Project Proposal

Anupam Pokharel and Lisa Mishra

November 2021

1 TITLE

The title of this project is the following: "Parallelization of Inference on Recurrent Neural Networks". The team working on the project consists of Anupam Pokharel and Lisa Mishra.

2 WEBPAGE URL

`https://lmishr.github.io/15418_project.html`

3 SUMMARY

Our objective is to discover and exploit opportunities for parallelism in Recurrent Neural Networks' (RNNs') inference-time forward pass computations. We aim to effectively utilize synchronization and atomic operations to mitigate the purely-sequential nature to which the data dependencies within and across neural network layers lend themselves, so that we can use parallel computation APIs (e.g. OpenMP) and vectorized operations on GPUs for multiprocessor speedup.

4 BACKGROUND

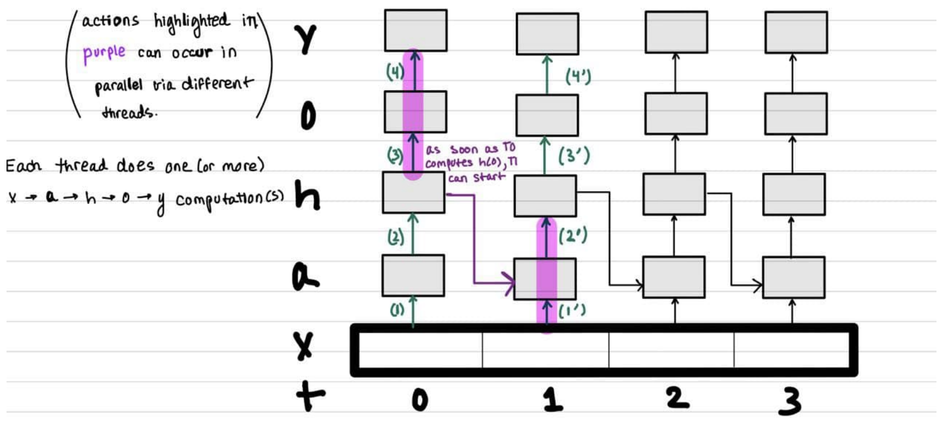
We are interested in finding effective ways to obtain speedup on the following basic algorithmic structure of a forward-pass computation in RNNs (source) :

```
// x is a vector of inputs
a[t] = b + W * h[t-1] + U * x[t]
h[t] = tanh(a[t])
o[t] = c + V * h[t]
y[t] = softmax(o[t])
```

where the following variables correspond to layers that are vectors of neurons, indexed by time and ordered by increasing proximity to the output layer, `y`: `a, h, o`.

We plan to find some ways to mitigate the constraints enforced by dependent data: for example, the constraint on `a[t]` is the completion of the computation for the hidden layer's (`h`'s) output at time `t-1`. If different threads were responsible for the forward-pass computations of adjacent time steps, there would need to be communications of hidden-layer values that would inhibit speedup. We elaborate more on the complications that this plan generates in the "THE CHALLENGE" section.

The following figure demonstrates the dependencies that this psudocode's computation entails:


(actions highlighted in purple can occur in parallel via different threads.)
Each thread does one (or more) x → a → h → o → y computation(s)
y
o
h
a
x
t
0
1
2
3
(1)
(2)
(3)
(4)
(1')
(2')
(3')
(4')
as soon as T0 computes h(0), T1 can start

5 THE CHALLENGE

RNNs' layers are composed of neurons that have dependencies both on the outputs from "below" layers and on outputs from previous time-steps, as illustrated in the "BACKGROUND" section. In the discussion from that section regarding forward passes for adjacent "columns" of neurons occurring on different processors, we alluded to the unavoidable necessity to communicate hidden-layer values. A salient topic to explore while implementing parallelization is whether this necessity is most efficiently met by using shared memory (which we will test using `OpenMP`), an orchestration of some form of point-to-point communication (we will test this as part of our additional goals, time permitting, with `OpenMPI`), and/or a tiering of caches that allows targeted memory-sharing among some processors.

6 RESOURCES

We will need access to a computer with a NVIDIA GPU, for our intention to develop CUDA-accelerated code. Additionally, we would benefit from access to a compute with a CPU capable of running programs on many processors (on the order of 128 would be ideal), like the PSC machines we used in Assignments 3 and 4. More discussion on our motivations for requesting these compute resources appears in the "PLATFORM CHOICE" section. We will consult online resources that have psuedocode or python code for RNN inference computations (one such resource was already consulted in our discussion and is linked in-line). Additionally, we anticipate that we will consult online resources for CUDA programming, including formal developer guides published by NVIDIA.

7 GOALS AND DELIVERABLES

7.1 What we PLAN TO ACHIEVE

(a) Identify parts of the computations involved in RNN inference that can be assigned to different threads

(b) Recognize and develop solutions to (or workarounds for, depending on reasonable incorrectness tolerance) inherent obstacles against parallelized computation of neuron output values, for both the OpenMP-based and CUDA-accelerated implementations of RNN inference.

(c) Achieve speedup through a parallel implementation of RNN forward pass computations using OpenMP.

(d) Achieve speedup through a parallel implementation of RNN forward pass computations using CUDA-accelerated code (for a GPU).

(e) Compare and analyze speedup (note: we expect the GPU will outperform the CPU handily, so we are interested in comparing the ratios given by speedup calculations, not absolute execution times) in both the OpenMP and CUDA implementations.

7.2 What we HOPE TO ACHIEVE

(a) Achieve speedup through a parallel implementation of RNN forward pass computations using MPI, and compare and analyze speedup to the CUDA and OpenMP implementations.

7.3 Our plans for the demo

Our main results will likely be encapsulated in speedup graphs to compare the speedup between the CUDA and OpenMP (and potentially OpenMPI, if we have extra time to implement it) implementations. We intend to supplement these graphs with discussions on the particular techniques utilized in both of the (or all three of the) implementations to mitigate the hindrances to parallelism created by data dependencies and to ensure maximal utilization of available parallel resources (i.e. to ensure the minimization of compute resource idleness). We expect the best way to interleave the discussion with the graphs will become more apparent to us as we acquire the results and approach the presentation-composing part of our project, in the last couple of weeks.

8 PLATFORM CHOICE

For the CUDA implementation, we will use C++, with CUDA code for GPU-based acceleration. This is an especially apt choice for us for two reasons. The first is that one of the most popular Python libraries for Machine Learning and Deep Learning, PyTorch, has many back-end implementations in C++ (and even is complemented by an API to augment library code with customized C++ routines). Secondly, we already have experience from 15-418's Programming Assignment 2 in composing kernel code that executes operations on many threads at once, so this is a great entry-point for us to begin development on the project. We will run this code on a machine that has a NVIDIA GPU (or is connected to one), like the `ghc59` machine our project group used in Assignment 2.

For the OpenMP-based (and, time permitting, OpenMPI-based) implementations, the code will be executed on a CPU (so no machine with a GPU will be required). Since we will be benchmarking all parallelized code against sequential implementations of RNN forward passes, there is no need for the CPU to be particularly robust. However, our results could be more meaningful and comprehensive if we had access to a machine like those offered by PSC used in Assignments 3 and 4 (so we can test our implementations on high processors counts, like 128).

9 SCHEDULE

WEEK OF	TO-DO ITEMS
10-31	- Write proposal report and populate website with proposal contents - Research C++ implementations of PyTorch library methods and how they are CUDA-accelerated
11-7	- Analyze dependencies, have an outline for implementation of RNN computations using CUDA acceleration
11-14	- Complete a basic implementation using the C++ implementations in PyTorch with CUDA Acceleration - Decide if project progress is sufficient to take on extension to compare additional implementation using OpenMPI
11-21	- Complete a basic implementation using OpenMP - Checkpoint meeting regarding milestone progress
11-28	- Obtain final results for both/all three implementations - Analyze data, compose graphs
12-5	- Polish results for demo