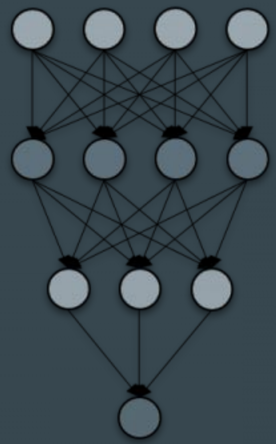
Improving Data Locality by Kernel Fusion in DNNs

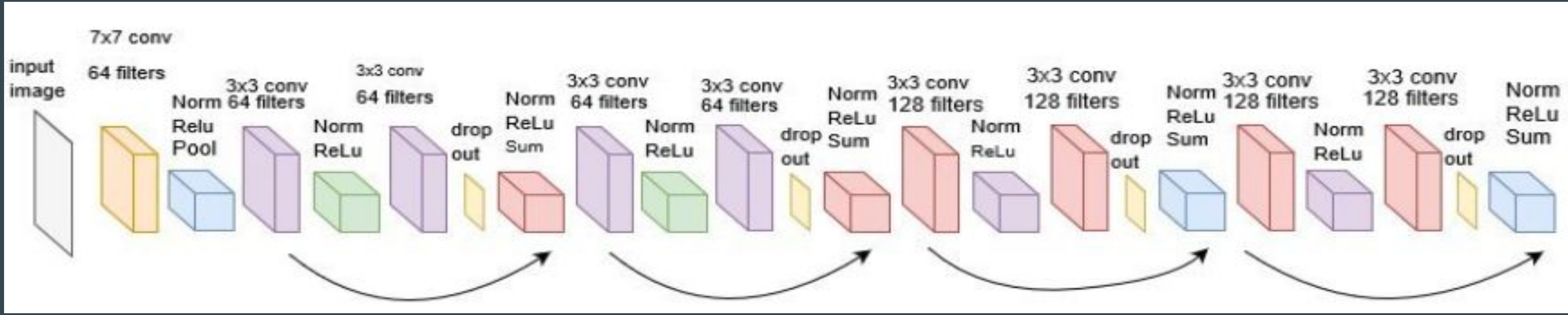


May 17, 2019

Snehil Verma, Bagus Hanindhito, Joseph Dean

Background

- Recently, there has been a growth in using CNN's for neural machine translation
- Training is very time consuming
- Many CNN's have numerous layers - each executed separately with separate kernel launches



Objective

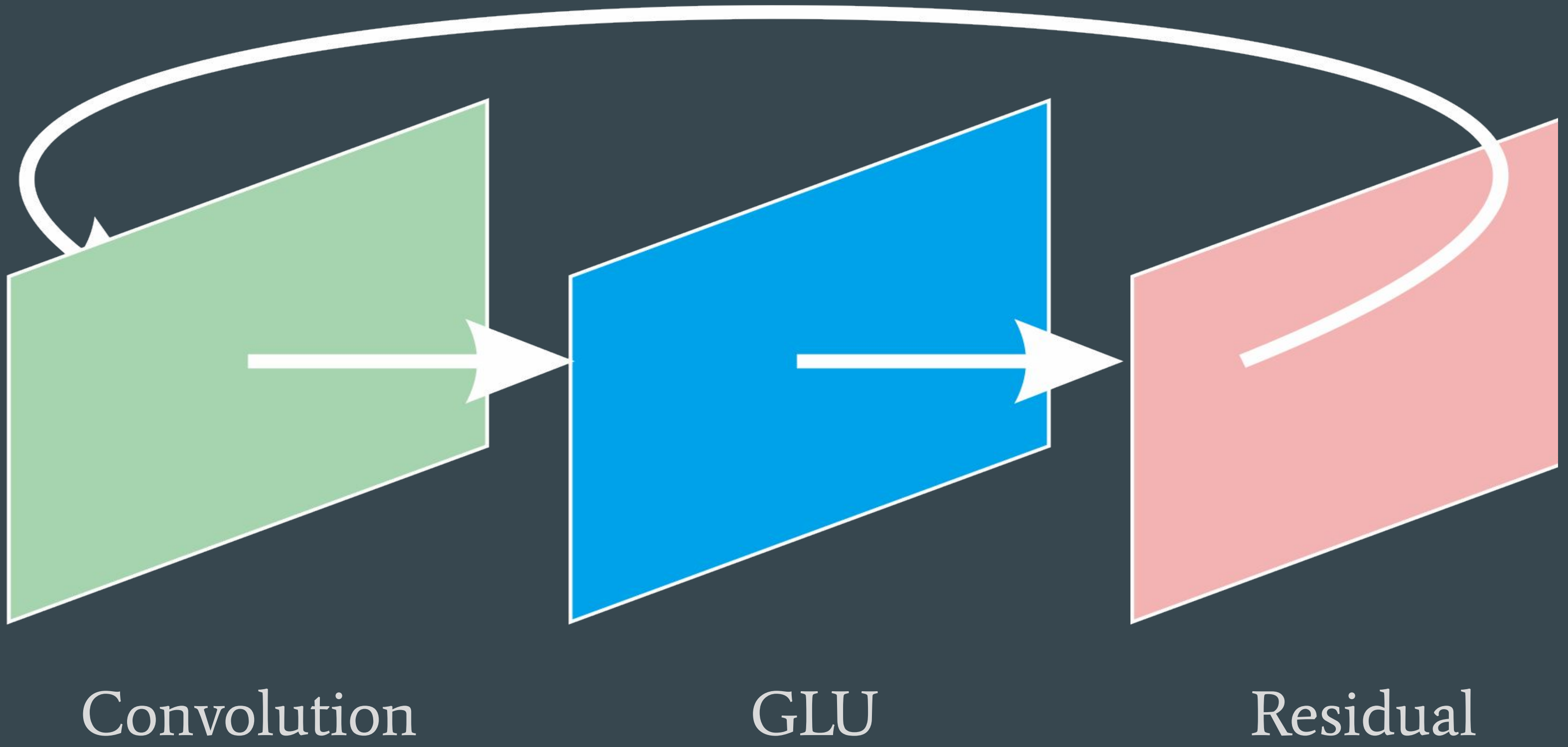
- Explore the benefits of fusing layers
- Many layers are element-wise operations and can be fused to improve locality
- Target Application - Fairseq

https://github.com/pytorch/fairseq/blob/master/fairseq.gif

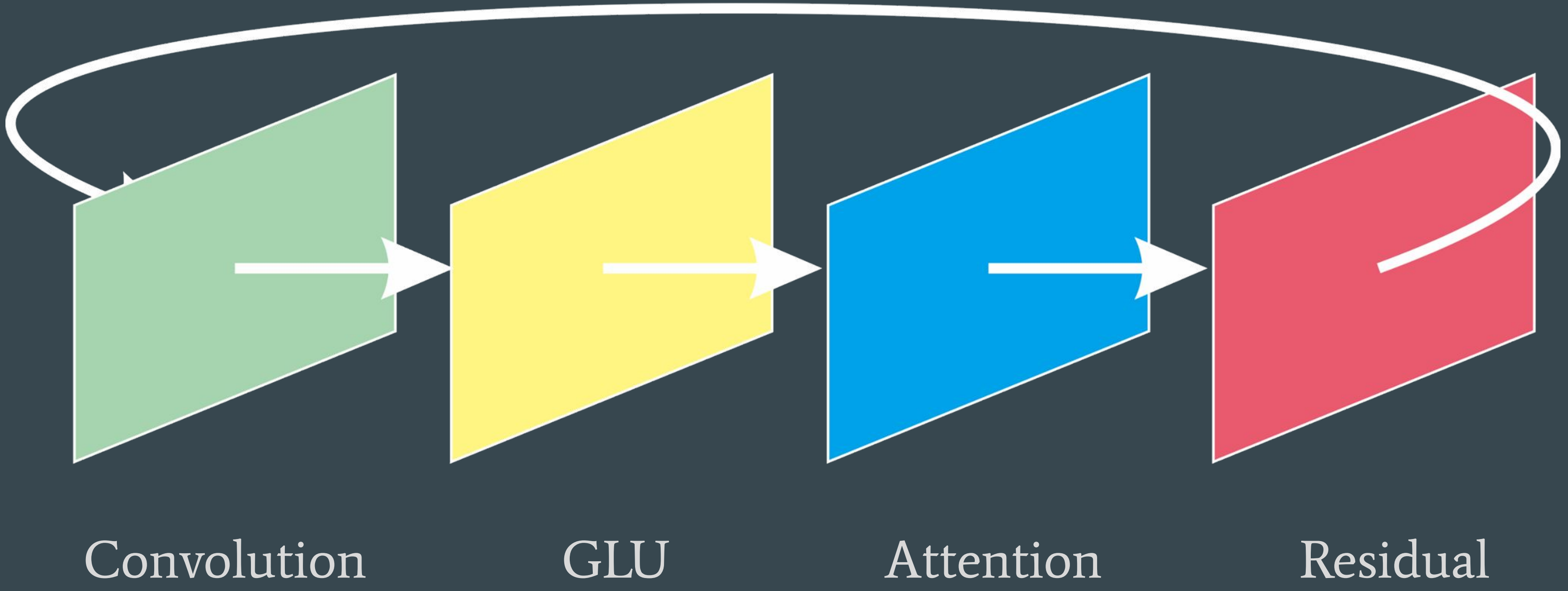
Software Setup

Model	• Gehring et al. (2017): Convolutional Sequence to Sequence Learning
Framework	• PyTorch
Language and Tools	• CUDA, C++ • cuBLAS, CUTLASS, cuDNN
Dataset	• WMT14 English-French

Encoder Stage



Decoder Stage



PyTorch Autograd analysis

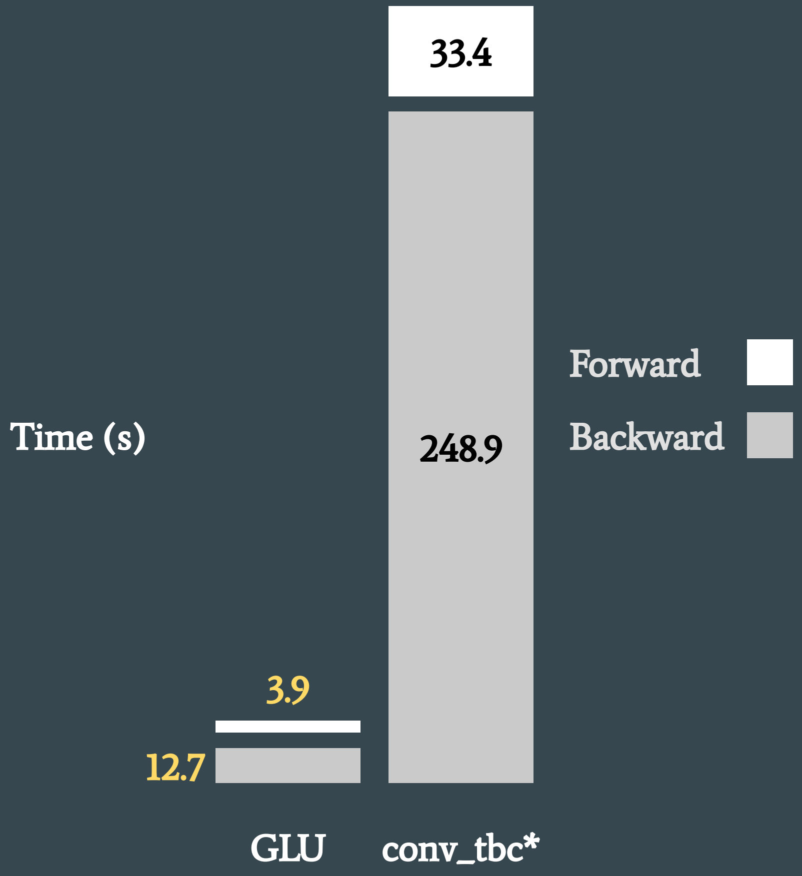
Profiled 3000 updates of the second epoch.

Findings

GLU operation takes around **6%** time of the convolution operation (including both forward and backward path).

Implications:

- Major performance improvement cannot be attained in fusing the two layers.
- However, fusing these layers is the first step towards improving data locality.



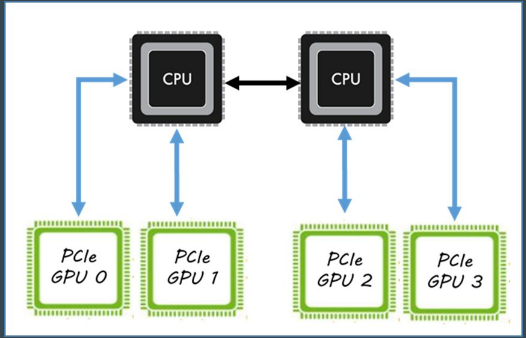
* Convolution TBC (Time, Batch, Channel)

Hardware Setup

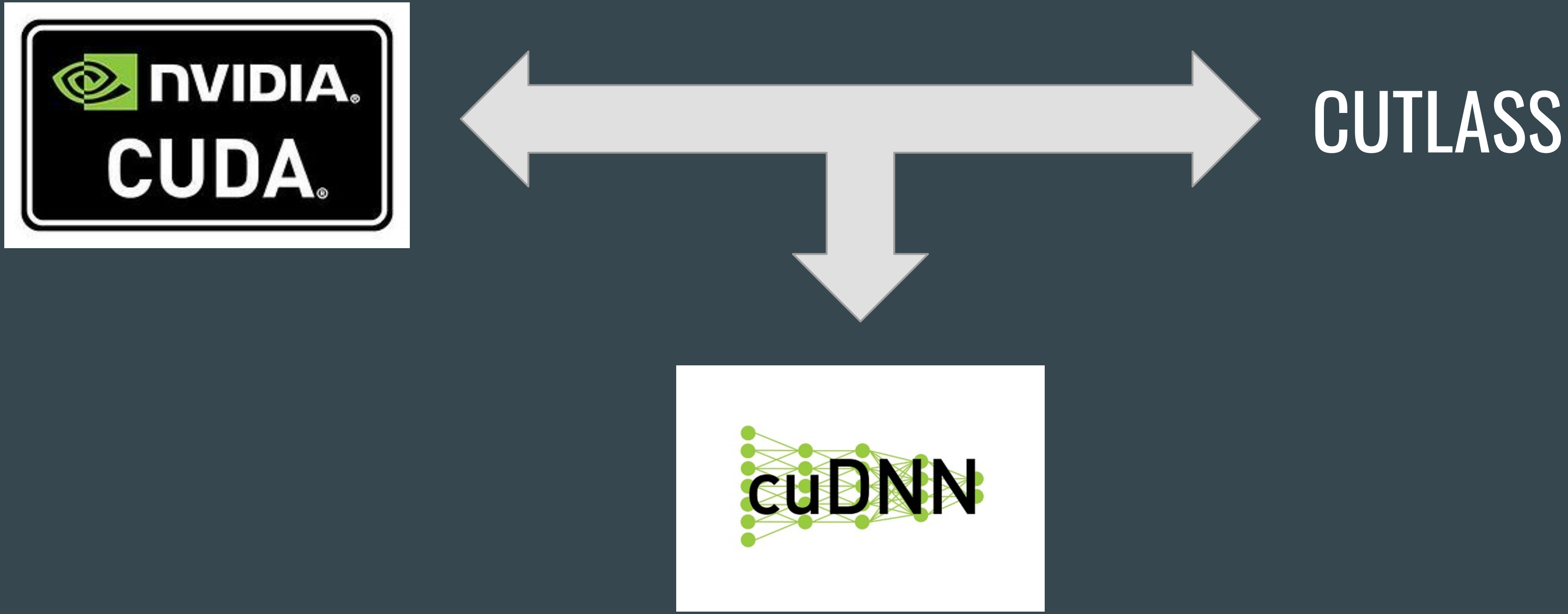
Dell PowerEdge T640

4 NVIDIA Tesla V100 GPUs

System Architecture



3 different approaches to improve Data Locality



CUDA implementation

Bare-metal CUDA implementation

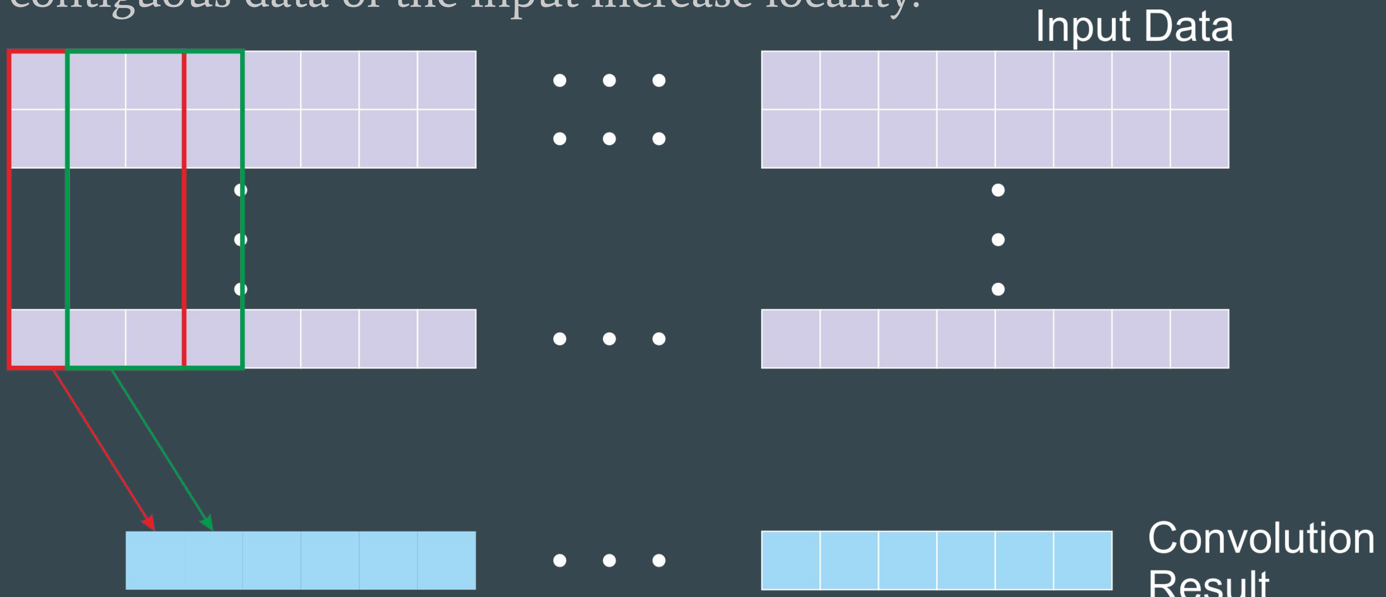
- ❑ More control over the data.
- ❑ The code's performance would not be comparable to the library performance.

Note that, the goal of the project is to perform kernel fusion and understand it's benefits, not to optimize the convolution function.

CUDA implementation (cont'd)

Convolution normally can be done by sliding the kernel into the input contiguously.

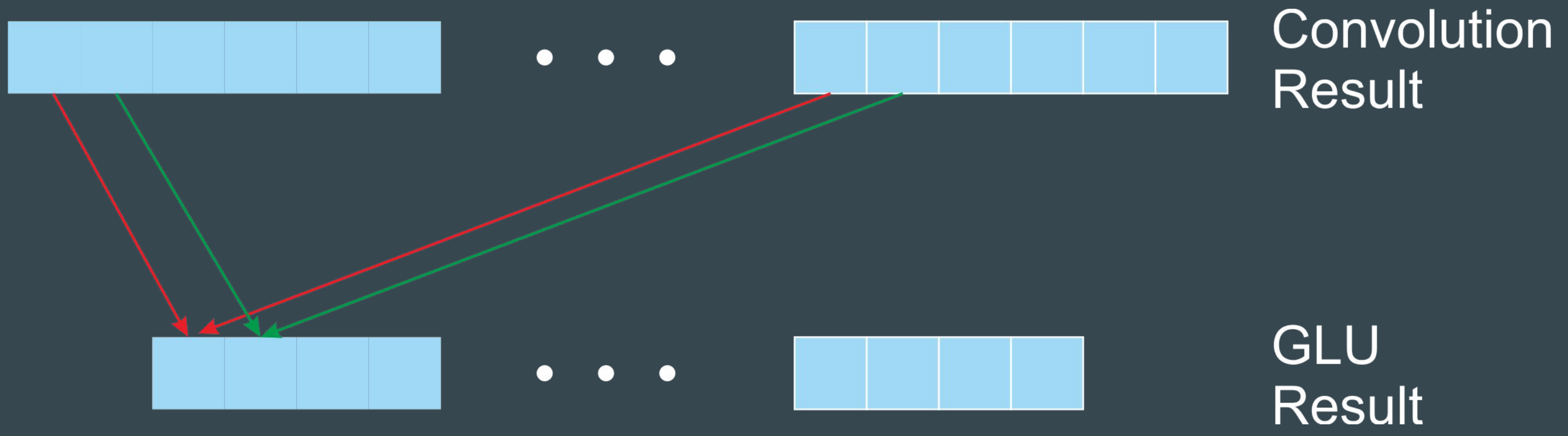
- Each computation for each kernel position is highly parallelizable.
- Operating on contiguous data of the input increase locality.



CUDA implementation (cont'd)

The GLU divides the data into two parts of equal size and operates on one element from each parts at a time.

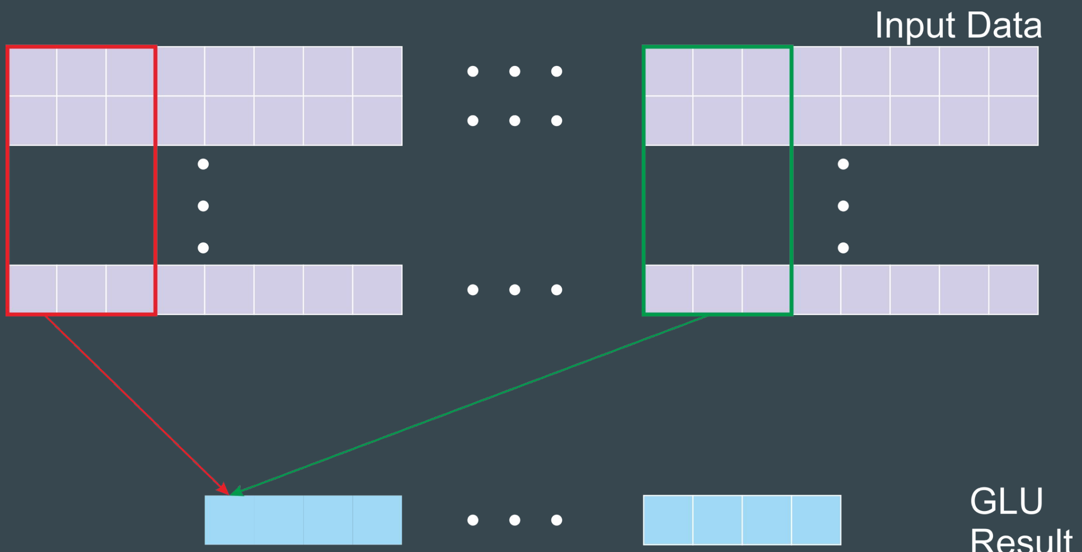
- ❑ Access pattern needs to be considered to fuse the GLU and Convolution together.



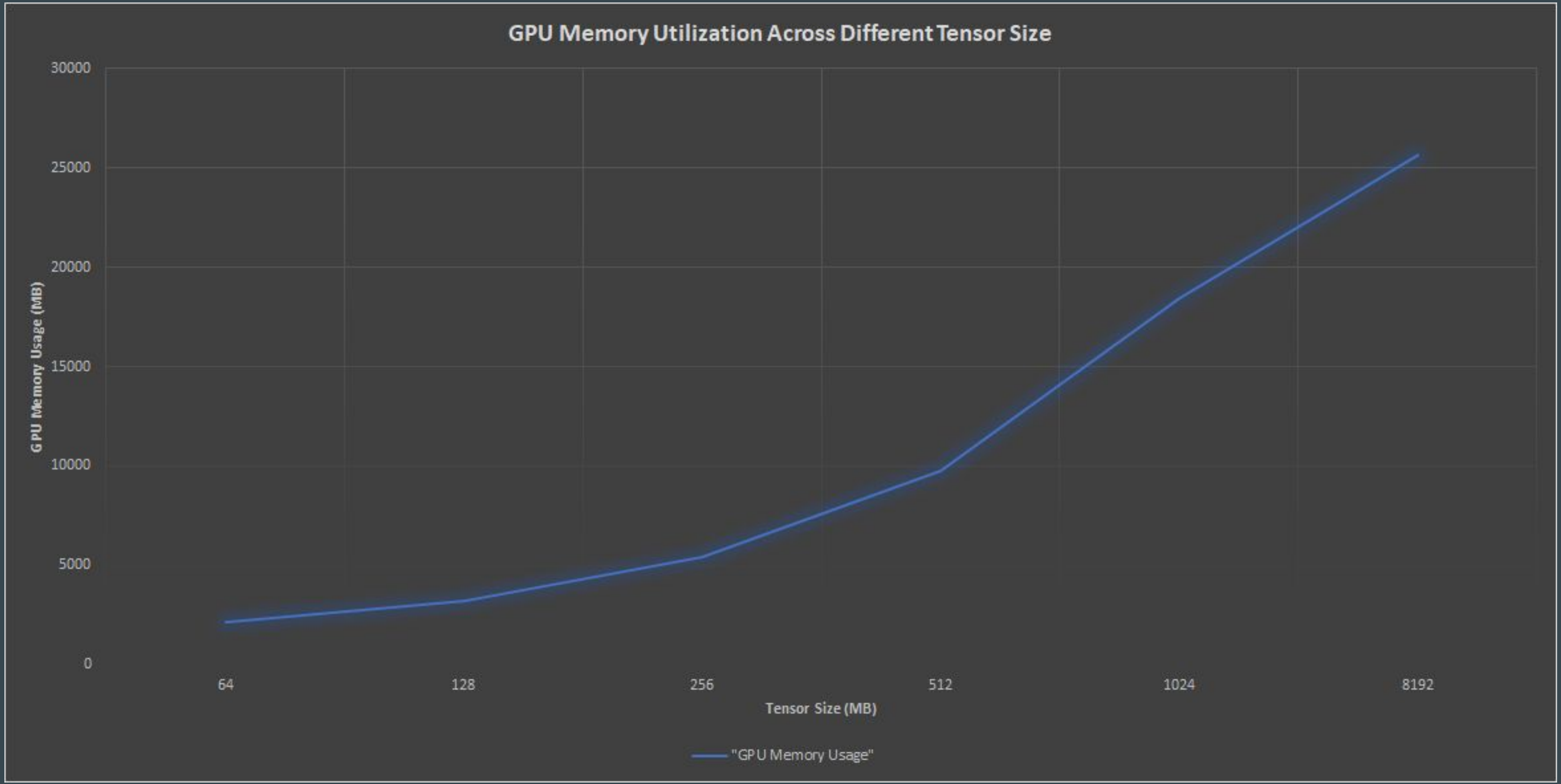
CUDA implementation (cont'd)

To enable GLU fusing, we need to modify the convolution operations so that it can produce two results required for GLU.

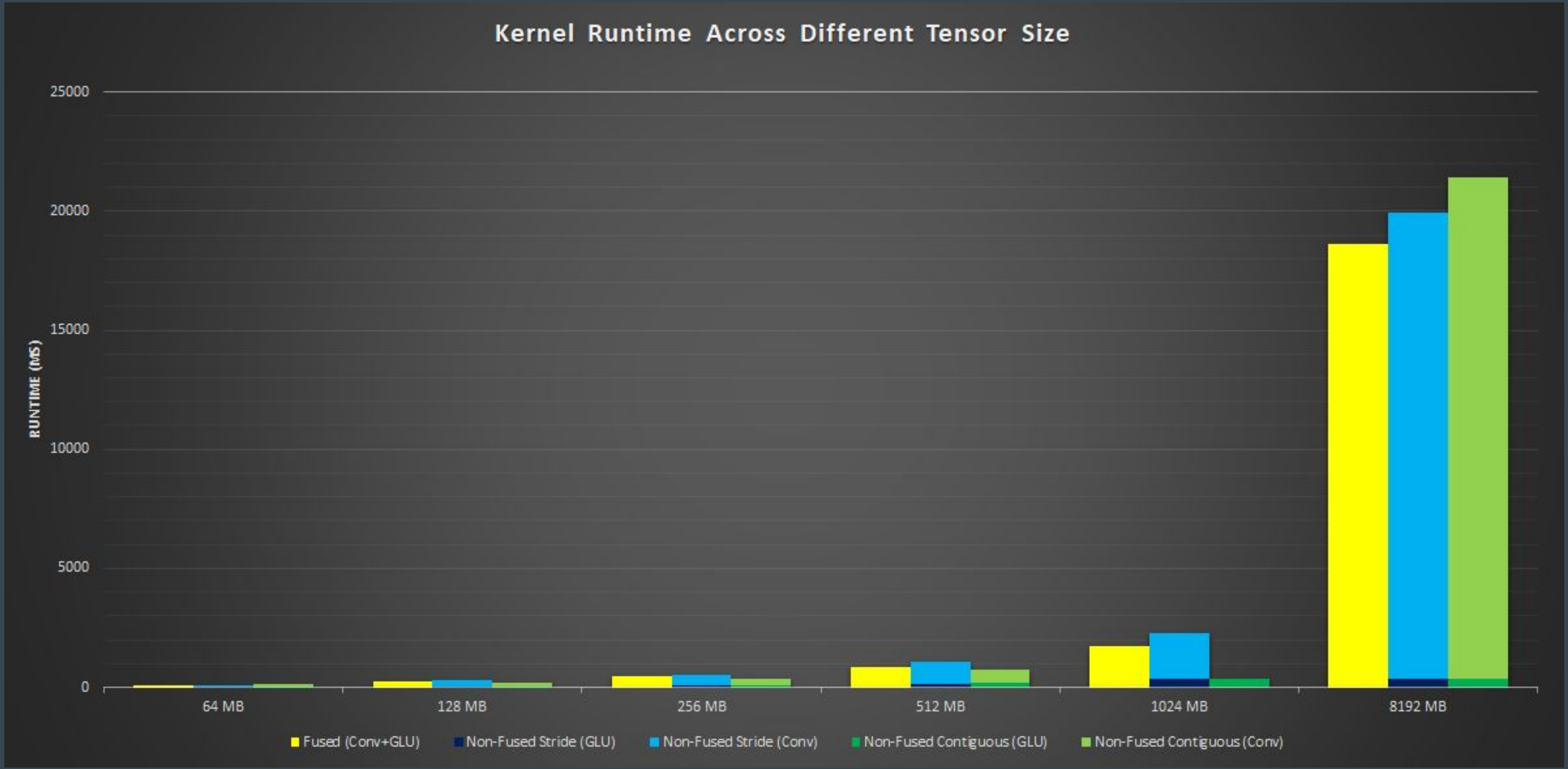
- ❑ Losing some locality for convolution because of non-contiguous operation on input data.
- ❑ Guarantee that the convolution results are still stored in register.
- ❑ Minimize the data that needs to be stored back into memory.



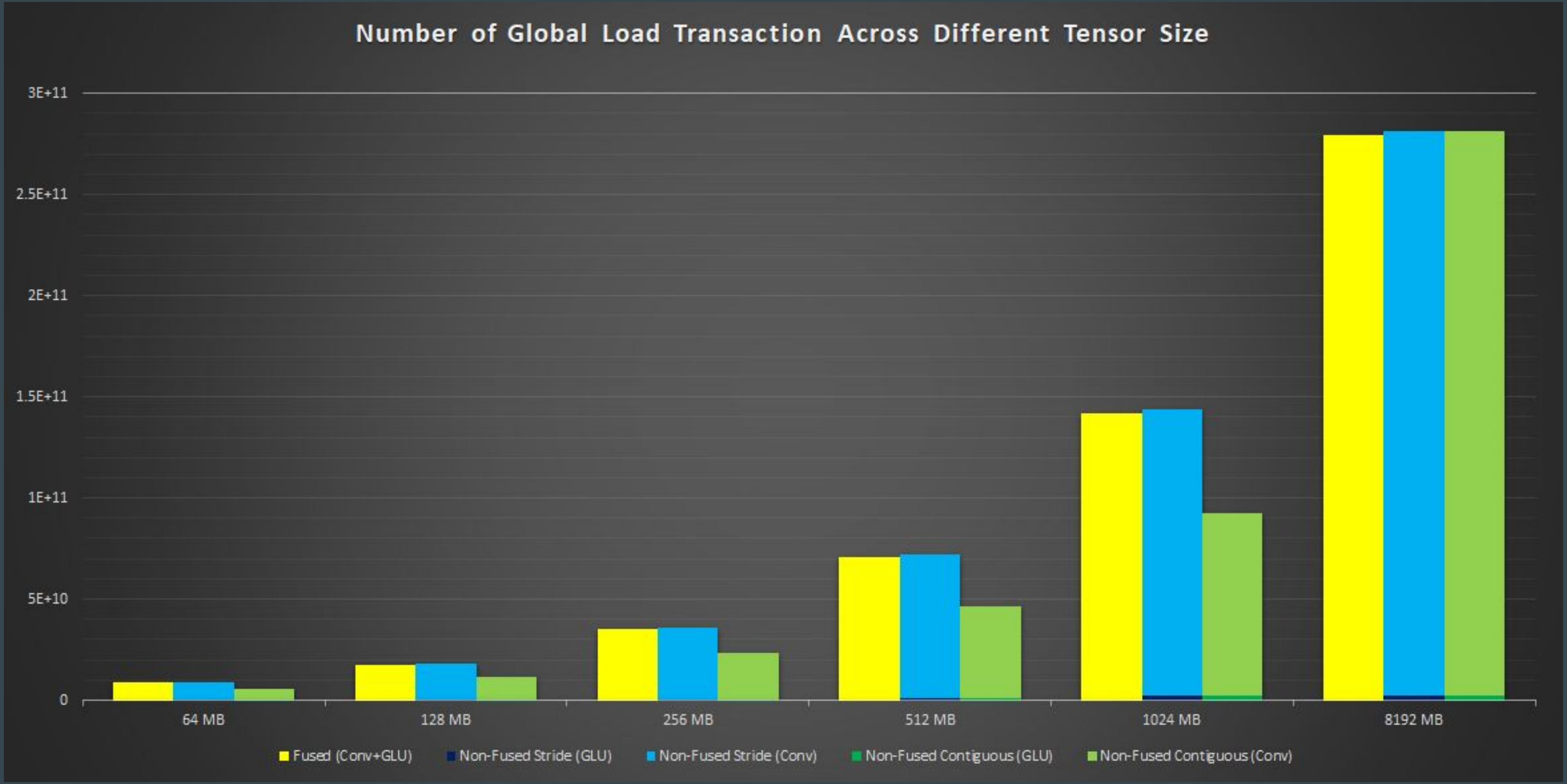
Results - Memory usage



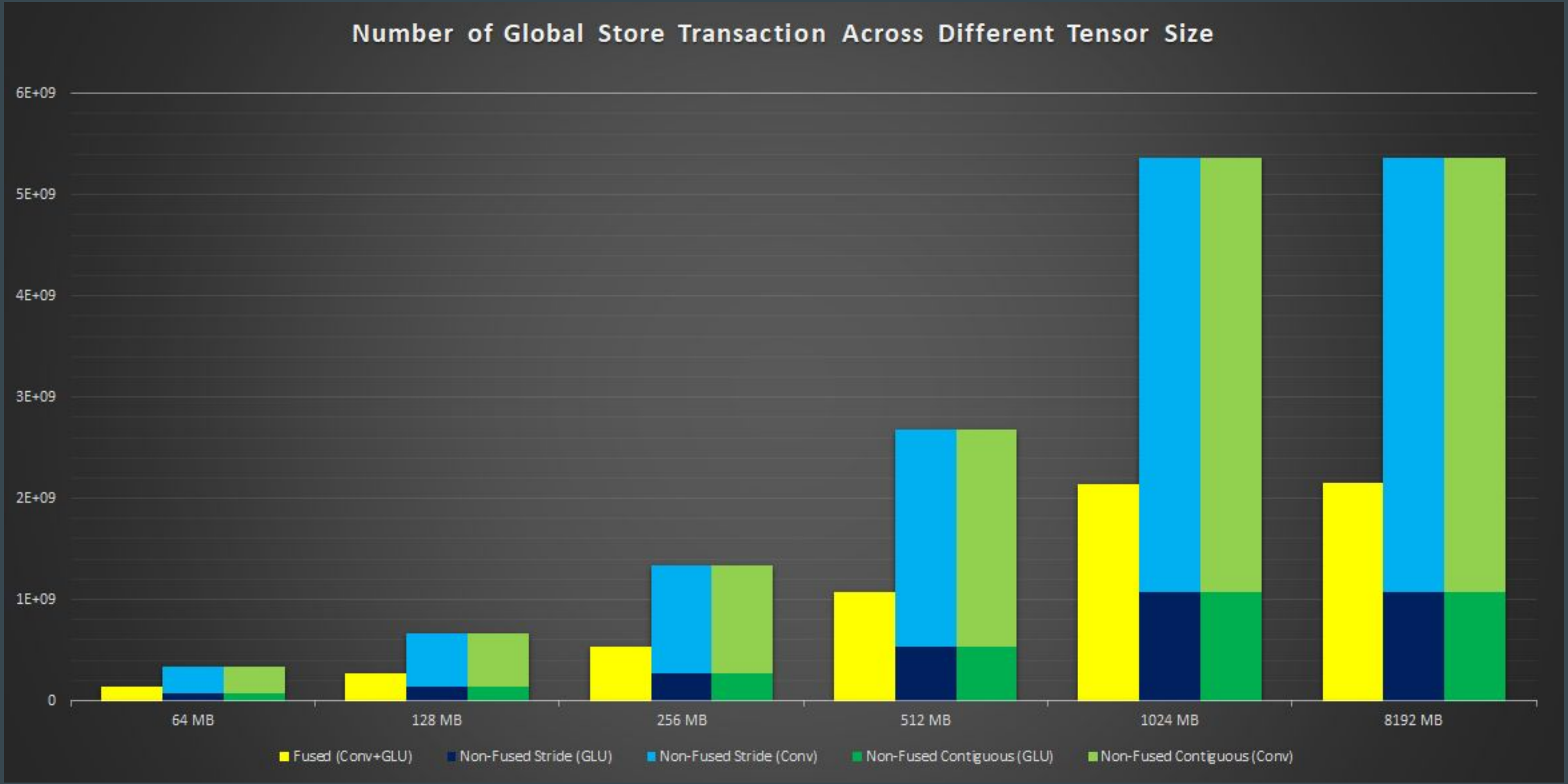
Results - Performance in seconds



Results - Global Memory loads



Results - Global Memory stores



What did we learn?

1. Closely understood the working of CNNs specifically concerning Translation (encoders, decoders, and attention layer).
2. A decent understanding of the working of PyTorch and its interface with the C++ and CUDA libraries.
3. Working with open source template library - CUTLASS
4. Working with cuDNN.
5. Data locality optimizations in CUDA by kernel fusion.
6. Extending PyTorch with custom C++ and CUDA functions.
7. One main thing we learnt is that we should have planned the timeline appropriately. We tried to cover a wide breadth but we weren't able to finish everything in time which lead to poor evaluation.

Acknowledgements

We would like to thank

- Qinzhe Wu
- Sangkug Lym
- Dr. Ramesh Radhakrishnan
- TAs: Yongkee Kwon and Kyushick Lee
- Prof. Mattan Erez

for their continuous advice and for making the hardware available.

Thank You!

Some of the code can be found @ https://github.com/UT-LCA/FusedConvGLU

Questions?