1. Introduction – what this report is about

This report is about OBLIG nr.3 in INF2440 written by Gabriell Vig (gabriecv)

2. User guide

javac *.java

java oblig3 <find primes less than> <cores to be used>

3. Parallel Sieve of Eratosthenes

- Sequential part:
  - find the primes less than $\sqrt{\sqrt{N}}$ by marking with 3, finding next prime, repeat
  - Mark numbers less than $\sqrt{N}$ with these primes
- Parallell
  - Prime part:
    - Threads traverse the marked numbers & find primes less than $\sqrt{N}$
      - Splitting byte_array into different start-end sectors for each thread
    - SEQUENTIAL: Synchronization:
      - Merge the primes found.
    - Use these primes to mark numbers less than N
      - Splitting byte_array…
    - Traverse the marked numbers and find primes less than N.
    - SEQUENTIAL: Synchronization:
      - Each thread stores these primes locally. Merge them.
      - Split the primes into different start-end sectors for each thread.

4. Parallelll Factorization

- Parallell Factorization:
  - Try to divide every big number from n*n-1 to n*n-101 with primes from your sector of primes.
    - Add prime-factor to your local Linkedlist[] (array) which contains a list of factors for each big number.
  - SEQUENTIAL: Syncrhonization
    - Divide the big numbers up into sectors..
  - Merge the factors
    - Each thread takes a colum in the array containing the arrays where the threads store the linkedlist's for each big number's factors.
    - Simpler put: Each thread merges the factors of the big numbers in their "sector", places output into a list which in the end contains 100 linkedlist with all the factors of the 100 biggest numbers.
      - While merging: Divide the big number by the factors which are stored,
        - then if it is not yet 1, try to divide again with all the factors as many times as possible with each.
        - Then if it is not yet 1, it is itself a prime.

5. Implementation

I've tried to program this with as little synchronization as possible.

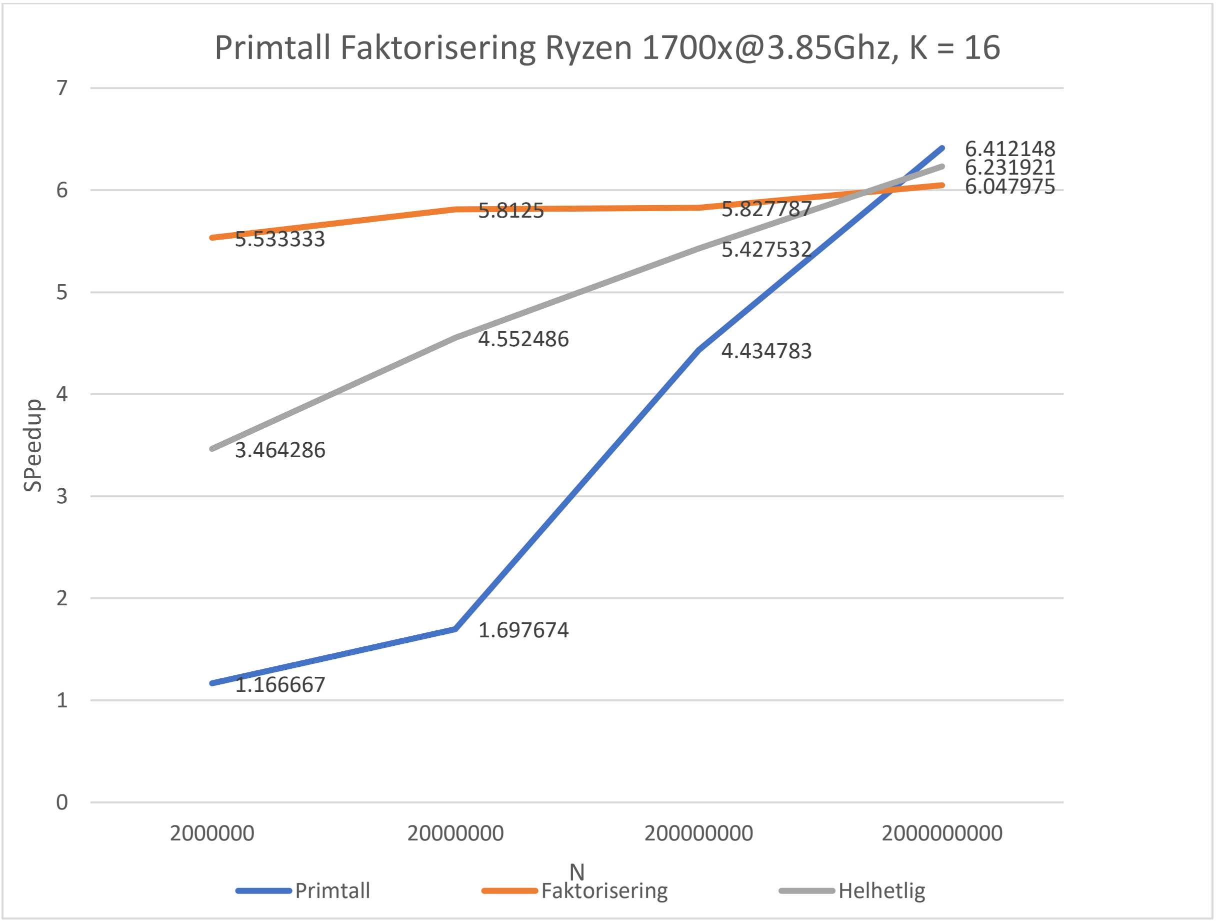
This means that I've sacrificed f.ex being able to end a factorization when all the factors have been found.

Also sacrificed by not iterating over each id'th prime, instead giving each thread a sector of primes which should efficiently use the CPU cache. Could have randomized array, would be interesting to see result.

The program asks you if you want to check if the programs give the same output(of primes), and if you want to print the 100 biggest factors. (you have to check the primes in this case).

I've also created a long version of the sequential program. You can try it if you uncomment a bit of code inside the go method in oblig3.java (ss for seq_sieve instead if iss for int_seq_sieve)

6. Measurements – includes discussion, tables, graphs of speedups for the four values of N, number of cores used.



tid i ms	Primtall			Faktorisering			Overall		
N	Seq	Para	Speedup	Seq	Para	Speedup	Seq	Para	Speedup
2000000	7	6	1.166667	83	15	5.533333	97	28	3.464286
20000000	73	43	1.697674	744	128	5.8125	824	181	4.552486
2E+08	1938	437	4.434783	7005	1202	5.827787	8950	1649	5.427532
2E+09	90578	14126	6.412148	67193	11110	6.047975	157786	25319	6.231921

7. Conclusion – just a short summary of what you have achieved

I have achieved my goal of getting a speedup bigger than 1. I have also achieved quite a high speedup. This is because of mostly 2 reasons:

1. I calculate approximately how many primes there are going to be and don't need to use a linkedlist for my primes. (the efficiency is printed if you say yes to the prime test and the 100 biggest factors.
2. Each thread only checks if a big number is divideable by a prime. It does not divide it immediately. The prime is added as a factor, and when the factors are (parallelly) merged, the big number is divided by the factors, as many times as needed. This means that each thread only iterates over it's sector of primes.

8. Appendix – the output of your program

If you answer y and y, you will get a output of:

- An error check, to see if the programs generate the same primes (nothing if no error)
- The factorization of the numbers from (n*n-101)..(n*n-1), from each program.