Hardware Aware Scientifc Computing (WS 2023) **Exercise 4**

Prof. Dr. Peter Bastian, Santiago Ospina De Los Ríos Submission date: 2023-12-05

IWR, Universität Heidelberg

Notes: You may volunteer for individual subtasks.

Exercise 1 *Tree Barrier*

In the lecture you have seen an implementation of a barrier class that can be used to synchronize threads. Instead of having a single point of synchronization we will implement a barrier where synchronization happens in a tree like structure, a so called combining tree barrier.

1. Implement a combining tree barrier using `std::condition_variable` or `std::atomic`[2]. You can restrict yourself to the case of a binary tree. The number of leaves will correspond to the number of threads. Note: In `scalar_product_v5.cc` you have an example of using tree like structures.
2. Make sure your tree also works if the number of threads is not a power of two.
3. Write a test problem that shows that your implementation works properly. Compare your tree barrier to the given one for a high number of tasks.

(3+2+2 Points)

Exercise 2 *High Level C++ Parallelism Scalar Product*

In this exercise we will use some of the higher level C++ language features for the calculation of an inner product. This exercise is mainly about getting more familiar with those language features.

1. Use `std::promise`, `std::future`, `std::packaged_task` and/or `std::async`[3] for a thread parallel computation of the scalar product.
2. **Bonus:** Use parallel execution policies and suitable algorithms from the standard library for the scalar product computation.

   Note: Your standard library implementation will probably not include parallel algorithms. In this case it might just do a sequential computation or you might get an error. You have the following options:

   - Try to get it work by linking against TBB or openmp. Starting points for investigation could be the GCC parallel mode documentation and Stack Overflow question on C++17 parallel algorithms [4].
   - You could try using TBB directly. Again some possibilities:
     – Get only TBB and somehow get it running.
     – Get the Intel Base toolkit. It is best to install only parts of it, otherwise it might take ages .

[2] `std::atomic` has now `wait` and `notiffy_[one|all]` member functions that may serve similar purpose as conditional variables.

[3] Use whatever you want to practice ;)

[4] This might also depends on your compiler versions and you might have trouble getting some to work

- Another possibility would be: Use Godbolt with gcc-11.1 the compiler flag `-std=c++17` and link agaist TBB (click on the book button). You can’t rely on any performance or time measurements but it does seem to do something.

(3+2 Points)