NLP 202: Deep Learning on GPUs

Jeffrey Flanigan

Jan 10, 2023

University of California Santa Cruz
jmflanig@ucsc.edu

Administrative

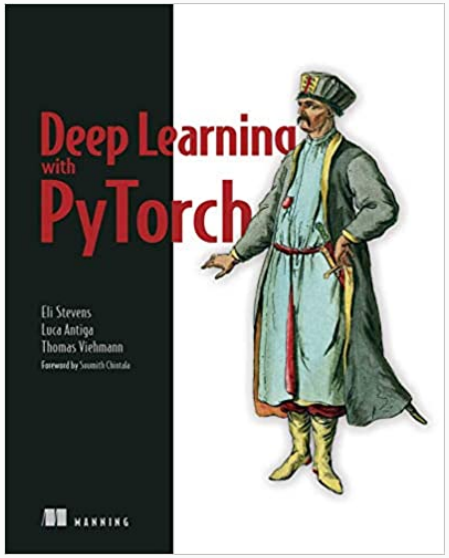
- Instructors: Jeff (instructor) and Rongwen Zhao (TA)
- Classes will be simultaneously on Zoom and recorded. First two weeks online only
- Assignments will be done either locally or on Google Colab
- We accommodate disabilities. If you require DRC accommodations (`https://drc.ucsc.edu/`), please let me know ASAP

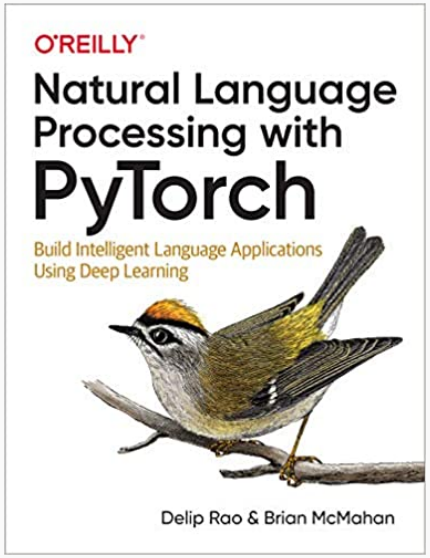
Resources

- Course website: `https://courses.soe.ucsc.edu/courses/nlp202/Winter23`
- Canvas (for videos, exams, assignment turn-in, and some materials)
- NLP wiki: `https://jlab.soe.ucsc.edu/nlp-wiki` Please do not share widely
- No official textbook. Readings posted on Canvas or the website

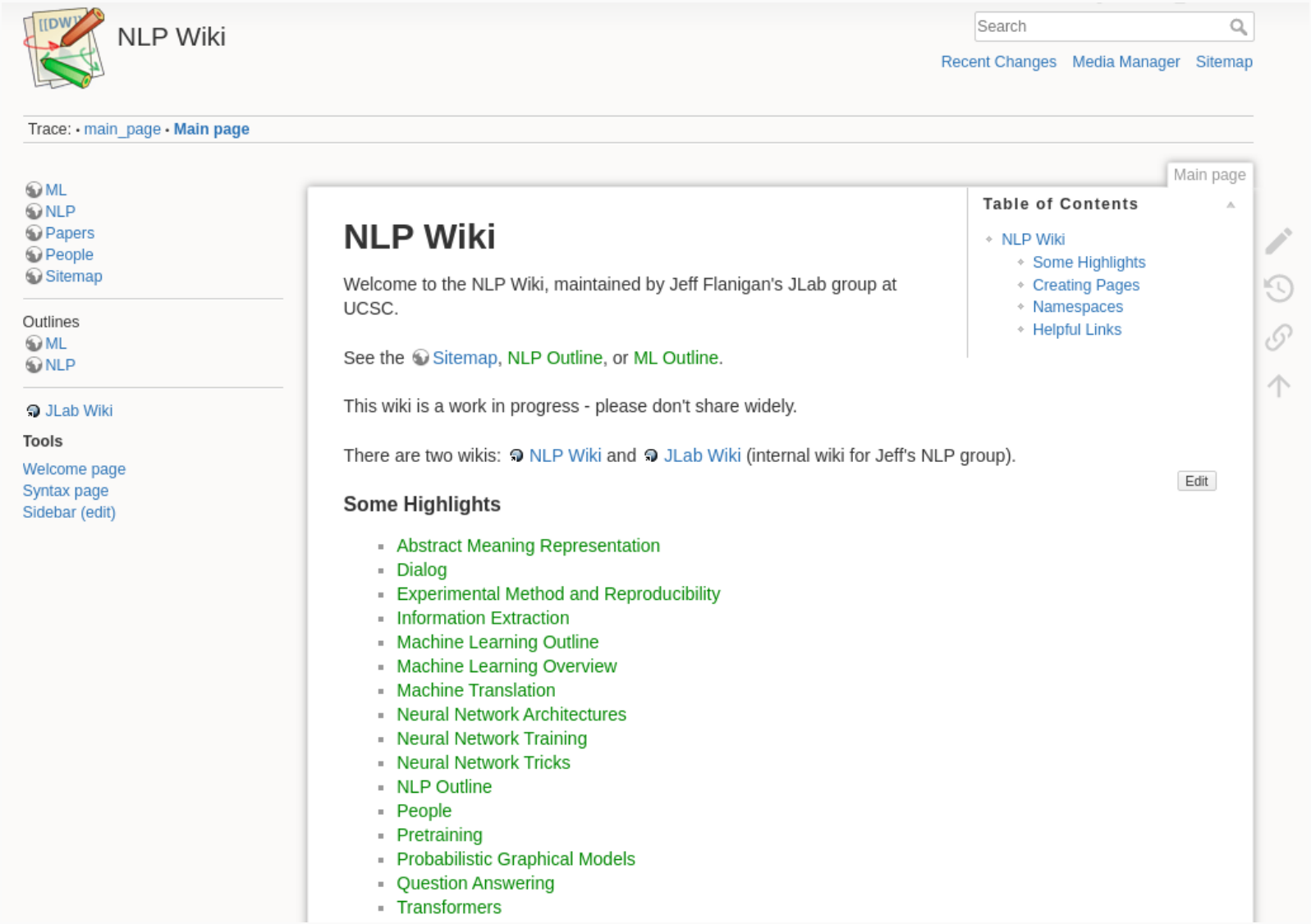
PyTorch Books

Highly recommend these two books





NLP Wiki


NLP Wiki
Search
Recent Changes
Media Manager
Sitemap
Trace: • main_page • Main page
ML
NLP
Papers
People
Sitemap
Outlines
ML
NLP
JLab Wiki
Tools
Welcome page
Syntax page
Sidebar (edit)
Main page
Table of Contents
NLP Wiki
Some Highlights
Creating Pages
Namespaces
Helpful Links
NLP Wiki
Welcome to the NLP Wiki, maintained by Jeff Flanigan's JLab group at UCSC.
See the Sitemap, NLP Outline, or ML Outline.
This wiki is a work in progress - please don't share widely.
There are two wikis: NLP Wiki and JLab Wiki (internal wiki for Jeff's NLP group).
Edit
Some Highlights
Abstract Meaning Representation
Dialog
Experimental Method and Reproducibility
Information Extraction
Machine Learning Outline
Machine Learning Overview
Machine Translation
Neural Network Architectures
Neural Network Training
Neural Network Tricks
NLP Outline
People
Pretraining
Probabilistic Graphical Models
Question Answering
Transformers

Evaluation

- 4 assignments (A1–4), completed individually (45%)
  You get a 24 hour grace period to turn it in late
- Quizes (10%) weekly on Canvas
- Midterm exam (15%), towards the middle of the quarter on Canvas
- Final exam (25%), to take place at the end of the quarter
- Attendance / Participation (5%) For attending each class and participating

Academic Integrity

- Assignments and tests are to be completed individually
- Do not look up or copy either code or solutions from others or the internet
- My assignments are designed to aid learning
- They are not designed to prevent copying
- I expect you to take responsibility for your learning

You are here to learn: plagiarism will only hurt **you**

Assignment "redos"

- Each student gives feedback on two assignments
- Three days for feedback
- One week for changes for final submission which TA grades
- Writeup explanation of what you changed, and where you learned it (This is important. You get marked down without it)
- We check if you copy

For the final deadline, assignments may be turned in up to 24 hours after the deadline for a 10% grade penalty. After 24 hours, assignments will receive zero credit.

Outline of NLP 202

Syntax and Advanced topics

1. **Deep learning on GPUs**
2. **Syntax and parsing**
3. **Structured prediction and loss functions**
4. **Dependency Parsing**
5. **Advanced Decoding**
6. **Optimization for deep learning**
7. **Neural network tricks**

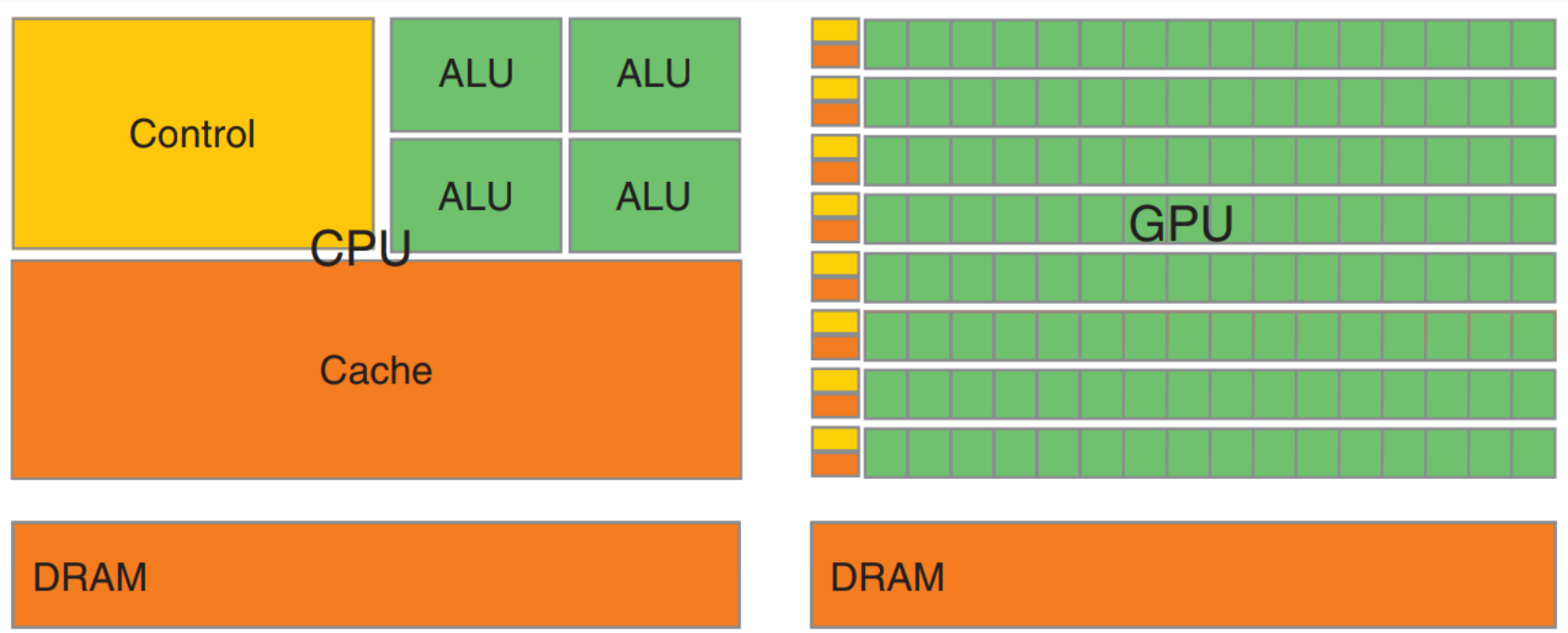
Plan for Today

- GPU computing
- Minibatching

Parallelization for Deep Learning

- Train deep learning models quickly $\rightarrow$ use parallelization
- Can train with CPUs, but **GPUs** are much faster
- I will focus on parallelization on GPUs
- Usually, code written for parallelization on GPUs will also run faster on CPUs

CPU vs GPU



- CPU: 10's of threads, 100's of GigaFLOPS (FLOPS = Floating point operations per sec)
- GPU: 10,000's of threads, 10-100's of TeraFLOPS

Why study GPUs

- We won't be programming GPUs directly
- But: knowledge of how GPUs operate will help us write efficient code
- Similar concepts apply to Tensor Processing Units (TPUs)

Programming GPUs

- Most popular language for programming GPUs is CUDA (NVidia)
- Deep learning libraries use math libraries for efficient implementations of common functions (such as Eigen, NNPack, Intel MKL, or Intel cuDNN)
- You can write your own CUDA code (called CUDA kernels) to run on the GPU $\rightarrow$ need to write CUDA code for forward and backward

GPU Memory Model

Device code can:

- R/W per-thread registers
- R/W per-thread local memory
- R/W per-block shared memory
- R/W per-grid global memory
- Read only per-grid constant memory

Host code can

- Transfer data to/from per grid global and constant memories

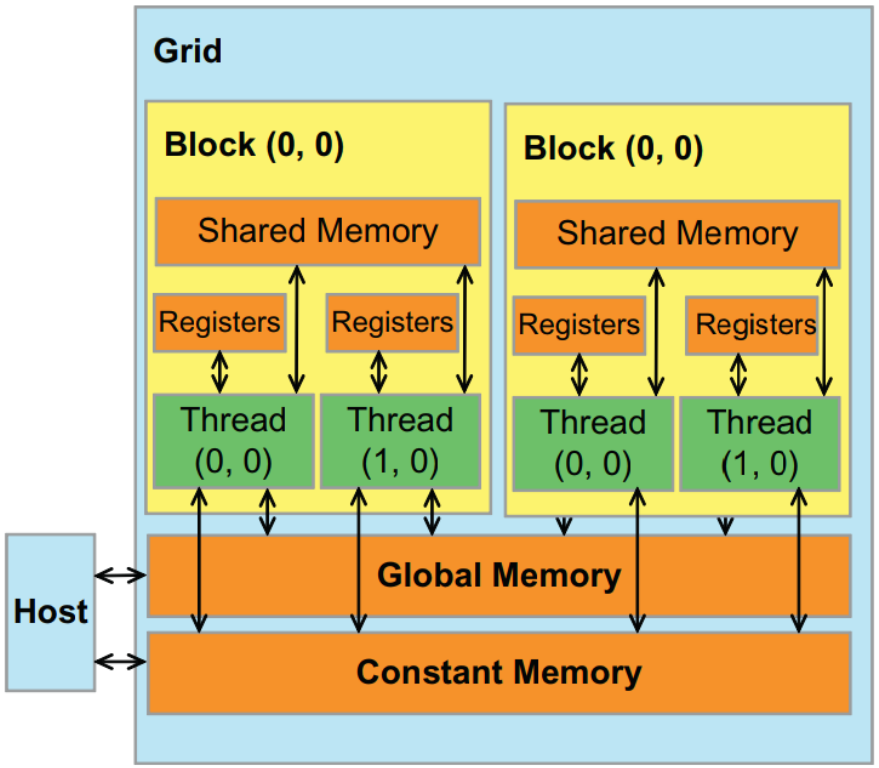


FIGURE 5.2

Overview of the CUDA device memory model.

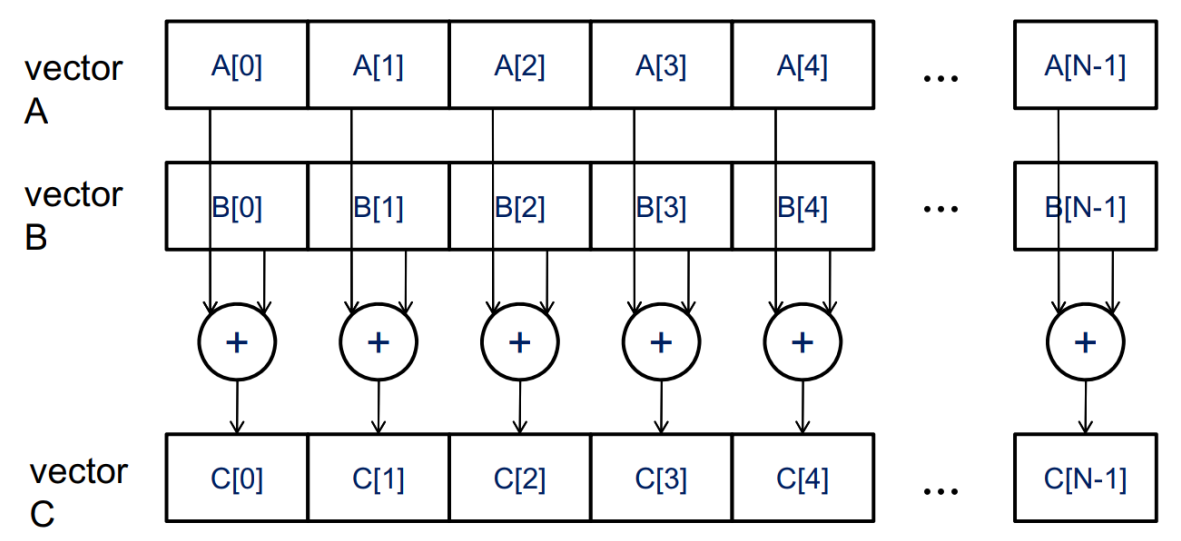
Memory Bandwidths and Sizes for GeForce RTX 3090

- Registers (20 MB): full speed
- GPU memory (24 GB): 1000 GB/sec (BW_{mem})
- Host to GPU: PCIe 4.0 x16 = 31 GB/sec

Single precision math bandwith (ops/sec): 35 TFLOPS
Memory bandwith required to keep up with ALU (BW_{arith}): 35*4 = 140 TB/sec
Using RTX 3090's tensor cores, it's even worse: 235 TFLOPs
FP16 $\rightarrow$ 235*2 = 470 TB/sec

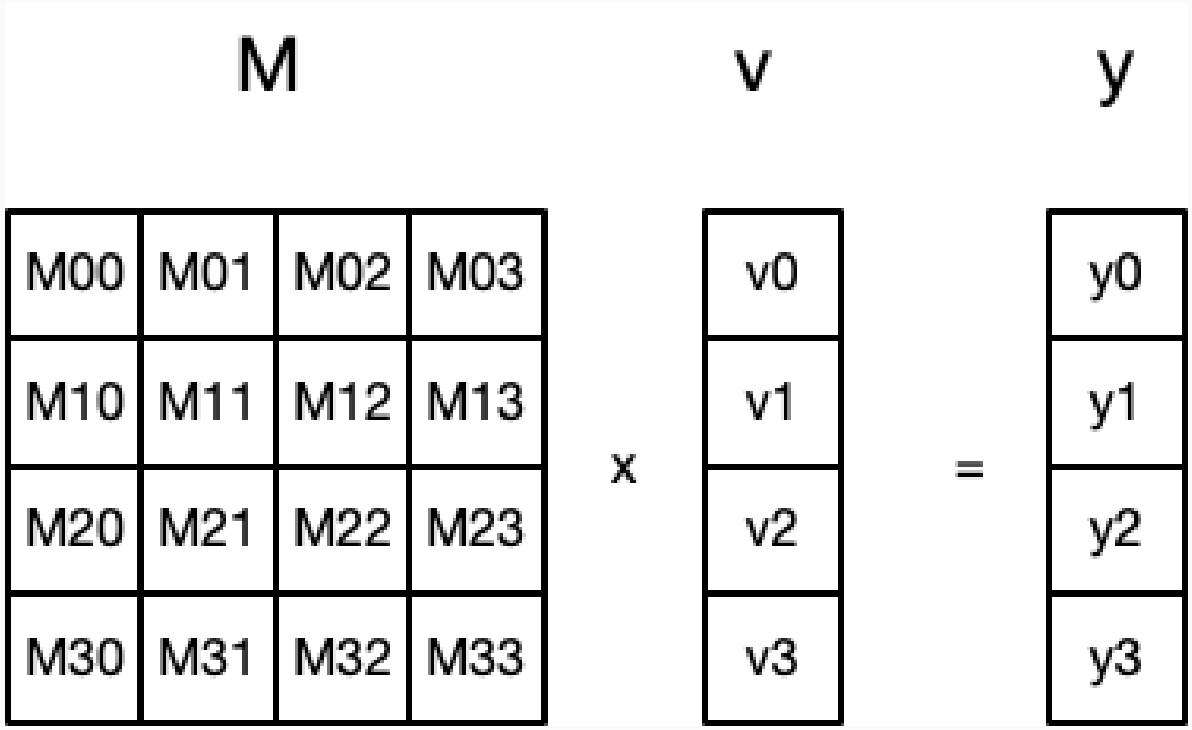
Ratio of arithmetic performance to memory limited performance of 140:1 or 470:1

Parallel Vector Addition



If vectors are in registers (<20MB in size) we get 35 TFLOPS. Otherwise, we load both vectors from GPU memory and get 35/(140*2) = .125 TFLOPS.

Vector-Matrix Multiply



$$y0 = M_{00}v_0 + M_{01}v1 + M_{02}v_2 + M_{03}v3$$

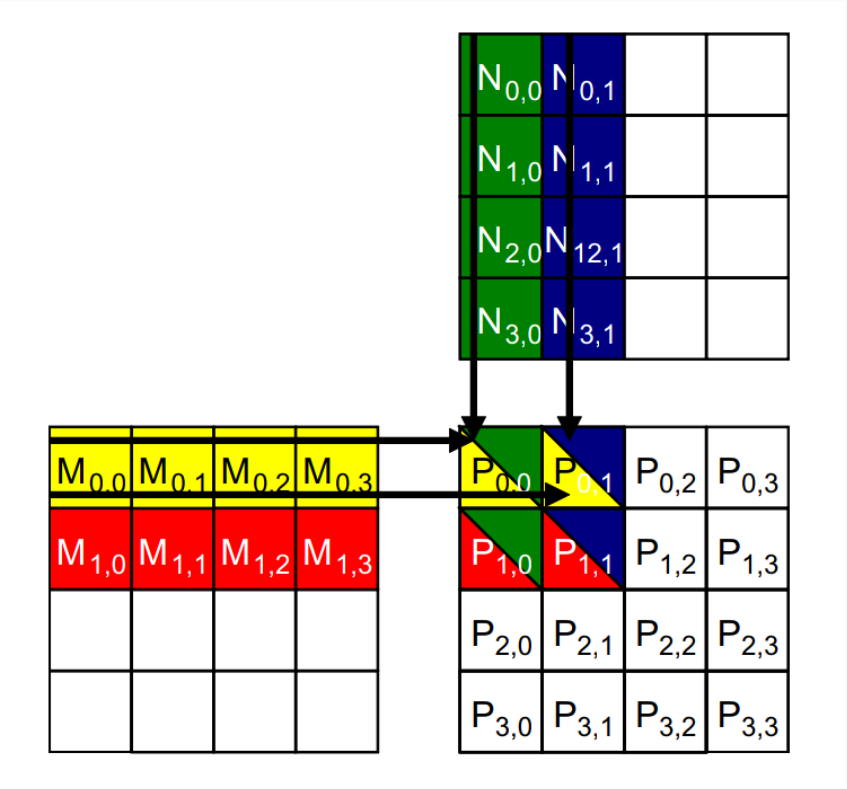
$$y1 = M_{10}v_0 + M_{11}v1 + M_{12}v_2 + M_{13}v3$$

None of the elements of M are re-used in the computation.

If M is in GPU memory, this will be slow.

Matrix-Matrix Multiply

$P = M \times N$



In this case, elements of M and N are re-used 4 times.

If M,N in memory, this gives us .125 TFLOPS * 4 = .5 TFLOPS

Arthmetic Intensity

- **Arithmetic intensity** of an operation
  = floating point operations / bytes loaded
- To make use of all the ALUs, we want arithmetic intensity $\times \text{bytes per float} > BW_{arith}/BW_{memory}$
- The larger the matrices, the more operations per byte loaded from memory.
- Arithmetic intensity of matrix multiplication AB = C, where A is M x K, B = K x N, C = M x N:

$$\text{Arithmetic intensity} = \frac{M \cdot N \cdot K}{M \cdot K + N \cdot K + M \cdot N} \cdot \frac{\text{Float}}{\text{bytes}}$$

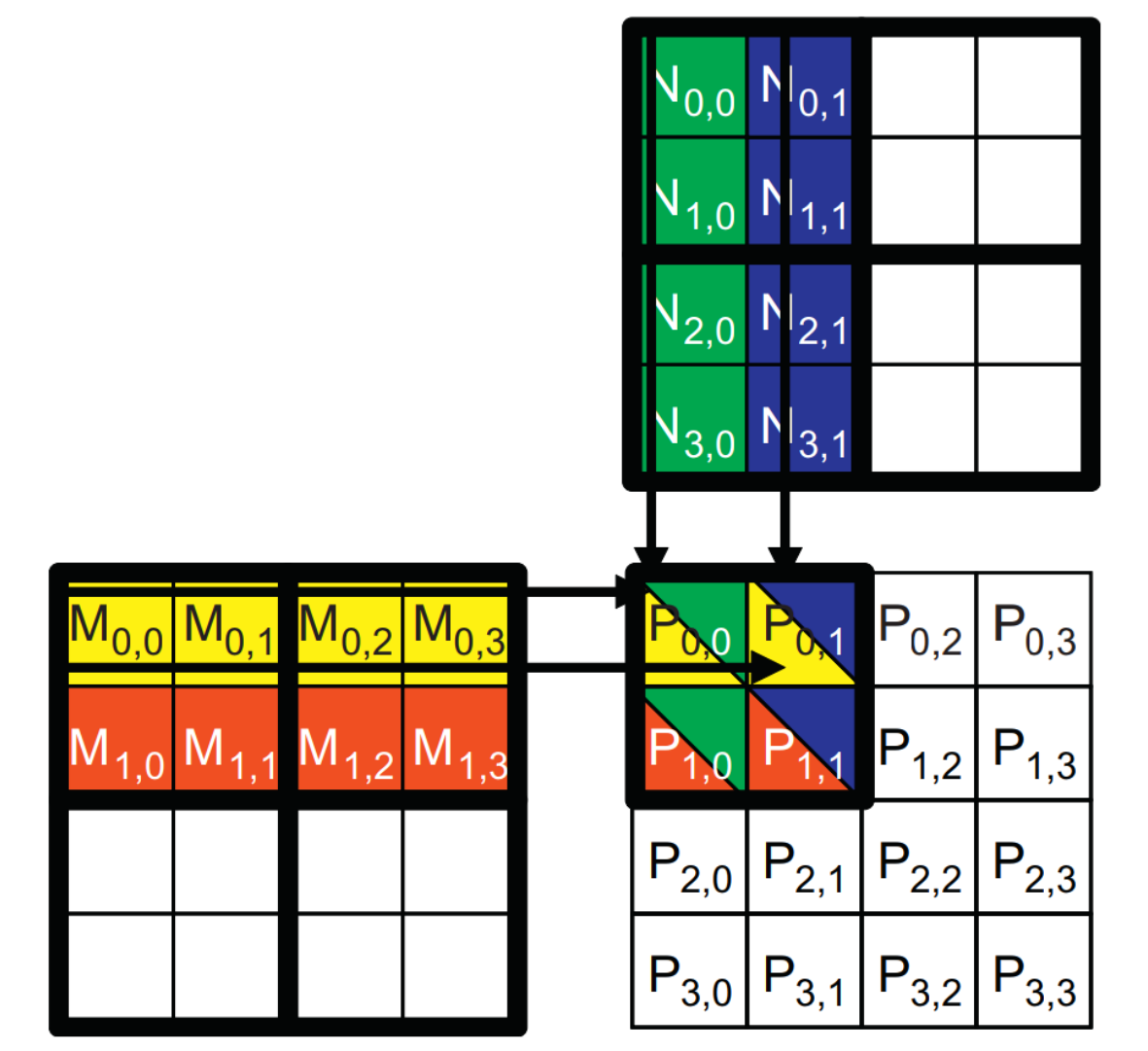
Arthmetic Intensity Example

- M x N x K = 8192 x 128 x 8192, with FP16 floats $\rightarrow$ $8192 \cdot 128 \cdot 8192 / (8192 \cdot 128 + 8192 \cdot 128 + 8192 \cdot 8192) \cdot \frac{1}{2} =$ 31.0 FLOPS/byte
- This is lower than RTX 3090's 470 FLOPS/bytes, so we are memory bandwidth limited

Tiling

- Larger matrices have higher arithmetic intensity
- Square matrices have higher arithmetic intensity
- We can't fit large matrices into the registers (too small)
- Solution: tiling

Tiling


N0,0
N0,1
N1,0
N1,1
N2,0
N2,1
N3,0
N3,1
M0,0
M0,1
M0,2
M0,3
M1,0
M1,1
M1,2
M1,3
P0,0
P0,1
P0,2
P0,3
P1,0
P1,1
P1,2
P1,3
P2,0
P2,1
P2,2
P2,3
P3,0
P3,1
P3,2
P3,3

Memory Access Patterns

- Like CPUs, GPUs work best if memory access is contiguous
- They access memory in large (512 byte) chunks
- One row of 256 element matrix (1024 bytes) takes 2 load operations, but a 257 (1028 bytes) takes 3 load operations. This is called a "quantization effect"

Data Parallelism with Minibatching

- A feedforward NN or RNN has many vector-matrix multiplies
- As we have seen, there is limited speedup for vector-matrix multiplies on GPUs
- Solution: mini-batching
    - Our vectors become matrices, where one dimension is the batch
- This is an instance of **data-parallelism** - running the model in parallel over many data points

Minibatching

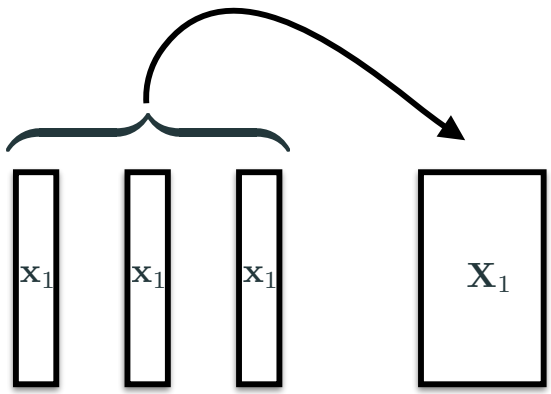
Single-instance RNN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

Minibatch RNN

$$\mathbf{H}_t = g(\mathbf{V}\mathbf{X}_t + \mathbf{U}\mathbf{H}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{Y}}_t = \mathbf{W}\mathbf{H}_t + \mathbf{b}$$

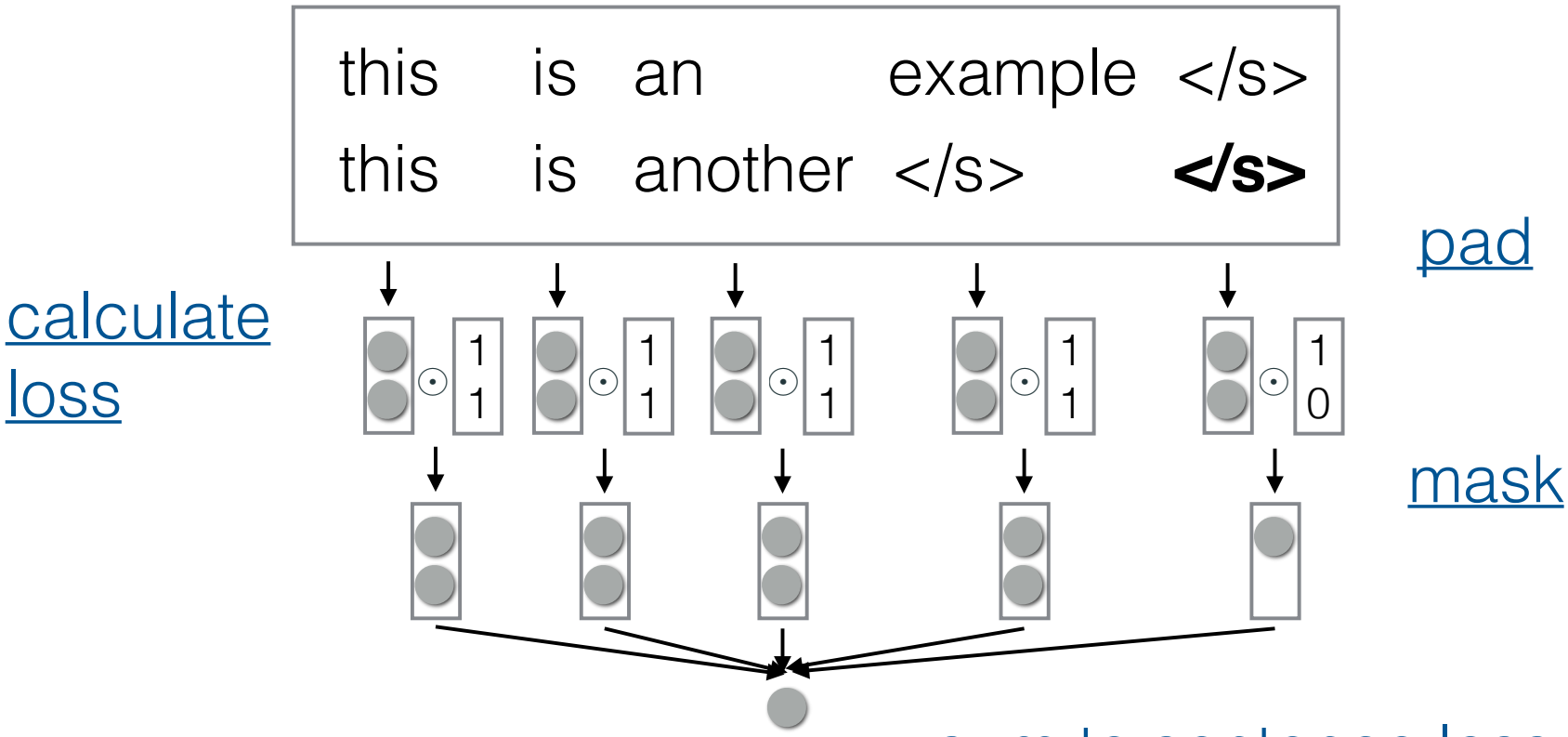
anything wrong here?



We batch across instances, not across time.

Minibatching Sequences

- How do we handle sequences of different lengths?



Example: Estimating Minimum Batch Size

- BW_{arith}/BW_{memory} roughly tells us the minimum batch size for maximum parallelization for FF or RNNs
- Example: GeForce RTX 3090 using tensor cores: $BW_{arith}/BW_{memory} = 470$. Thus, we want a batch size of at least 470 for maximum performance when training
- Can use this to estimate the GPU memory size needed to train a model on a given GPU most efficiently

Calculating BW_{arith}/BW_{memory}

- Be sure to use the same units for BW_{arith} and BW_{memory}
  $\frac{Bytes/sec}{Bytes/sec}$
- For GeForce RTX 3090 using tensor cores (235 TFLOPS FP16):
  BW_{arith} = 235 Tera FP16/sec * 2 Bytes/FP16 = 470 TB/sec
  GPU Memory bandwidth: BW_{memory} = 1TB/sec
  $BW_{arith}/BW_{memory} = 470$

Recommendations

- Larger batch sizes and neuron counts improve parallelization and efficiency
- Choose batch sizes and neuron counts greater than 128 (and in multiples of 128) to avoid quantization effects hurting performance

Data parallelization vs Model parallelization

- We have talked about batch (data) parallelization
- There is also **model parallization** possible for some models
- The Transfomer is model parallel (one of the reasons it was invented)