LLPP: Assignment 2

Arveen Emdad Hamit Efe Cinar Iason Kaxiras

8 February 2024

1 Documentation

By default, the serial implementation will be selected. To choose different versions, pass in the `--pthread` flag to use C++ threads and `--omp` to use OpenMP. To use vectorization, pass in the `--vector` flag.

```
$ demo/demo --timing-mode --vector scenario.xml
```

By default, the current code utilizes OpenMP with vectorization. To change this behavior, you must comment out the `#pragma omp parallel for` directive in `ped_agent.cpp` under the corresponding function, `Tagent::computeNextDesiredPositionSIMD()` and rebuild the program.

Note: If multiple implementations are passed via command-line arguments, the last implementation passed will be used.

You can change the number of threads for C++ threads implementation using the `--nthreads n` flag where `n` is the number of threads.

```
$ demo/demo --timing-mode --pthread --nthreads 8 scenario.xml
```

2 Machine Specifications

The experiments were ran on the Snowy compute cluster with the following specifications:

"Snowy consists of 228 compute servers (nodes) where each compute server consists of two 8-core Xeon E5-2660 processors running at 2.2 GHz. We provide 198 nodes with 128 GB memory (s1-s120, s151-s228) , 13 nodes with 256 GB (s138-s150) and 17 nodes with 512 GB (s121-s137). All nodes are interconnected with a 2:1 oversubscribed FDR (40 GB/s) Infiniband fabric. In total Snowy provides 3548 CPU cores in compute nodes" [1].

3 Questions

A. After refactoring the code, this code can now utilize data-level parallelism more easily. This is because we have packed the values we want to compute in aligned, cache-friendly arrays of contiguous memory that can be loaded into the extended-width registers provided by Intel SSE.

B. The first transformation needed was to convert the current model from using an array of structures (AoS) to use structure of arrays (SoA). We did this by converting each member that stores the agent's position components into arrays. Without vectorization, only the first element of each vector is used, mimicking the current functionality of `Tagent`. Secondly, we unrolled the for loop by a factor of four, allowing us to load and compute four single-precision floating point values at a time using vector intrinsics provided by Intel SSE.

C. The serial version started to run about slightly slower than the previous version. This might be because the heap-allocated array accesses are less efficient due to pointer indirection in our new SoA implementation.

D. Vectorization has been the most beneficial in these labs. According to Figure 1, Figure 2, and Figure 3, you can see that it performed the best when compared to the other parallelization techniques as it had the lowest average execution time.

E. It would have definitely been way easier to write the SIMD vectorization part had we been able to write the code from scratch. It took a while to be able to transform it from an object-oriented approach to an approach that is more SIMD-friendly.

F. UPPMAX's GPU session has been used with the command:
interactive -A uppmax2024-2-5 -M snowy -p core -n 1 -c 1 -t 1:00:01 –gres=gpu:1 –gres=mps:25

G. The code section was not suitable to be offloaded to the GPU in terms of its data structure being object oriented. Therefore, the similar operations (creating structure of arrays out of the agent objects) done for the SIMD version was applied to agents in this version too to increase the GPU threads throughput and meet the GPU thread capabilities. In addition to the operations done for the SIMD version, the waypoints deques were flatten and were put in SoA's for the same reasons. On the other hand, number of agents being high in numbers somehow prevented data transfer overhead and limited parallelism to some point. Therefore, in that point of view, it can be regarded as suitable for GPU offload. This case can be observed in Figure 3. However, the other scenarios were not suitable for GPU programming as

H. Both SIMD and GPU programming exploit data level parallelism by performing the same operation on multiple data elements concurrently. Also, both SIMD and GPU programming involve vectorized operations, where a single instruction operates on multiple data in parallel.

One of their difference is the architecture of these approaches. While SIMD is typically implemented within a single processor core, GPUs consist of multiple processing units (cores) that work in parallel to process data concurrently. Another difference is that the frameworks they use. While SIMD uses frameworks like SSE and AVX, GPU programming is done using CUDA or other similar ones concerning parallel programming with GPUs.

4 Plots

See Figure 1, Figure 2, and Figure 3 in the Appendix.

5 Contribution

Arveen Emdad, Hamit Efe Çınar, and Iason Kaxiras have contributed equally to the solution of this assignment.

References

[1] *Snowy User Guide*. 2023. URL: `https://www.uppmax.uu.se/support/user-guides/snowy-user-guide/`. (accessed: 2024-01-22).

Appendix

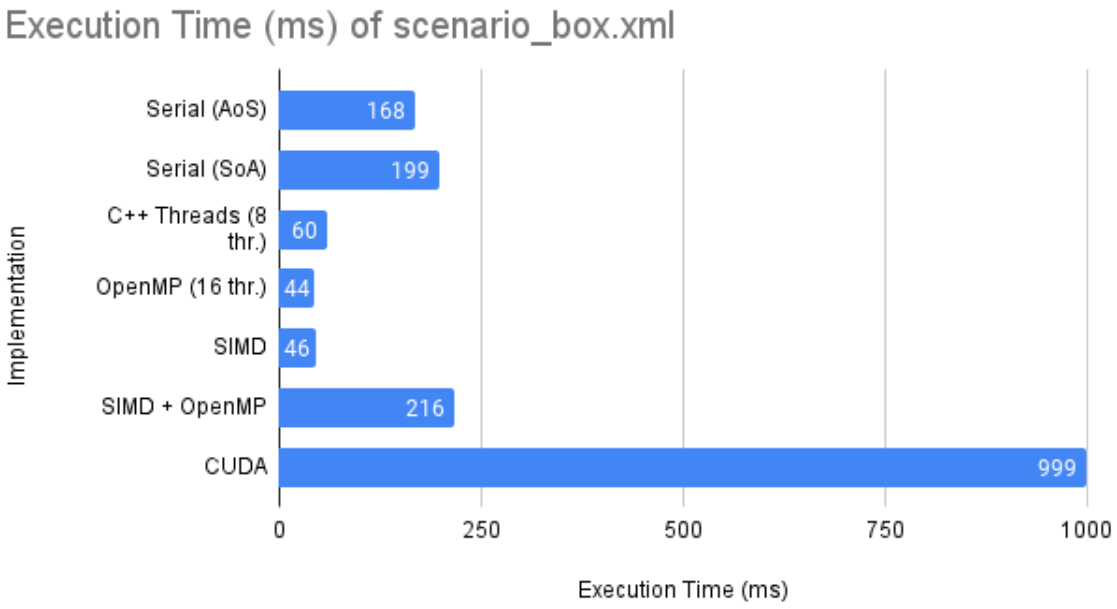

Figure 1: Execution Time of implementations running scenario_box.xml

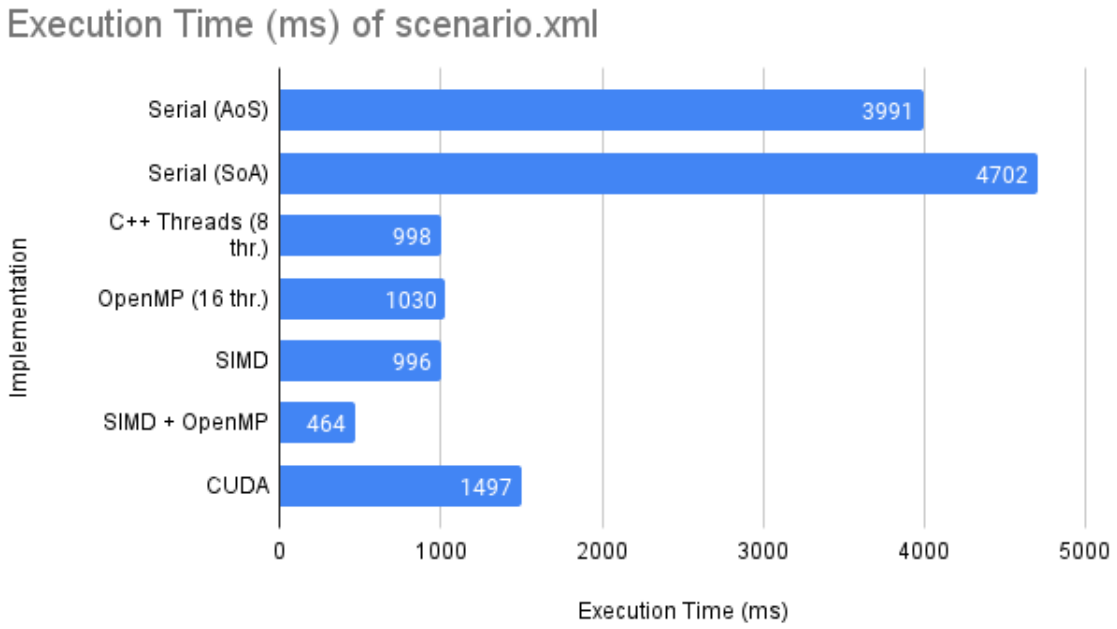

Figure 2: Execution Time of implementations running scenario.xml

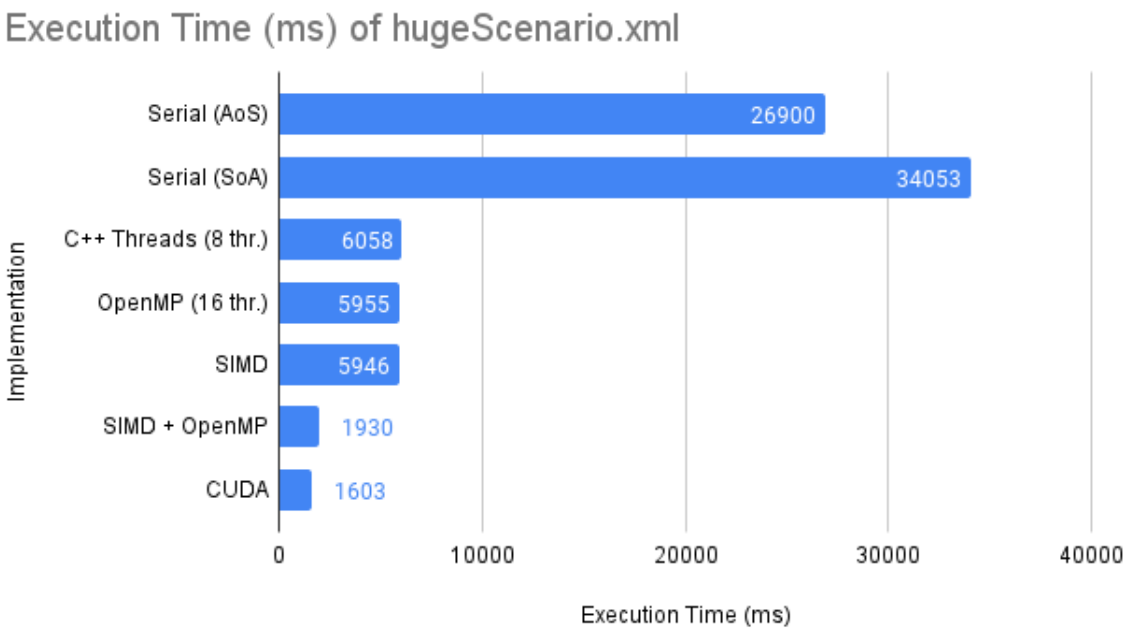

Figure 3: Execution Time of implementations running hugeScenario.xml