Programming Paradigms
Caucasus University – Spring 2025
Assignment 4 (10 points)

– Deadline: 23:59 on Sunday 8 Jun 2025 –

Building

You will need to install OpenMP development libraries for this assignment. e.g. for ubuntu linux:

```
sudo apt install libomp-dev
```

We use cmake to build our applications.

You can build debug version with thread sanitizers for testing your code:

```
> mkdir build-debug && cd build-debug
> cmake -DCMAKE_BUILD_TYPE=Debug..
> make
```

Build the release version for running benchmarks:

```
> mkdir build-release && cd build-release
> cmake -DCMAKE_BUILD_TYPE=Release ..
> make
```

Part I
OpenMP

1 Sieve of Eratosthenes (10 points)

The previous assignment asked you to use Posix threads to implement a parallel version of the Sieve of Eratosthenes algorithm for finding prime numbers and also suggested a strategy to parallelize your program. This exercise asks you to use OpenMP instead of Posix threads for the parallelization of your solution. You can of course write an OpenMP solution from scratch or reuse the previous implementation.

Besides your code, you need to provide a short section in your report that explains how you modified your solution and reports the speedup curve you get as the number of cores is increased. Was the OpenMP version easier or more difficult to write? Are the speedups you get the same better or worse (and why)? Refer to the third assignment for more information about how to benchmark your program.

Submitting

Don't forget to write a report with explenations and performance graphs. Once you're finished working, zip all source code, CMakelists.txt files and pdf report together and upload them on teams.

Grading

Correctness: 80% Your code should compile without any errors or warnings. We will test your programms on different inputs and edge cases. For a full score your code should pass all tests (some exercises might already have predefined tests).

Code Quality: 20% We expect your code to be clean and readable. We will look for descriptive names, defined constants (not magic numbers!), and consistent layout. Be sure to use the most clear and direct C syntax and constructs available to you.