UNIVERSITÉ DU LUXEMBOURG

Rule 110: Three Ways to Parallelize

Parallel Computing

Goals

- ✯ Learn to parallelize a code using C++ execution policy, OpenMP and `std::thread`.
- ✯ **Relevant videos**: if you want to get started as quick as possible, follow the videos annotated with "fast track". Of course, all videos should be watched eventually.
  - C++ execution policies:
    - HPC Top-down
    - Benchmarking
    - Easy Acceleration (**fast track**)
    - Arithmetic Intensity
  - C++ `std::thread`:
    - Multithreading in Theory (**fast track**)
    - Multithreading in Practice (**fast track**)
    - Static Decomposition (**fast track**)
    - Load imbalance
    - Fork-Join Model
    - Privatization
    - Synchronization with Barrier
  - OpenMP (**fast track**)

Deliverables

1. The code on your Github repository generated by clicking here: `https://classroom.github.com/a/oqsJYk8l`
2. **Reviewer:** Paul Aromolaran (Github: `PaulAroo`)

Rules

1. You can discuss your design and your results on Discord or orally, but please don't share your code.
2. This is a solo project.

Exercise 1 – Three shades of parallelism

Parallelize the Rule 110 algorithm you wrote previously without pattern detection (the parameter `--pattern` will not be provided). Propose three versions:

- Using C++ execution policies and standard algorithms (check out `std::transform` and `std::views::iota`).
- Using C++ threads and explicit division of the data.
- Using OpenMP.

Add a flag `--version [policy|openmp|stdthread]`, e.g. we can call your code with `./rule110 --version openmp`. The primary criterion is correctness, and an incorrect implementation gives 0 point. The next laboratory targets efficiency, so a parallel algorithm that is correct is sufficient to pass this lab. **Output:** the number of 1s in the array of the last iteration.

Exercise 2 – Benchmarking

Benchmark your code with the different versions, and various size of arrays and iterations. Plot your results and discuss the plots and results in the README.md.