Lecture 14: GPT Project

Hrishi Shah and Charles Pei
ECE408/CS483/CSE408

GPT Project Reminder

Application Form is due Today!

- Teams consist of 3-4 people
- Signup form was sent over email/Campuswire
- If accepted, GPT project will replace CNN M2 & M3
    - Once chosen to pursue the GPT project, can not return to CNN
- No prior experience with ML/AI/Transformers expected!

Objectives

- To learn about the different layers of a transformer architecture, their anatomy, and how they interact
- Brief overview of the GPT project

Shortcomings of CNNs

“Hrishi was not able to visit Charles as his car broke down.”

What problems would a CNN have in understanding the context of this sentence?

- Who is “his” referring to? A CNN with a small window might think Charles, but he’s probably not the one driving based off this sentence
- CNNs can not flexibly support distant words (can raise window length, but even then you can’t dynamically change weights from sentence to sentence)
- Somehow need to support local context as well as distant words

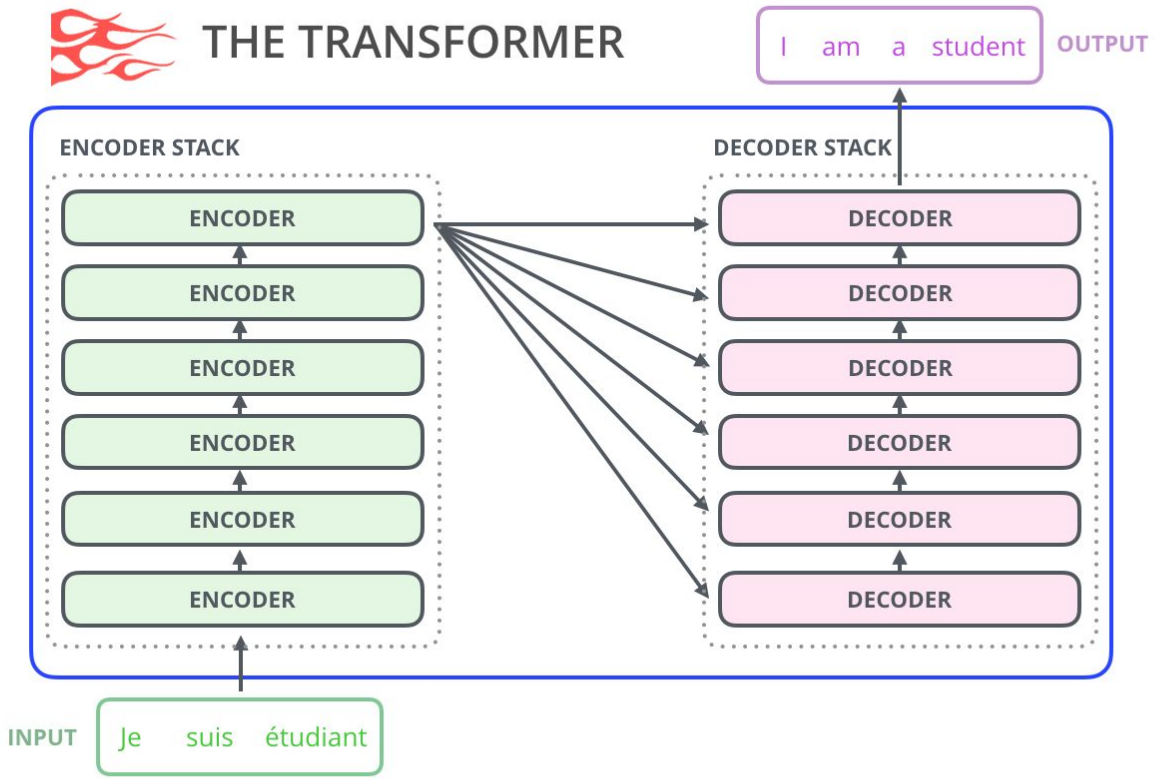
Motivation for Transformers

Self-attention mechanism can be used to capture dependencies and features across an entire sequence of text at once (much larger than some convolution window).

- Unlike static convolution mask, self-attention assigns different input elements different weights
- Led to the transformer model (Attention is All You Need) which removed all convolution aspects for solely attention mechanisms
- However, this has quadratic runtime scaling to the input length

Transformers are more modern, hence the motivation to introduce a new project that is more relevant.

The Original Transformer



A Transformer for Language Generation - GPT

The transformer from the aforementioned paper excels at tasks like text translation, but it is not set up for text generation.

- For language generation, we want the model to only see text before the next prediction. Should not be able to see into the future (causal)
    - Can remove the encoder stack then and replace it with a masked attention (triangular)
- Learnable position encodings, which are trained with the model rather than constant ones
- Much more layers through something called a transformer block that can be chained several times

Other minor nuances exist, yet this is at the heart of GPT.

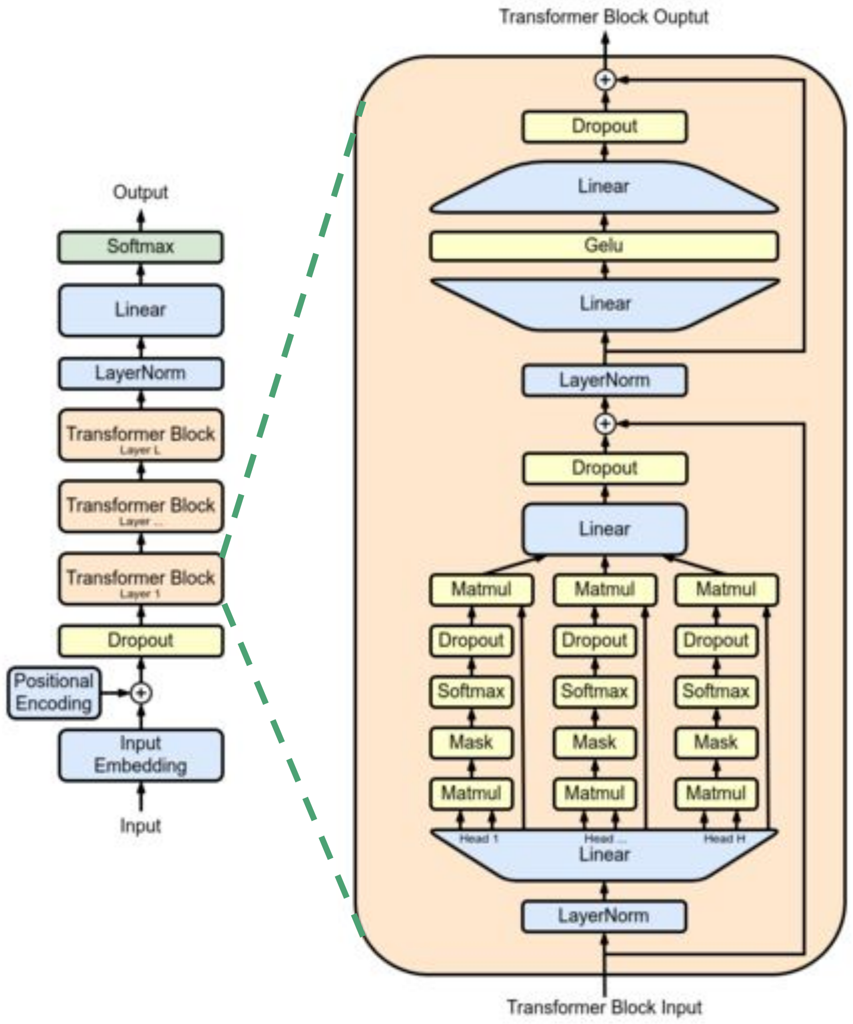
GPT Architecture Overview

“Decoder-only” architecture

Autoregressive:
Tokens are generated once at a time and fed back into the model inputs to generate the next.

Transformer block = decoder block, can stack several in a row (how do the # of parameters scale?)

# of attention heads and the hidden size can also be scaled. (More on this later…)

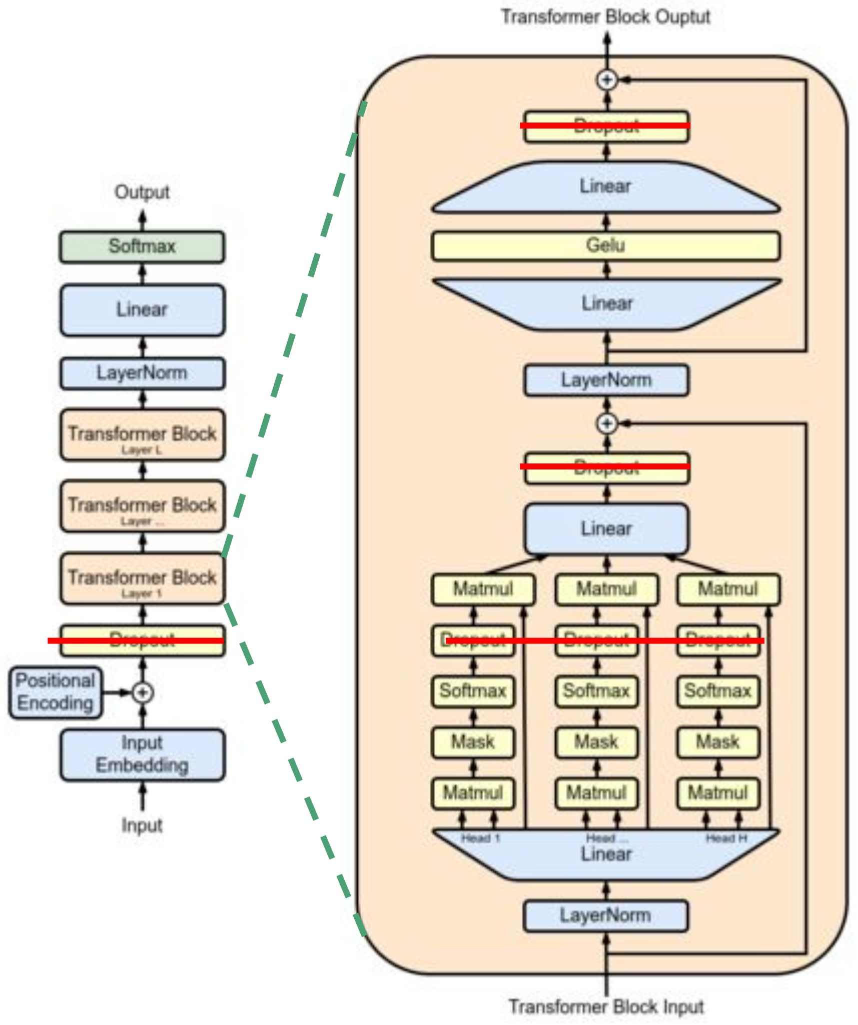


GPT Inference

Inference is the act of performing a task after the model is trained.

During inference, we do not care about turning off nodes (no dropout).

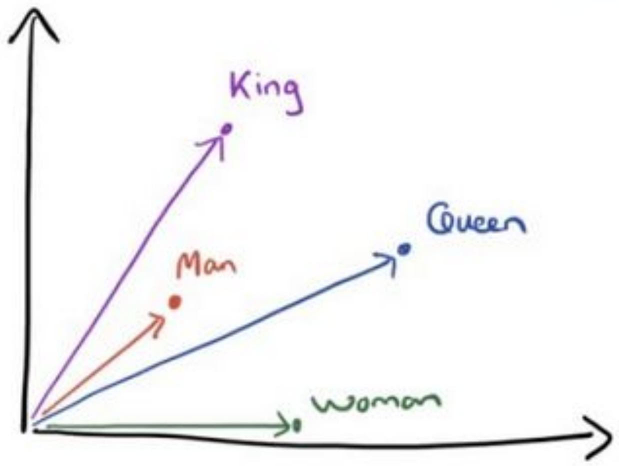
Lots of matrix multiplications, with some activation and functions such as layernorm.

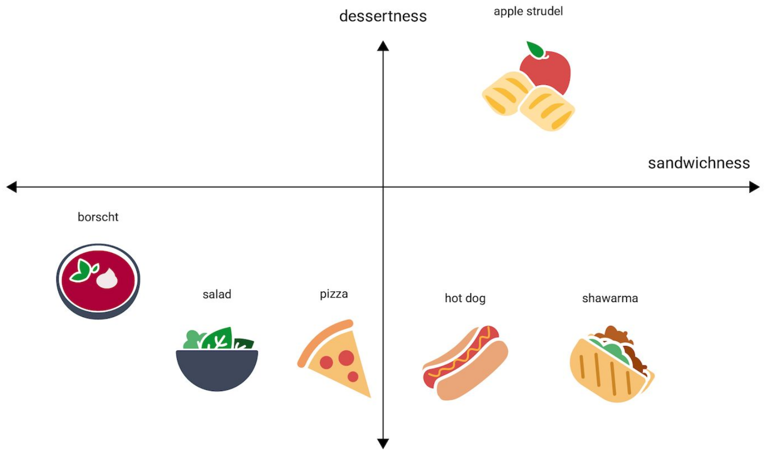


LLMs: Word Embedding

In NLP, tokens and semantics are often encoded as vectors (let us assume token = word). Meaning is then attached to different directions, and vectors pointing in similar directions have semantic relation. These parameters are also trained.

In GPT-2 Small, tokens are embedded in a 768-dimensional space





Tokenization

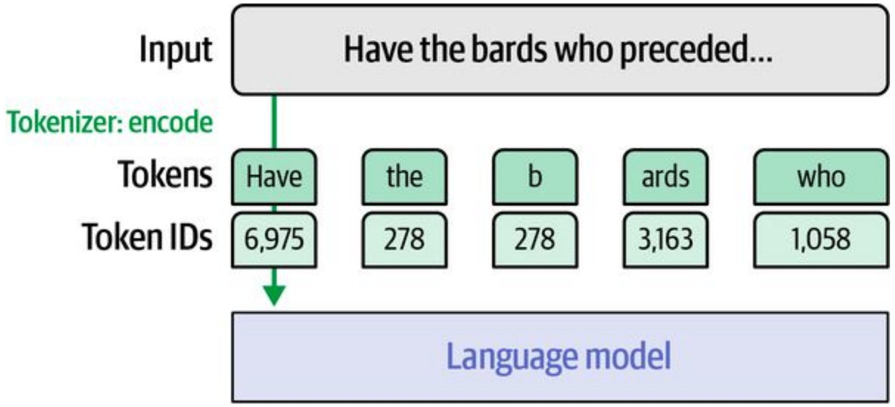
Operation: Split a string into smaller substrings, which each correspond to a numeric token value.

Purpose: Convert human language input into numbers for easier processing

Performance: Varies based on tokenization scheme, but minimal performance impact

Can be quite nuanced; several complex tokenization strategies exist, and the choice can have implications on model behavior.
(i.e. GPT-2 uses byte-pair encoding)

Token tables acts as dictionary.



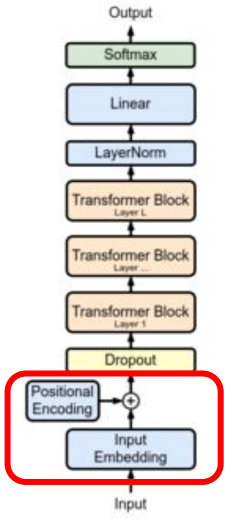
Encoding (embedding)

Operation: Turn a string of tokens into its word vector embeddings (semantics), and also encodes token positional information (sentence structure)

Purpose: Embed the tokens into a higher dimensional space so that linear transformations and other mechanisms take meaning.

Need more information than just corresponding token IDs.

Performance: Once again, negligible time taken as only performed once per forward pass.

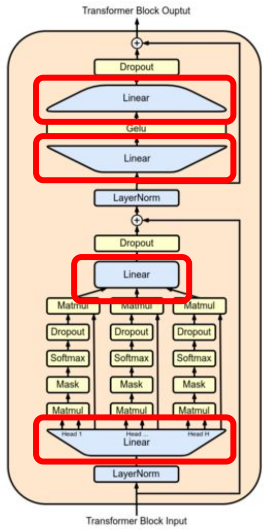


Matrix Multiplication

Operation: Standard matrix multiplication, have talked about this a lot before

Purpose: Performs the fully connected layer computations, de-embedding, and parts of the attention mechanism itself

Performance: Significant impact, large proportion of computation with many matrix multiplications per forward pass.

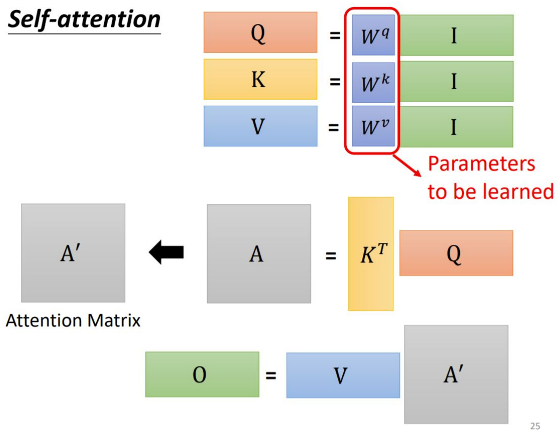


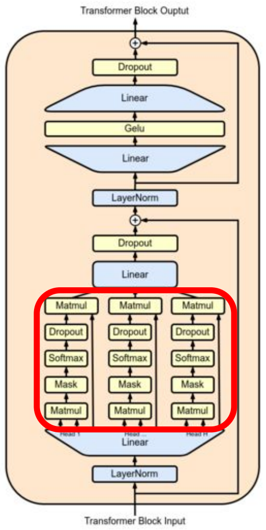
Attention

Operation: Essentially comprises of a series of matrix multiplications. “Looks up” key, query, and value vectors, takes the pairwise dot product between key and query, performs a masked softmax and scales the value vector accordingly.

Purpose: Allows tokens to attend to each other, providing context across the text.

Performance: significant impact, large proportion of computation



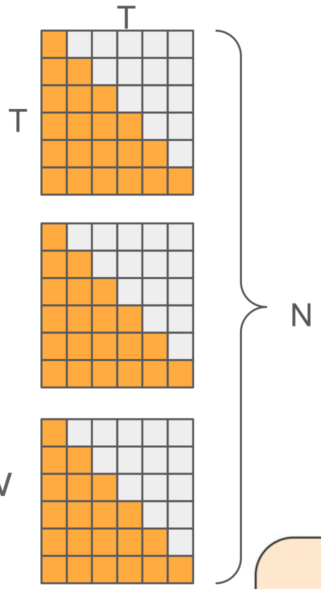


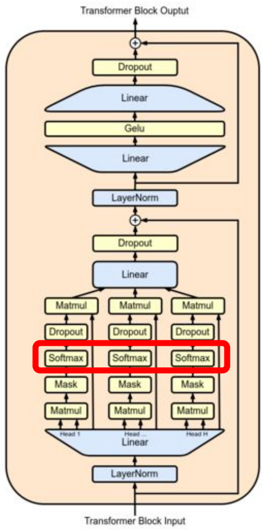
Softmax

Operation: Reduction-type computation. Performs the softmax on the first t elements of each row t modulo T, which converts a list of numbers into a probability distribution

Purpose: Used in attention mechanism, causal masking to allow for autoregressive next token generation. Softmax to allow certain tokens to attend to others much more strongly

Performance: medium impact



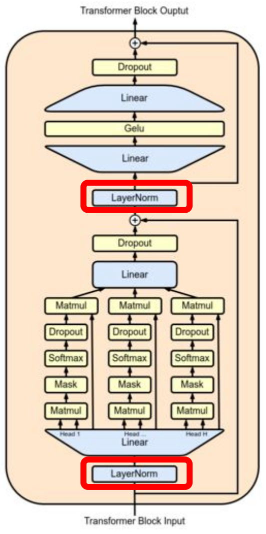


Layernorm

Operation: Reduction-type operation. For each row, calculate the mean and standard deviation, and normalize the row to a different mean and standard deviation.

Purpose: maintain numerical stability within the model

Performance: medium impact

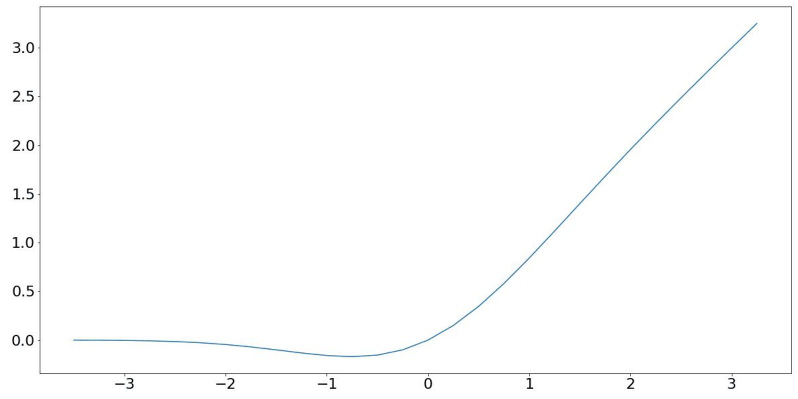


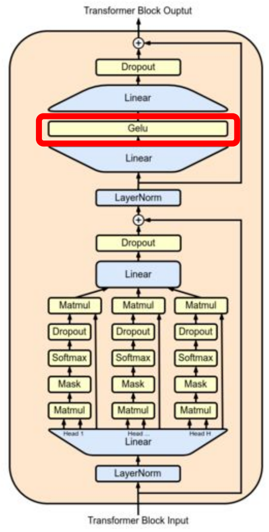
GeLU

Operation: Apply a Gaussian Error Linear Unit to each element in the input

Purpose: Used in the fully connected layers, introduces non-linearity into the model to capture more complex patterns

Performance: small impact



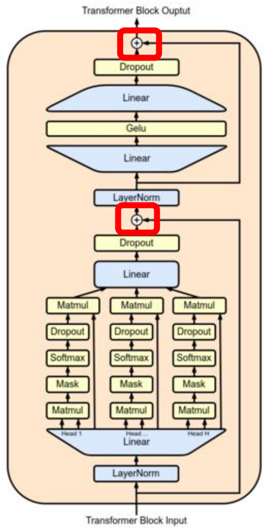


Residual

Operation: Vector addition

Purpose: Preserve previous token meaning to the next stages

Performance: small impact



GPT Project Overview

Milestone 1: implement basic GPU kernels (given CPU implementation)

Milestones 2 & 3: Profiling & Optimizations (the bulk of your project)

Main goal is to optimize the kernels to run as fast as possible

You will be required to implement certain optimizations, as well as some that you figure out yourselves! (profiling will give you guidance)

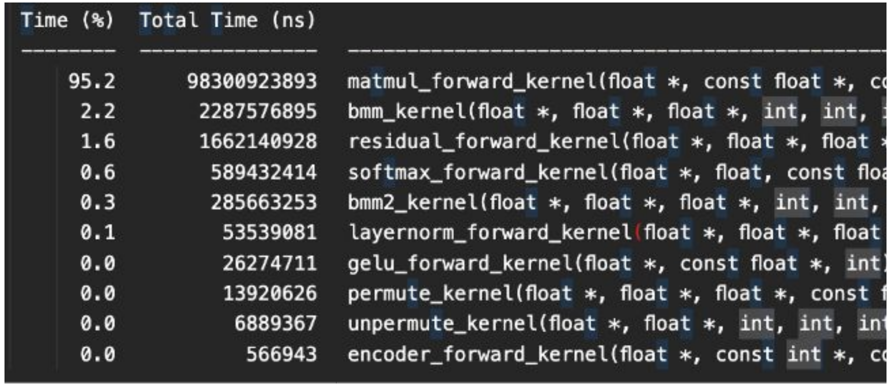
Some (friendly) competition

Profiling

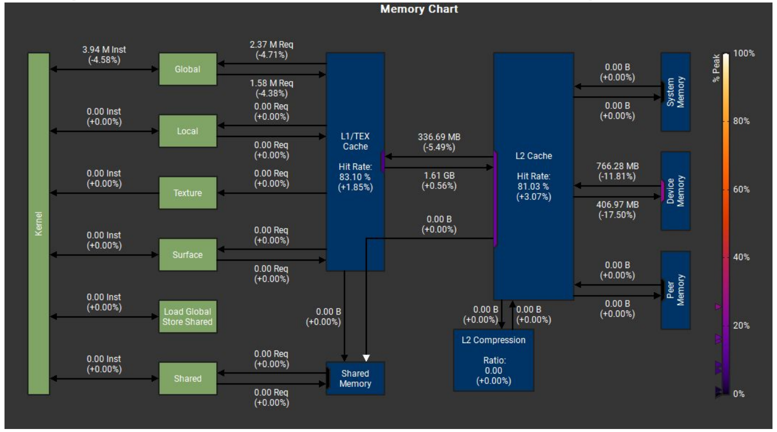
Profiling gives us iNsight into how our CUDA code performs to guide us towards meaningful optimizations.

Nsight Systems shows higher-level kernel launch & execution times, API calls, host-device interactions, etc.

Nsight Compute provides detailed device performance metrics: occupancy, memory bandwidth, cache usage & hit rate, warp stalling, etc.



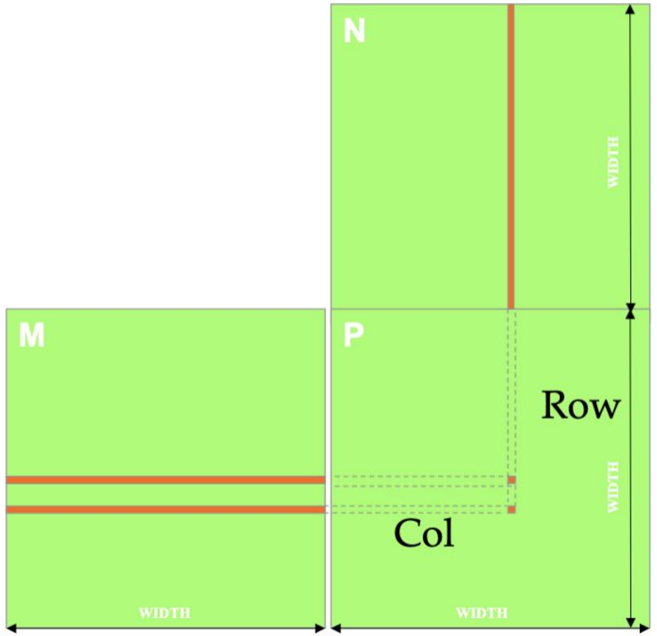
```
Time (%)  Total Time (ns)
--------  ---------------  ------------------------------------------------
    95.2      98300923893  matmul_forward_kernel(float *, const float *, co
     2.2       2287576895  bmm_kernel(float *, float *, float *, int, int, i
     1.6       1662140928  residual_forward_kernel(float *, float *, float *
     0.6        589432414  softmax_forward_kernel(float *, float, const floa
     0.3        285663253  bmm2_kernel(float *, float *, float *, int, int,
     0.1         53539081  layernorm_forward_kernel(float *, float *, float
     0.0         26274711  gelu_forward_kernel(float *, const float *, int)
     0.0         13920626  permute_kernel(float *, float *, float *, const f
     0.0          6889367  unpermute_kernel(float *, float *, int, int, int
     0.0           566943  encoder_forward_kernel(float *, const int *, co
```



Register Tiled Matrix Multiplication

Idea: output elements that are part of the same row use the same element from M

One thread computes multiple output elements in the same row -> can store the M value in registers



Register Tiled Matrix Multiplication

Store weight tile in shared memory

Store elements of input and output in registers

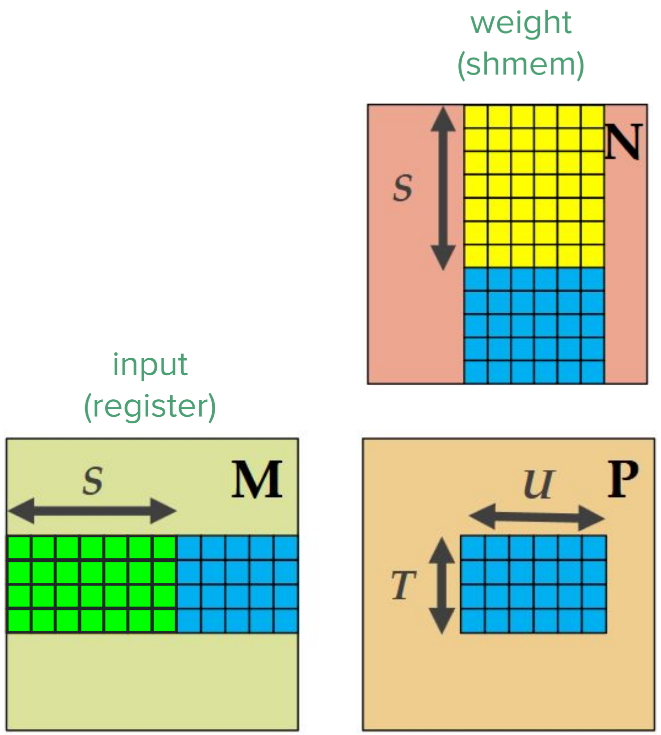


**Illustration taken from ECE 508

Register Tiled Matrix Multiplication

Store weight tile in shared memory

Store elements of input and output in registers

```
S = T/U;
float partial_sums[U] = {0.0f};
__shared__ float weight[S][U];
for (phase = 0...(C/S)) {
    weight[tx/U][tx%U] = N[phase*S][bx*U+tx%U];
    __syncthreads();
    for (i = 0...S) {
        float elem = M[by*T+tx][phase*S+i];
        for (e = 0...U) {
            partial_sums[e] += elem * weight[i][e];
        }
    }
    __syncthreads();
}
```

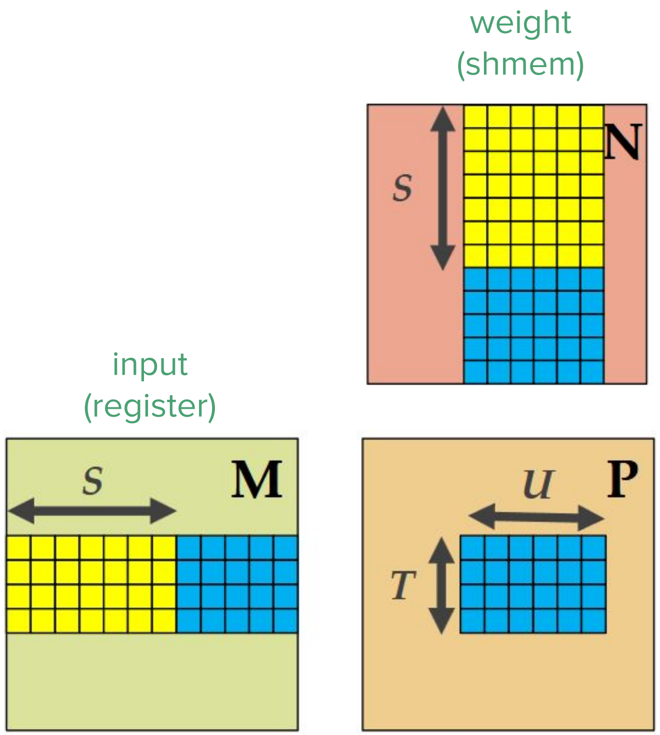


**Illustration taken from ECE 508

Questions?