Transformer From Scratch

Thitrin Sastarasadhit: thitrin.sastarasadhit@gmail.com
Prof. Kenjiro Taura: tau@eidos.ic.i.u-tokyo.ac.jp

Methodology

Models:

- C++
- Chapel

} Implemented from scratch

- Pytorch A

} From Transformer-from-scratch

- PyTorch B

} PyTorch A with the transformer layer replaced with `torch.nn.Transformer`

PyTorch A was the original implementation from Transformer-from-scratch

Test Environment:

Property	Small-Size Model on Single Thread	Full-Size Model on Single and Multiple Threads
Machine	Machine A	Machine B
Model Size	Small	Full
Iteration	500	40

- Models were run on English-Italian machine translation task
- C++ and Chapel code structure are identical
- The Degree of parallelism is the same in both C++ and Chapel
- Timers were insert in each layer

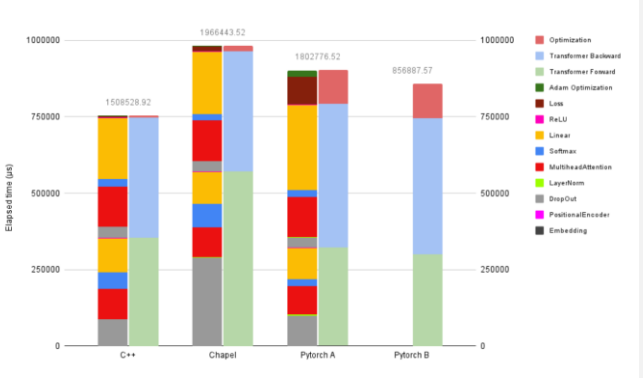
All code

Transformer from scratch

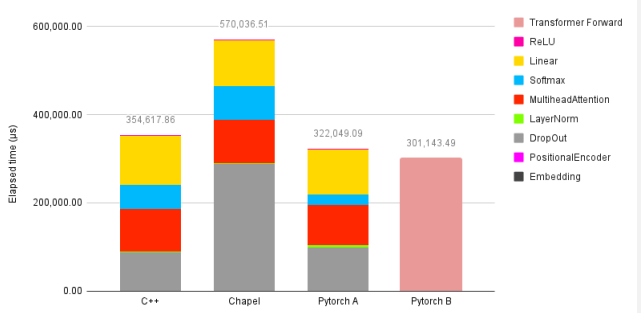
Small-Size Model on Single Thread

Tested on Machine A with Small-Size configuration

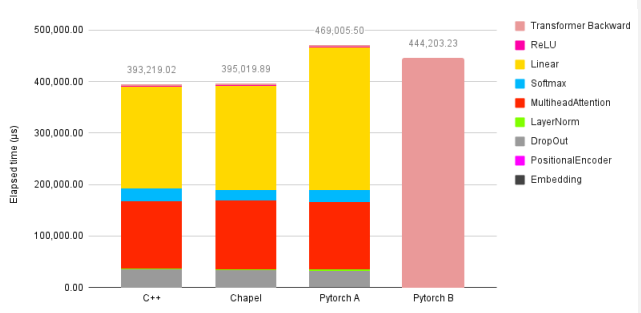
Time spent on each layer (us) for one training iteration
Slowness mainly came from forward-pass



Time spent on each layer (us) during single forward pass iteration
Softmax and Dropout layer are slow, other layer just as good as others.



Time spent on each layer (us) during single backward pass iteration
Chapel do as good as others in backward pass



Encountered performance issues and tricky solutions

Code

Detailed Results

Matrix Representation
- 1D array is used
- Multi-dimensional array is slow
- Nested arrays cause non-continuous array

Matrix Multiplication
- Block tiling with 64x64 block-size
- Chapel performs better/worse than C++ at some specific matrix size

Matrix Operation
- Passing an array view may prevent certain optimizations
- Passing start, end, and array separately influence more optimizations

Softmax
- Major cause of the model slowness
- No exponential vectorization is used.

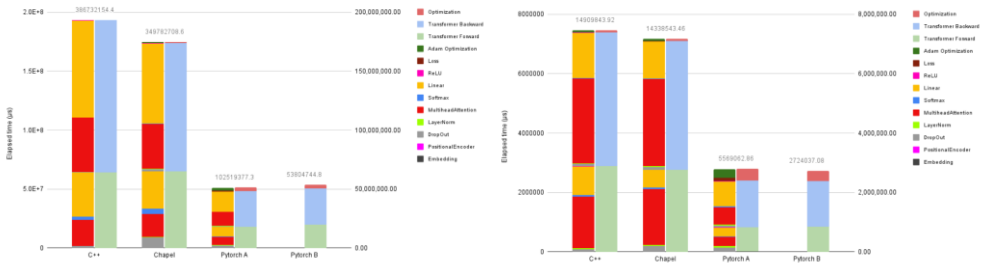
Dropout
- Use the interger version of randomSteam.fill()
- Slower random number generator
- Another main cause of the model slowness

Multihead Attention
- Mysteriously requires changing `param` to `var` in the config file to make the optimization to occur in certain place

ReLU
- The backward pass must be divided into two separate sections for optimization to take place
- The compiler generated code for the forward pass differs from C++ version which is slower when tested on full-size model

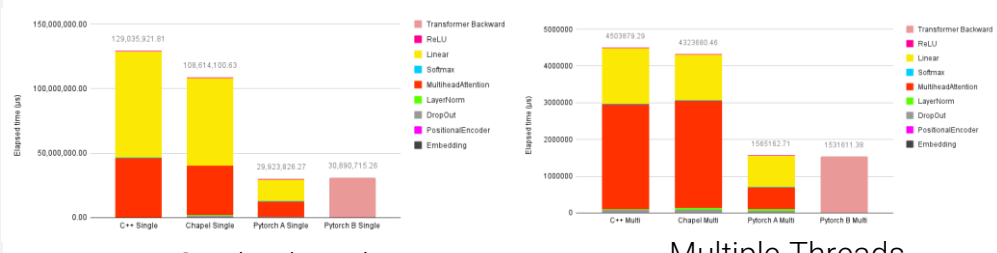
Full-Size Model on Single and Multiple Threads

Tested on Machine B with Full-Size configuration



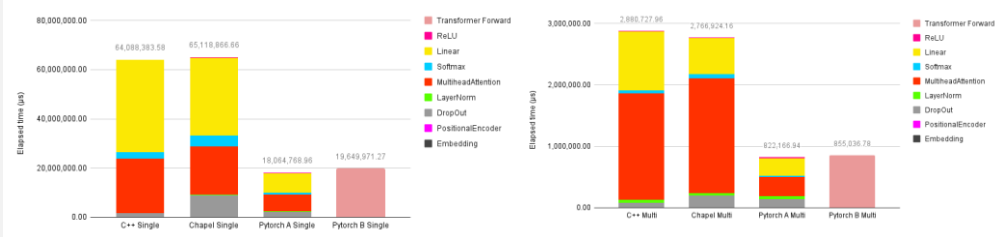
Single Thread | Multiple Threads

Time spent on each layer (us) for one training iteration
PyTorch is fast mainly because of its optimized linear algebra library. Chapel performs most layers as well as C++, with a better forward pass in some cases



Single Thread | Multiple Threads

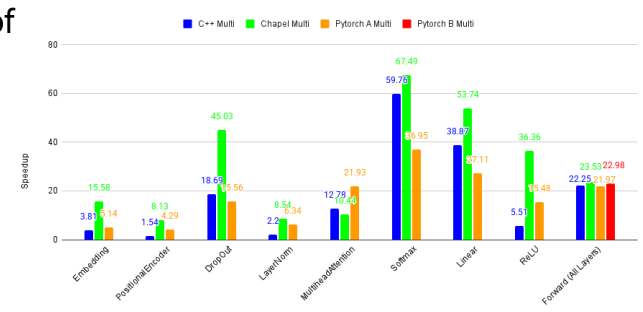
Time spent on each layer (us) during single forward pass iteration
Chapel performs better on the linear layer in a single-threaded setting. This is mainly the reason it is faster than C++ in this case.



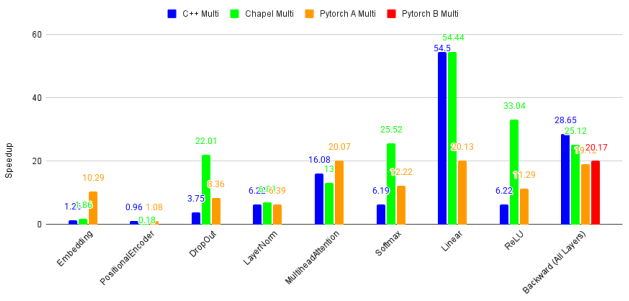
Single Thread | Multiple Threads

Time spent on each layer (us) during single backward pass iteration
In the backward pass, most layers perform as well as in C++, but achieve much greater speedup.

Speed up of each layer of forward pass.



Speed up of each layer of backward pass



Notes on the result

The multithreaded performance of most Chapel layers is comparable to C++, even though they perform worse in single-threaded execution. This is likely due to memory bandwidth limitations.

Code

Detailed Result