Low-Level Parallel Programming: Assignment 2 - Vectorization and CUDA

Assignment 2 Report

Group 14: Aljaz Kovac, Jakob Nordgren, Simon Jaklovsky

QUESTIONS

What kind of parallelism is exposed with your code refactorization?

Data-level parallelism.

Explain your solution. What kind of code transformations were required for this lab, and why were they needed?

The given codebase was looping through an array of agent objects, or in other words, an array of structures (AoS). This data layout does not work with SIMD instructions since one cannot perform one type of instruction on different types of data. Thus, transforming the data into a structure of arrays is paramount. Storing the same type of data (i.e., the agents' positions) in arrays allows for one SIMD instruction to act on multiple agents' data.

How is an agent represented in the new code?

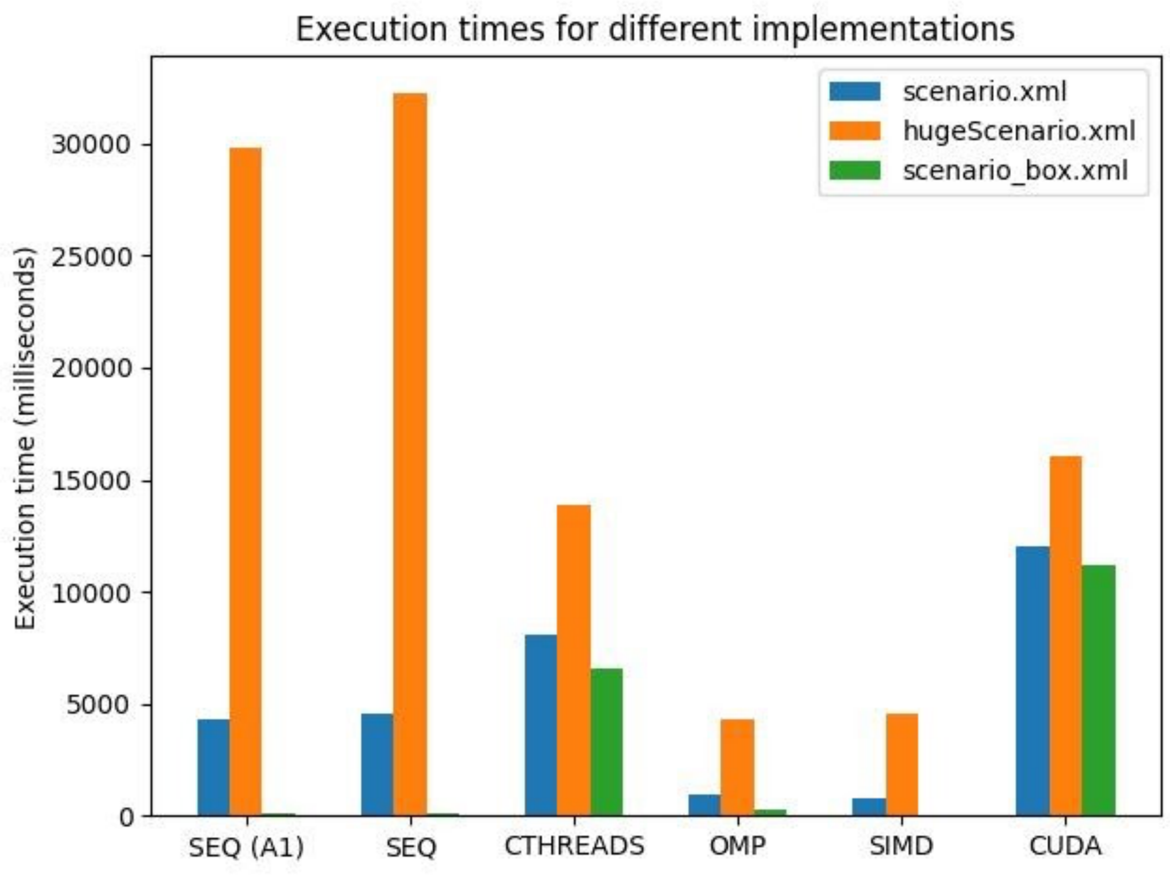
Each agent corresponds to an index in the newly introduced arrays. The agent classes' x and y coordinates were refactored to integer pointers, pointing to the corresponding element in the array.

Did the performance of the serial version get affected by the changes you made to allow SIMD operations and if so, how?

It actually performs a bit worse than the original serial version. This is probably due to the fact that agent coordinates no longer reside in the agent object, but must be fetched from memory on each agent update, possibly causing cache misses.

Which parallelization has been most beneficial in these two labs? Support your case using plotted timing measurements.

Across the different scenario sizes, SIMD and OpenMP were the most successful implementations and there is not much difference between the two for our measurements. While OpenMp slightly outperforms SIMD for the huge scenario, the SIMD fares much better with smaller jobs. One can suspect, however, that the OpenMP implementation would have been favored if it used the same cache friendly vector programming techniques used in the SIMD implementation. The best performing implementation would probably be a combination of the two. Our current CUDA implementation unfortunately suffers heavily from excessive memory allocations, and given another round of development the execution time for the huge scenario would probably be much faster.



Compare the effort required for each of the three (four) parallelization techniques. Would it have made a difference if you had to build the program from scratch, instead of using given code?

Compared to C++ threads and OpenMP, SIMD required a refactoring of the given code and as such required the most amount of effort. There was a sense of restriction in refactoring the code because all code strictly not related to the calculations assumes that the agents are represented as objects. Building the program in a data-oriented way from scratch could perhaps be a less confusing process, however, it would (with a near 100%-probability) necessitate writing more lines of code.

(BONUS) List your GPU specifications

The experiments involving CUDA were run on the snowy node, using an interactive session started with the following command:

interactive -A uppmax2021-2-28 -M snowy -p core -n 1 -c 8 -t 1:00:01 --gres=gpu:1

(BONUS) Is this code section suitable to be offloaded to the GPU? What makes it suitable / not suitable?

It is suitable to offload to the GPU since there are no data dependencies between the agents and the operations performed on each agent are the same, for the most part. The detection of whether or not an agent has reached its current destination and should have their destination updated requires an if-then-else clause. At this stage a bit of performance is given up as some threads which do not need to get a new destination will insert NOPs to wait for the other threads.

(BONUS) Give a short summary of similarities and differences between SIMD and GPU programming.

In both approaches the data layout is important for performance. SIMD gives more fine grained control as the instructions only act on a limited number of bits. GPUs usually act on larger amounts of data and are thus applicable to massively parallelizable tasks. When programming on the GPU one must take into account the (often large) overhead of moving data between the CPU and GPU caused by the slow PCIe link. In contrast, since SIMD is making use of the wider registers on the CPU there is little to no overhead.