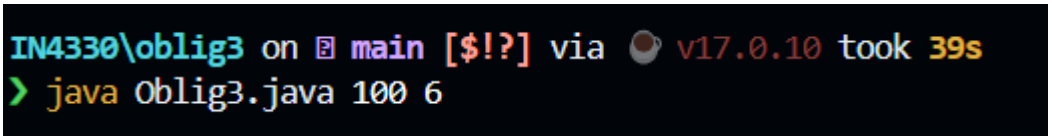
The program to be runned is the Oblig3.java which runs sieve and factorization both sequentially and in parallel, collects tests results, and finally verifies the prime numbers calculation.

General use: java Oblig3.java <n> <t>

where <n>. is any positive integer, and t is the number of threads(optional).

The second argument which controls the choice in threads to be runned under parallel sieve and factorization is optional. when this argument is not provided, the program defaults to the maximum number of available cores.

Use Case example



Implementation

Parallel sieve

For the parallel sieve, there are several choices and methods in what to parallelize. I decided to divide the traversing and marking part up into threads with equal sized work. Then using the java executor module to start up threads. In order to keep track of calculation and avoid racing a countdown latch was implemented.

Sequential Factorization

The factorization works by having a while loop that for every iteration finds the first prime number(from primesToN Array) that the inputed prime number is divisible by and divides, then adds it to the list . For example if the inputted number isn't divisible by 2 , then it attempts 3, if not divisible by 3 then 5 so on and so forth. After division we now sit with a new prime number and repeat until division is no longer possible and we should sit with a list that consists for the prime number factorization of the original inputed number.

Parallel factorization

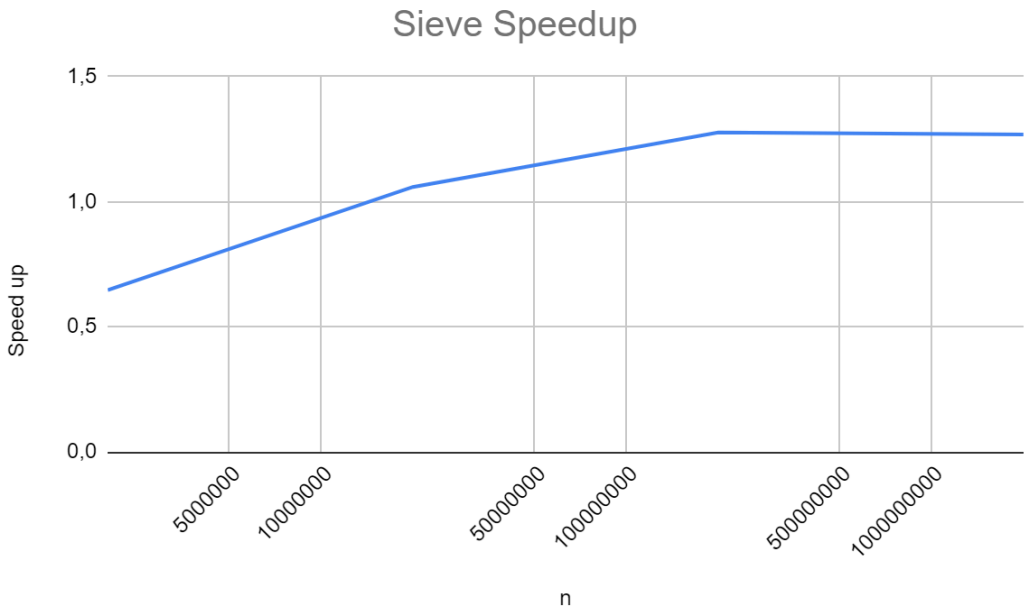
For parallel factorization a similar approach to the parallel sieve was implemented. The running of the primesToN array is divided up in chunks that each thread works on to check if prime division is possible. To avoid racing a couple things were implemented. First a concurrent linked queue, this is to avoid a form of racing that happens when for example a number is both divisible by 3 and 5 and the thread found that it was divisible by 5 before 3. Here we want the lowest number to persist so we make a que that hinders these types of errors, and the list is then filled later from the properly filled linked que. Similar to the parallel sieve java executor was used to start up the threads and a countdown latch was implemented to avoid other types of racing.

Measurements

All measurements results are a median of 7 runs for each problem. This means 7 individual runs for both the sequential and parallelized version of sieve and factorization.

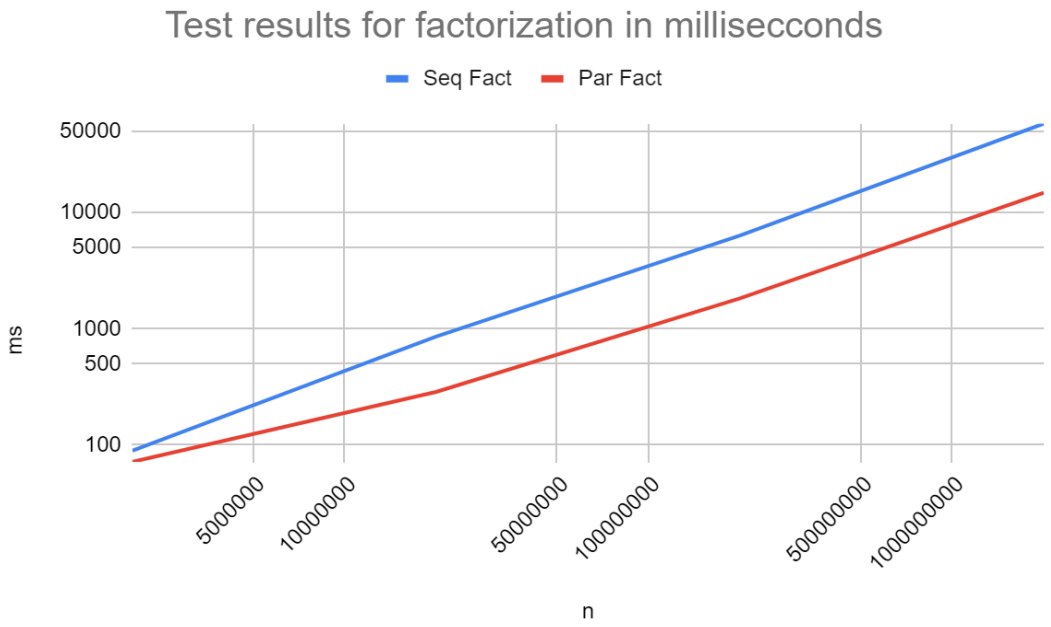
Test results for l Sieve in milliseconds		
n	Seq Sieve	Par Sieve
2000000	8,6215	13,3215
20000000	91,03	85,9905
200000000	1206,6222	944,9393
2000000000	16716,4629	13175,7261

Speed results for Sieve	
n	Speed up
2000000	0,647
20000000	1,059
200000000	1,277
2000000000	1,269

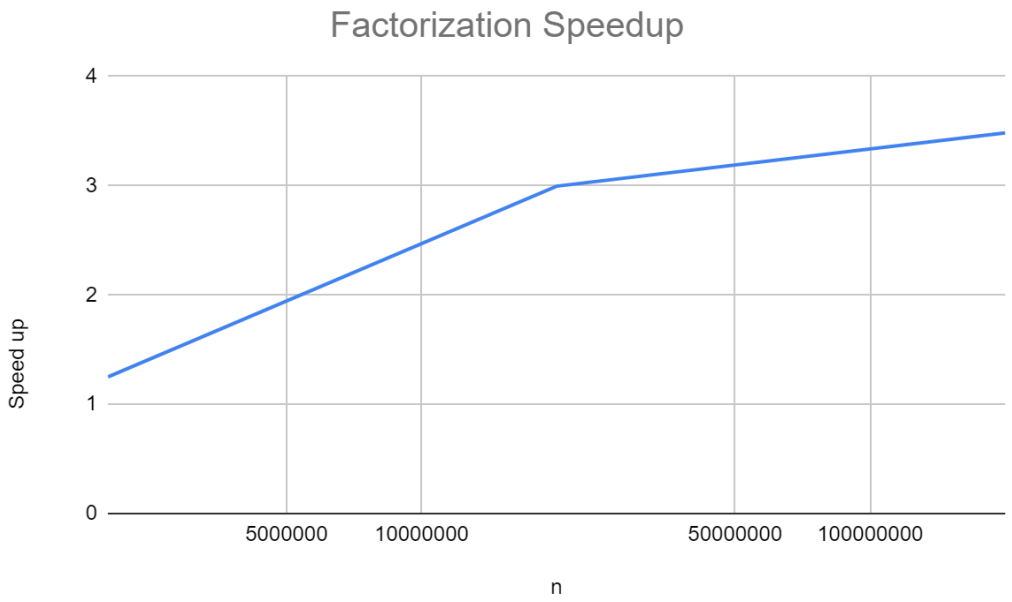


For the sieve the general trend is that the sequential comes out on top on anything below for when n is about 10-15 million. From there on the parallel solution is faster and speed up continues for every variable of n tested until 2 billion. The speed seems to come to a plataeu with speeds rarely going above the threshold of 1.3, but continuously staying in the 1.25-1.3 range regardless of increase in n.

Test results for factorization in milliseconds		
n	Seq Fact	Par Fact
2000000	89,3321	71,4067
20000000	856,1037	285,7861
200000000	6351,4148	1823,6918
2000000 000	58339,9548	14814,509



Speed results for Factorization	
n	Speed up
2000000	1,251
20000000	2,996
200000000	3,483
2000000000	3,938



For the factorization, from the get go the parallel version comes out on top and continues that trend. For every increase of n the parallel version shows better and better results, with the best results being for the 2 billion almost reaching a speed of 4. Although I don't have the resources to test higher numbers, it does seem that above 2 billion we hit a plateau, and would probably struggle to see significant improvements in results.

Appendix

```
IN4330\oblig3 on  main [$!?] via  v17.0.10
❯ java Oblig3.java 2000000000 6
Sieve sequential times for a median of 7 test run: 16716.4629ms (16.7164629s)
Sieve parallel times for a median of 7 test run: 13175.7261ms (13.1757261s)
Speed up of sieve for 7 runs: 1,269
Times for a median of 7 test run sequential factorization 58339.9548ms (58.3399548s)
Times for a median of 7 test run Parallelized factorization, using 6 threads 14814.509ms (14.814509s)
Speed up of factorize for 7 runs: 3,938
Test done for n= 2000000000 and t= 6
```