Results for CUDA Implementation

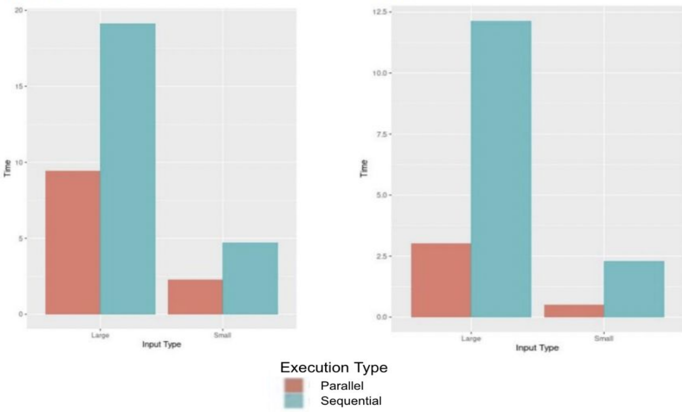
These are the parameters for the two types of problems on which the implementation was tested

Problem Type / Parameters	VSIZE	HSIZE	TIMESTEPS
Small Input Size	8000	50	5000
Large Input Size	8000	125	10000

- TIMESTEPS: number of units of time (typically seconds) that the data spans
- HSIZE: number of features computed in the hidden state of the network (i.e. the dimensionality of the hidden vector `h` for each time step)
- VSIZE: number of values in the output vector (over which an argmax will yield the prediction)
- For all tests, the number of threads launched per block was 5 (empirically found to be a good balance between (under)-utilization and speedup)

Speedup Plots: Before-and-After switch to per-kernel shared memory access (for `h` storage)

Before **After**

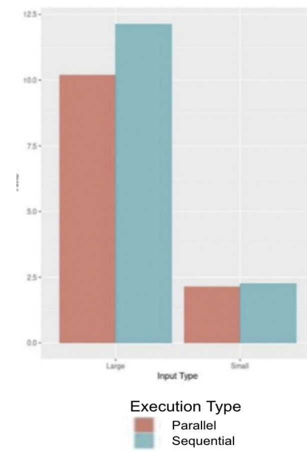


Results for OpenMP Implementation

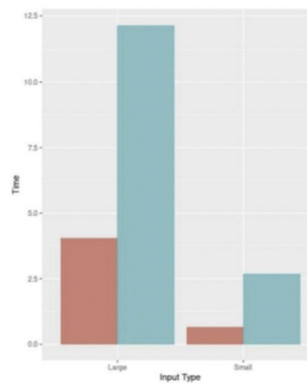
- We observed middling speedups with a logical replication of the code we had for CUDA kernels.
  - So, we devised the forward-filling strategy

Speedups with and without Forward-Filling

without



with

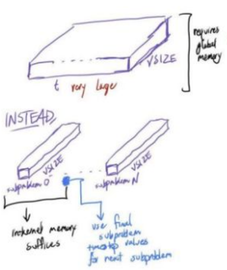


Note: this plot depicts the best result from many trials with multiple ff values. The results for ff = 15 when `num_of_threads = 500` were the data for this plot.

Because GPUs are much faster than CPUs, we chose the following (smaller) problem parameters to increase comparability with our CUDA impl.

Problem Type / Parameters	VSIZE	HSIZE	TIMESTEPS
Small Input Size	8000	50	500
Large Input Size	8000	125	1000

Discussion of Discoveries, Shortcomings, and Matters for Further Inquiry



CUDA:

- Using shared memory (per-kernel) was very conducive to speedup
- But we still have some aggregation of results that involves global memory
  - May be some optimizations in that approach we overlooked
    - For sufficiently small problem parameters, don't bother with global memory at all?
    - Compute fragmented sub-solutions (of contiguous sections of data) in separate executions and combine later, for bigger input sizes?

OpenMP

- Forward-filling turned out to be great for speedup
- However….
  - Its underlying logic is somewhat unsophisticated
    - interpolating data by copying an average, pretty much
  - Did not impinge correctness too much when the ff exceeded num_of_threads by 1-2 (Occam's Razor!), but correctness declined precipitously thereafter
  - Maybe doing some inference on the values to supply in the forward fill can mitigate this!
    - We are doing machine learning for value prediction after all :)
    - But, might require additional compute power at training time, to also develop neuron-like mechanisms for ff's values

Learning
…(1) network's parameters — We parallel-ized this
AND
…(2) parameters to interpolate network's outputs (?)
could this help us get more speedup?

References

Nabi, J. (2019, July 21). Recurrent neural networks (rnns). Medium. Retrieved December 10, 2021, from https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85.

Nvidia. (n.d.). Libcudacxx/atomic_thread_fence.MD at main · NVIDIA/libcudacxx. GitHub. Retrieved December 10, 2021, from https://github.com/NVIDIA/libcudacxx/blob/main/docs/extended_api/synchronization_primitives/atomic/atomic_thread_fence.md.

Using shared memory in CUDA C/C++. NVIDIA Developer Blog. (2021, October 29). Retrieved December 10, 2021, from https://developer.nvidia.com/blog/using-shared-memory-cuda-cc/.

GPU computing with CUDA Lecture 3 - efficient shared ... - bu. (n.d.). Retrieved December 10, 2021, from https://www.bu.edu/pasi/files/2011/07/Lecture31.pdf.

Cuda C++ Programming Guide. NVIDIA Documentation Center. (n.d.). Retrieved December 10, 2021, from https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

Cuda – tutorial 4 – atomic operations. The Supercomputing Blog. (2011, September 11). Retrieved December 10, 2021, from http://supercomputingblog.com/cuda/cuda-tutorial-4-atomic-operations/.

Understanding and using atomic memory operations. Understanding and Using Atomic Memory Operations. (n.d.). Retrieved December 10, 2021, from https://on-demand.gputechconf.com/gtc/2013/presentations/S3101-Atomic-Memory-Operations.pdf.

How to access global memory efficiently in CUDA C/C++ kernels. NVIDIA Developer Blog. (2020, August 25). Retrieved December 10, 2021, from https://developer.nvidia.com/blog/how-access-global-memory-efficiently-cuda-c-kernels.

Fisseha Berhane, Phd. Building a Recurrent Neural Network - Step by Step - v1. (n.d.). Retrieved December 10, 2021, from https://datascience-enthusiast.com/DL/Building_a_Recurrent_Neural_Network-Step_by_Step_v1.html.