Programming Parallel Computers

Instructions

Rules

Please read carefully the description of each individual task for detailed rules. Please note that some tasks ask you to use a **specific algorithm or technique**, while others give you a lot more freedom.

Your code has to compile and run correctly **in the computers we use for automatic grading**. Other than that, there are only a few specific rules you need to follow (see below). You have got **lots of freedom** and plenty of room for creativity — among others, you can freely use OpenMP, GCC vector extensions and other GCC-specific extensions, compiler intrinsics, inline assembly, POSIX library functions, etc.

Please keep in mind that our automatic tests are primarily there to help you catch honest mistakes. Your code **has to work correctly for all possible inputs**. Please keep in mind that our course staff will read your submissions and will double-check that you have been following the rules.

Be careful with these

You must not have **data races** or other similar bugs that lead to **undefined behavior** according to the relevant specifications. It is not enough that the code "seems to work", it has to be written so that it is **guaranteed** to work.

In GPU code, you must **check for errors** in all CUDA API calls — please see **our course material** for examples of how to do that.

What you cannot use

In CPU code, you are **not allowed** to use `std::valarray` or any parallelized or vectorized collections or algorithms in the C++ standard library (e.g. `_GLIBCXX_PARALLEL`, `__gnu_parallel`, `std::experimental::parallel`, `std::execution`).

In GPU code, you have to stick to the basic CUDA API. For example, you are **not allowed** use **Thrust**, **cuBLAS**, **NVBLAS**, or other similar libraries.

In the GPU exercises, you are **not allowed** to use **OpenMP** (or any other means of multithreading) in the CPU-side code. This helps to ensure that you are doing a majority of computationally intensive calculations in the GPU side. You can still do lightweight single-threaded processing on the CPU side when it is helpful.