Homework 2

due date: 7/6 23:59

General Guidelines:

- Submit a zip file with all source code + relevant files (e.g., MSVS soln. files) to build your source code.
- Your code should be compiled and run in release/x64 (in MSVS) or "g++ -o3" in WSL/Ubuntu
- Avoid command-line parameters and special compilation flags/configuration for your programs.
- Don't use any external libraries except Native C++11 and TBB.

Exercise 1: a combining binary tree barrier **– 30 points**

The use of sense barriers we learned in class may suffer from high-memory contention. One way to reduce this contention (possibly at the cost of increased latency) is to use a combining paradigm like a combining tree discussed in class.

Implement a combining full **binary** tree barrier. Assume:

1. The number of threads (N) is power of 2.
2. Each thread (upon arriving to the tree barrier) is mapped to one of the N/2 leaf barriers

Use the following class definition. Do not modify the existing interface. However, you can extend it as needed. *Node* is the class name of the inner tree nodes.

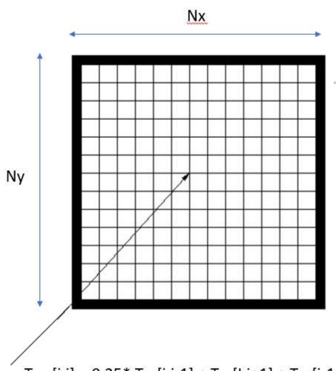
```
class BinaryTreeBarrier {
  public:
    BinaryTreeBarrier (int n); // n is total number of threads
    void barrier();
  private:
    void build_tree(Node* parent, int tree_depth);
}
```

Submit .hpp and .cpp files with the interface and implementation of this barrier class, respectively.

Exercise 2: heat 2D stencil computation – 40 points

This exercise is based on https://comp.anu.edu.au/courses/distMemHPC/sessions/PS3.html.

A 2D heat diffusion models the propagation of heat from edges to middle of a plate by a stencil computation. Assume a plate is of size Nx by Ny. The temperature at the edge is a constant value Tedge. The new temperature at each grid point [i,j] is initially set to zero and it is calculated iteratively as the average of the current temperatures at the four adjacent grid points. Iterations continue until the maximum change in temperature for any grid point is less than some threshold or when reaching a limit of maximum number of iterations (defined by max_iter).



$T_{new}[i,j] = 0.25* T_{old}[i,j-1] + T_{old}[I,j+1] + T_{old}[i-1,j] + T_{old}[i-1,j-1].$

1. The code in heat.cpp (attached with this assignment) currently provides a sequential implementation of this heat stencil. Get familiar with the code. Run the code with “heat.exe -mode 0”.
2. What are the main differences between the heat stencil pattern and the grid solver stencil pattern and how these differences may impact the way we can parallelize the heat stencil ?
3. Add code to run the heap stencil in parallel, using TBB “parallel for”. Make sure this parallel version runs as “heat.exe -mode 1”.
4. In case your “parallel_for” version (mode 1) uses a global “max_diff” variable to accumulate the convergence values, add code to further optimize this “parallel_for” using “local_max_diff” variables to accumulate the local max convergence values for each thread. Make sure this parallel version runs as “heat.exe -mode 2”.
5. Add code to run the heap stencil in parallel, using TBB “parallel_reduce”. Make sure this parallel version runs as “heat.exe -mode 3”.
6. Hope you experience the different performance of each mode.
7. Submit the heat.cpp file with your additional code.

Exercise 3: MapReduce – 30 points

MapReduce is well known pattern for processing big data sets on clusters (). In this exercise we want to implement MapReduce as a composition of two TBB patterns of *parallel_for* and *reduce*. The *parallel_for* applies a function across all elements of an input data collection while *reduce* performs a summary operation. We discussed the idea of this composition in class.

Implement mapreduce() to compute the dot-product of two vectors, A and B of equal size.

1. The mapreduce function should comply with its interface in mapreduce.h (attached with this assignment).
2. Pass the *map* function and *reduce* functions parameters as *lambda functions.*
3. Submit a source code file which includes the mapreduce() and a main() to demonstrate it.

Good luck !