IN3030 – oblig 3

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Daniel Sharifi – danish

Contents

1. Introduction

This report is about parallelization sieve of Eratosthenes and prime number decomposition (factorization). It discusses and compares the sequential and parallel version of the algorithms, and aims to see if there are benefits to gain by parallelizing.

2. User guide

Parallel:

- java Main 0 [n] [threads]
- java Main 0 200000 10 n = 200 000, threads = 10.

Sequential:

- java Main 1 [n]
- java Main 1 125000 n = 125 000.

3. Parallel Sieve of Eratosthenes

The algorithm can be split into four parts. Assume 'n' is the maximum number we are searching primes for.

1. Find all primes up to $\sqrt{n}$, and cross out all multiples of these primes (but not itself).
   All remaining numbers not crossed out are primes.

2. The remainder of the sieve is of size n - $\sqrt{n}$. This remainder is now searched through, and for each cell if it is not flipped, it is a prime number.

I am parallelizing step 2 by partitioning the remainder of the sieve to the worker threads, and make the search for the prime numbers. However I'm having trouble to make my solution work, as it has a lot of errors and bugs at the moment.

4. Parallel factorization of a large number

The parallelization of the factorization is simple. However, after implementing it I've realized it can be implemented in a much better way, giving greater speedups compared to the sequential version.

```
private void factorize_partly(int start, int end, ArrayList<Integer> factors, long base){
        long factor = base;

        // Find all factors factors[start]...factors[end]
        while (end > start && primes[start] <= (int) (Math.sqrt(base)) + 1) {
            if (factor % primes[start] == 0) {
                // prime number found
                factors.add(primes[start]);
                factor = factor / primes[start];
            } else {
                // check next prime number
                start++;
            }
        }
```

After finding all primes up to n, the primes are stored in an int array. This array is partitioned with a start and end index for each thread, which then executes the method above. The factors are then all stored in each threads' local ArrayList and are all added in the precode container when they are done. All the factors are multiplied together and this product is divided by the base, to check if the last factor also might a prime.

5. Implementation

My sieve is not working now, but the implementation is aimed to be as explained above. I have tested the factorization by using the sequential sieve. All the factorization results that the precode outputs are tested with a python script to assert that factors multiplied together are equal to the product.

6. Measurements

7. Conclusion