# Assignment 2

Bjørn Christian Weinbach Thameez Bodhanya

September 19, 2021

## Instructions

For compiling the code that was provided in the assignment as well as the code that was edited for solving the exercises, the makefile provided was used. In some cases, the programs would not compile with -std=c++11 flag. This was mentioned in the lectures. If that is the case. Replace the flag with -std=c++0x flag.

We have used *siegnbahn.it.uu.se* for checking that the code runs. The code is structured in the following way:

- Q1
  - *question1.cpp*
  - Makefile
- Q2
  - *question2.cpp*
  - Makefile
- Q4
  - Coarse-grained locking.
    - *benchmark_example.cpp*
    - *benchmark.hpp*
    - *sorted_list.hpp*
    - Makefile
  - Fine-grained locking.
    - Same as above
  - Coarse-grained locking TATAS
    - Same as above
  - fine-grained locking TATAS

    - Same as above
- fine-grained locking using scalable queue lock
    - Same as above

There is a makefile for every question and subquestion. To check a question just navigate to the corresponding code and run:

*Make PROG=**filename***

Where ***filename*** is the name of the C++ file to compile.

## Numerical Integration

To numerically calculate a integral of a function $f$. We can use the trapezoidal rule [Wikipedia contributors, 2021]. The trapezoidal rule approximates the area under a graph of a function as a trapezoid and calulates the area. This can be done for more and more trapezoids to increase accuraccy but also coming at a computational cost. The trapezoidal rule can approximate a definate integral with riemann sums by splitting the interval $[a, b]$ such as $a < x_0 < x_1 < \cdots < x_{N-1} < x_N$ and perform the following calculation

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{2}\left(f(x_0) + 2f(x_1) + \cdots + 2f(x_{N-1}) + f(x_N)\right) \tag{1}$$

The sum in this calculation does not depend on eachother and can therefore be parallelised and calculated independently and summed up when they are complete. This has been implemented in our code by creating a C++ program that takes in the command line arguments $N$ for the no of threads and $T$ for the no of trapezoids. All threads gets $\frac{N}{T}$ trapezoids besides the last unlucky thread that gets $\frac{N}{T} + T \bmod N$ due to ease of implementation. All the threads operate this function and work on a shared variable to calculate the sum, hence the mutexes.

```
void *calculateFactorial(void *conf)
{
  Config *cfg = (Config *)conf;
  double localResults;

  for (int i = 0; i < cfg->numTrapPerThread; i++)
  {
    int pos = cfg->startI * cfg->numTrapPerThread + i;
    if (pos == 0 || pos == numTrapezes)
    {
      continue;
    }
    localResults = localResults + 2 * function(a + ((pos)*w));
  }

  results_mutex.lock();
  results += localResults;
```

```
18    results_mutex.unlock();
19
20    pthread_exit(0);
21 }
```

Listing 1: non-determinism.cpp

# References

[Wikipedia contributors, 2021] Wikipedia contributors (2021). Trapezoidal rule — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Trapezoidal_rule&oldid=1036645689. [Online; accessed 19-September-2021].