Cloud and Machine Learning

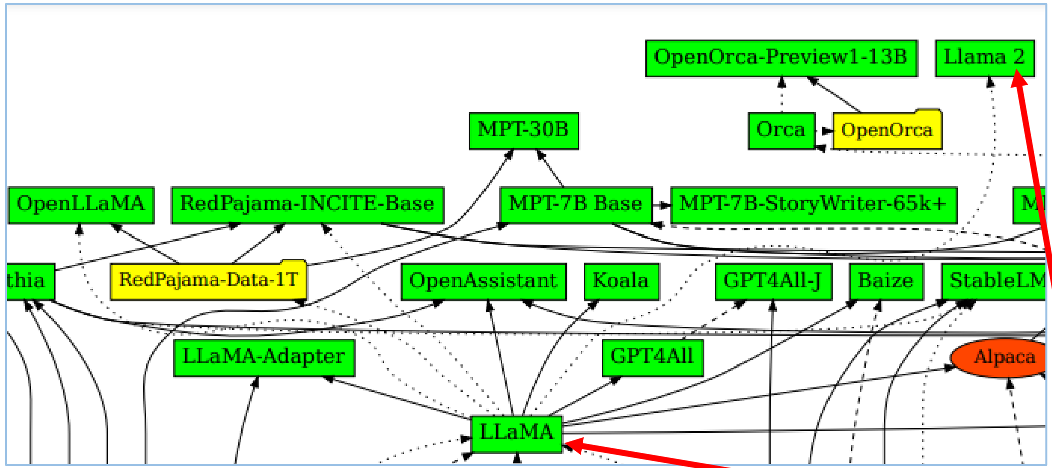
CSCI-GA.3033-085 Spring 2024

Prof. Hao Yu
Prof. I-Hsin Chung

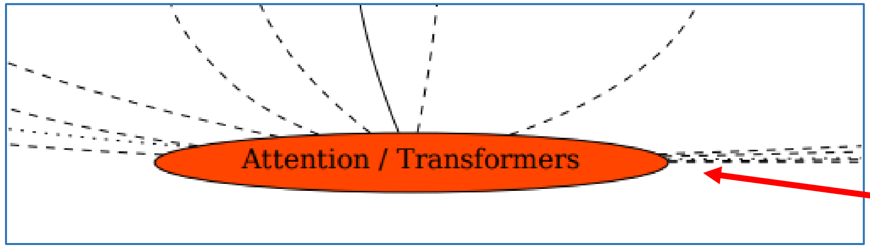
Lecture 7: Performance Analysis

NLP/LLM/FM explosive development

- An explosive 7-year development of LLM models
  - https://github.com/rain-1

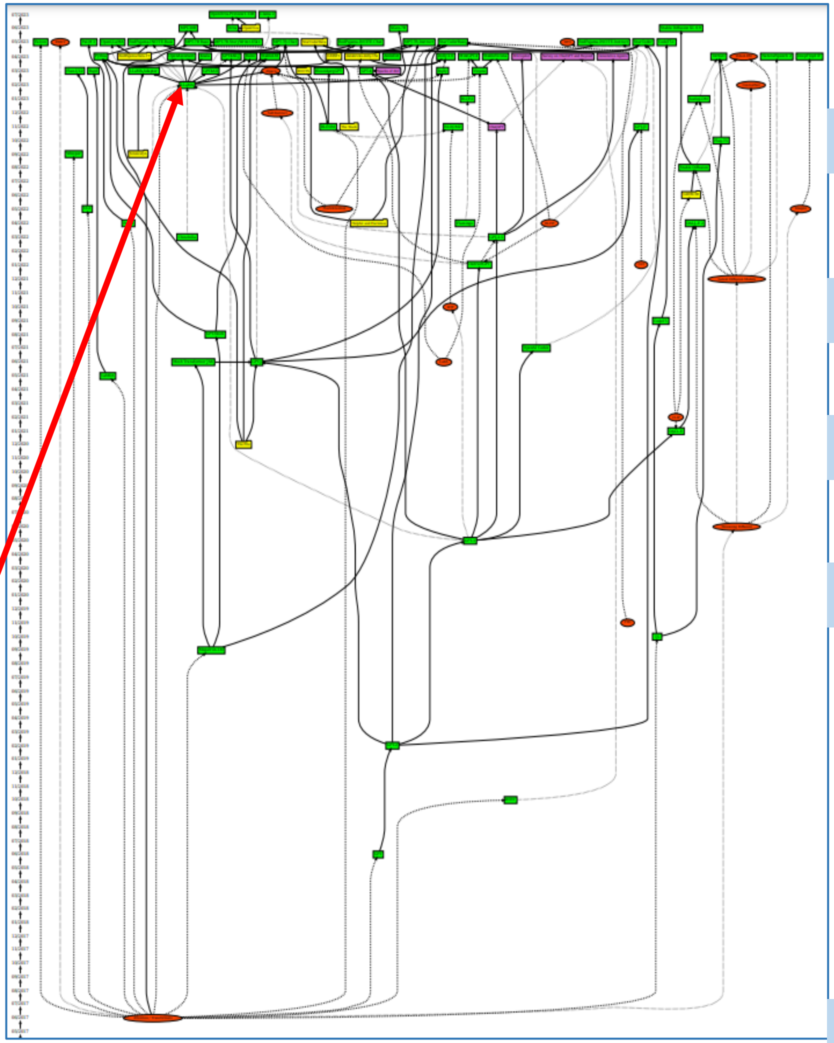


Current focus



Starting point

exponential growth



2023

2022

2021

2020

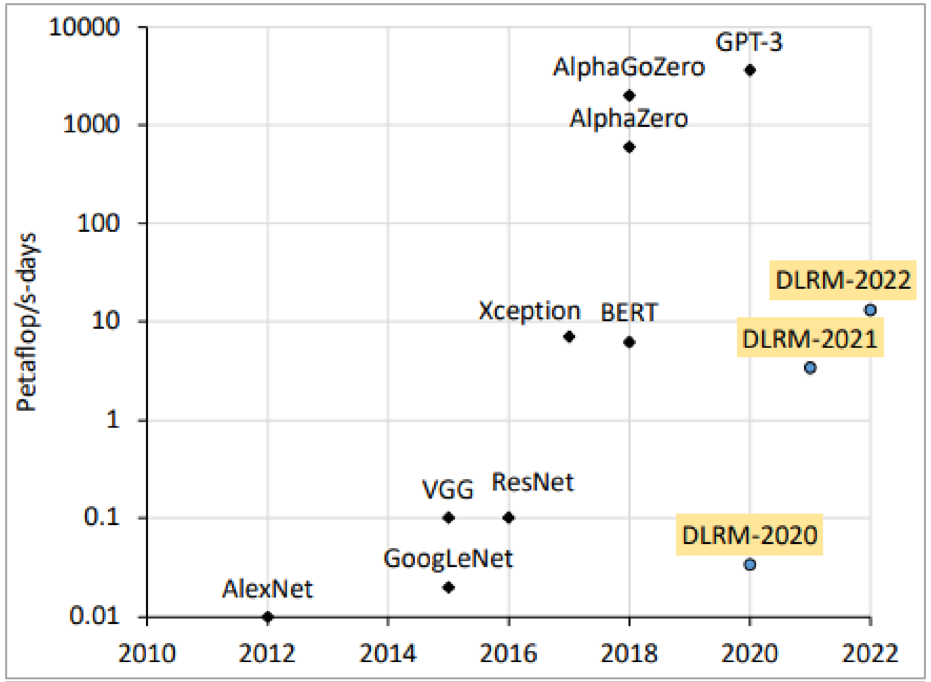
2017

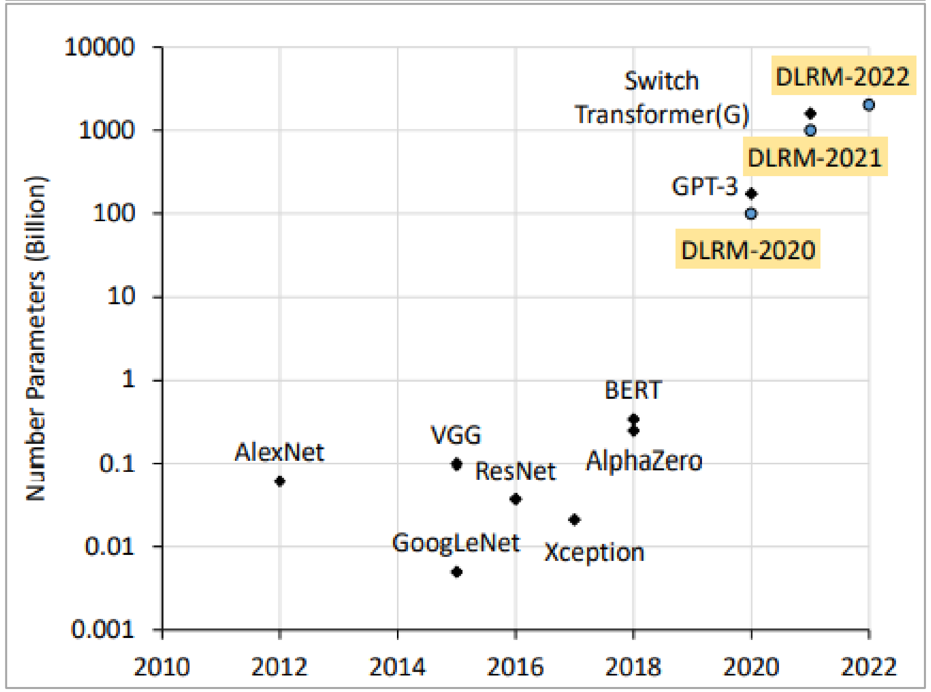
Model Size Scaling

The scaring trend in 2022

Model Name	# tunable params	Primary precision	Type	Organization	Open source available	Accelerator types	Announcing publishing dates
BERT	340 M		Dense	Google	y		Nov-18
GPT3	175 B		Dense	Open AI	N		Jun-20
Jurassic	178 B		Dense	AI21			Aug-21
Gopher	280 B		Dense	DeepBrain			Jan-22
MT-NLG	530 B		Dense	Nvidia Microsoft	y		Feb-22
LaMDA	137 B		Dense	Google	n	TPU v4	Feb-22
Chinchilla	1.4 T, 70 B		Sparse	DeepBrain	n		Mar-22
OPT	175 B		Dense	Meta	y (gh, hf)		Jun-22
BLOOM	176 B		Dense	BigScience	y (hf)		Aug-22
GLM	130 B		Dense	Tsinghua	y	GPU	Aug-22
GLaM	1.2 T		Sparse	Google	n	TPU v3	Aug-22
PaLM	540 B		Dense	Google	n	TPU v4	Oct-22
MoE (meta)	1.1 T		Sparse	Meta	y (gh)		Nov-22

Nov. 2022





[June 2022 DLRM co-design, ISCA]()

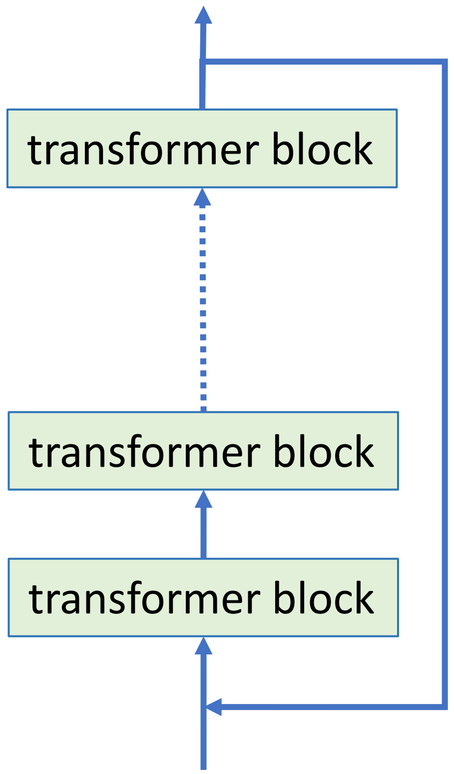
Foundation Model: the naming

- **Foundation model (FM)** Definition (wiki):
  - "a large machine learning (ML) model trained on a vast quantity of data at scale (often by self-supervised learning or semi-supervised learning)[2] such that it can be adapted to a wide range of downstream tasks[3][4] ."
  - By: CRFM, in **On the Opportunities and Risks of Foundation Models, 2021**
- **Transformer model** centered
  - Attention block
  - Repeated transformer blocks
- Developed in **NLP** field with 20 years of slow brewing.
- Ever growing model sizes: **LLM**
  - For a given family models (e.g. gpt2, bert), the predictive capability grows with it model size.

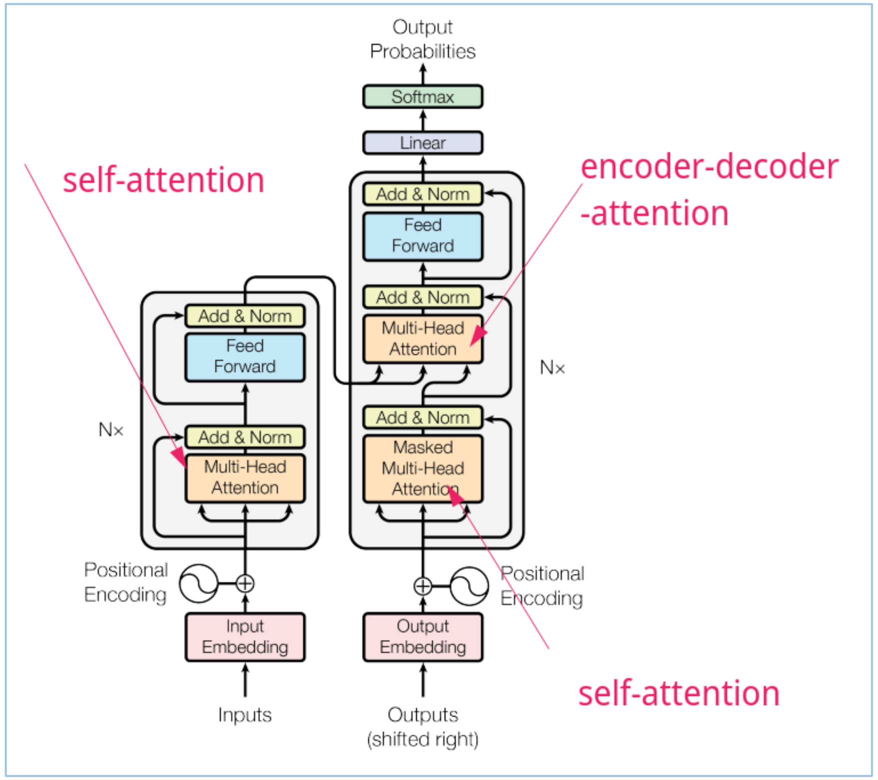
Transformer model architecture

- Repeated transformer blocks, with key configuration parameters:
  - Number of transformer block layers
  - Transformer block: Encoder, decoder, encoder-decoder
  - Embedding (hidden, reduction) dimension size

Typical transformer model



transformer block architecture

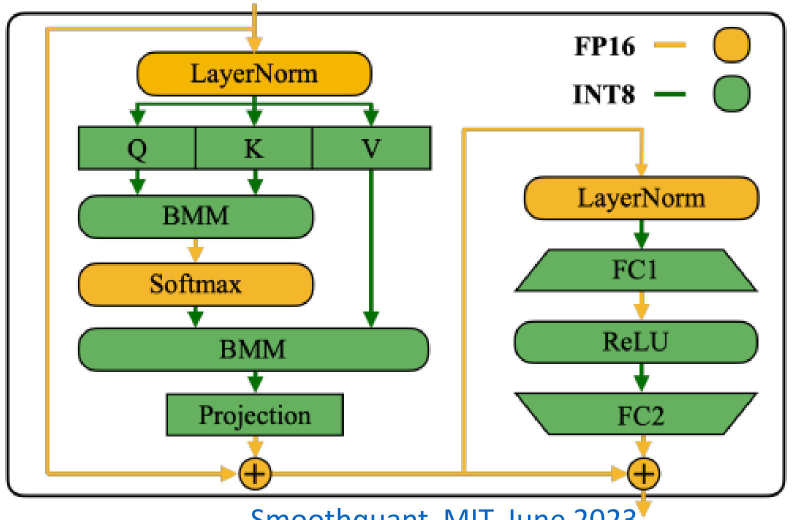


Attention is all you need (oversimplified), Aayush Neupane

Dot-product attention

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^{T}}{\sqrt{d_k}}\right)V$$

Attention is.. , Google, June 2017



Smoothquant, MIT, June 2023

FM Applications

- Extensible usage pattern (typical):
  - Unsupervised **pretraining** over huge amount of data
    +
    Supervised **finetuning** over domain/task specific labeled data
    - Fine tuning, changes model weights
  - Pretrain + multi-shot : prompt engineering
    - Prompt eng. “getting the model to do what you want at inference time by providing enough context, instruction and examples without changing the underlying weights. fine-tuning”
    - Fine tuning vs prompt engineering LLM, Niels Bantilan, may 2023
- Application/Tasks
  - NLP tasks for training/tuning: language modeling, QA, reading, sentiment, paraphrasing.
  - In more than NLP: translation, summarization, writing, image segmentation, bio-sequence, coding, etc.

Transformer models in practice

- HF/transformers:
  - transformer-based NN architectures + pretrained models.
- See the architecture: FX Graph, NSYS, torch-profiler, model-print, abstract code,
  - Transformer explained
  - Terms: encoder, decoder, encoder-decoder, position embedding (where the words are in the input sequence),

Transformer models in practice

- HF/transformers:
  - transformer-based NN architectures + pretrained models.

Selected list of pretrained models hosted out of HuggingFace

bert-base-cased	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on cased English text.
bert-large-cased	24-layer, 1024-hidden, 16-heads, 340M parameters. Trained on cased English text.
gpt2	12-layer, 768-hidden, 12-heads, 117M parameters. OpenAI GPT-2 English model
gpt2-medium	24-layer, 1024-hidden, 16-heads, 345M parameters. OpenAI's Medium-sized GPT-2 English model
gpt2-large	36-layer, 1280-hidden, 20-heads, 774M parameters. OpenAI's Large-sized GPT-2 English model
gpt2-xl	48-layer, 1600-hidden, 25-heads, 1558M parameters. OpenAI's XL-sized GPT-2 English model
t5-large	~770M parameters with 24-layers, 1024-hidden-state, 4096 feed-forward hidden-state, 16-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-3B	~2.8B parameters with 24-layers, 1024-hidden-state, 16384 feed-forward hidden-state, 32-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-11B	~11B parameters with 24-layers, 1024-hidden-state, 65536 feed-forward hidden-state, 128-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)

https://huggingface.co/transformers/v3.3.1/pretrained_models.html

Huggingface Transformer Lib

Super easy starting point

Google: huggingface OpenAI gpt2 → https://huggingface.co/transformers/v3.3.1/model_doc/gpt2.html

```
from transformers import GPT2Tokenizer, GPT2Model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2Model.from_pretrained('gpt2')
text = "Replace me by any text you'd like."
encoded_input = tokenizer(text, return_tensors='pt')
output = model(**encoded_input)
```

```
from transformers import pipeline, set_seed
generator = pipeline('text-generation', model='gpt2')
set_seed(42)
generator("Hello, I'm a language model,", max_length=30, num_return_sequences=5)
```

Getting serious?

https://github.com/huggingface/transformers/blob/main/examples/pytorch

- Manage realistic input datasets
- Inference performance boost
  - Float-16 vs flaot-32 (bits)
  - Model computation graph optimization (torch.jit.trace)
- Reproducible and reusable effort (code)

Visual of Transformer Models

```
DistilBertForMaskedLM(
  (activation): GELUActivation()
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (vocab_transform): Linear(in_features=768, out_features=768, bias=True)
  (vocab_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (vocab_projector): Linear(in_features=768, out_features=30522, bias=True)
  (mlm_loss_fct): CrossEntropyLoss()
)
```

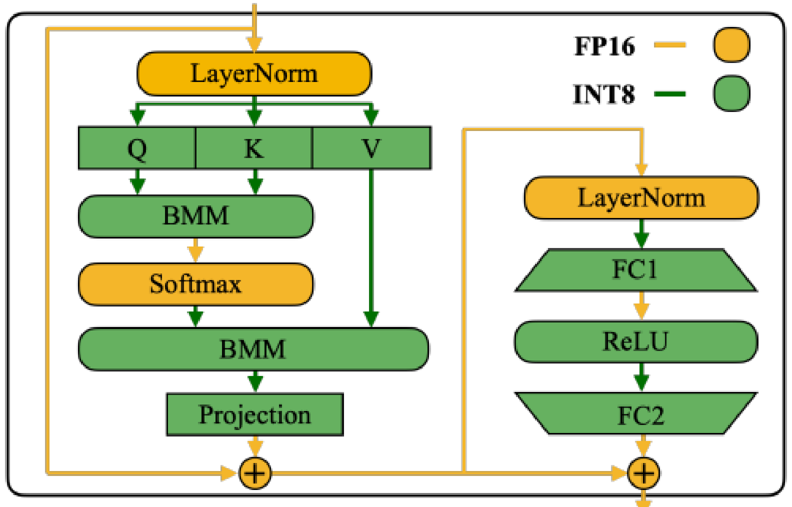
https://huggingface.co/docs/transformers/v4.34.0/en/model_doc/distilbert#transformers.DistilBertForMaskedLM

- Model file only keeps the trained weights (parameters)
- There are no trainbles for softmax BMM, GELU layer. Some time model print won't show.

Get the visual memory

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{T}}{\sqrt{d_k}}\right)V$$

Attention is.. , Google, June 2017



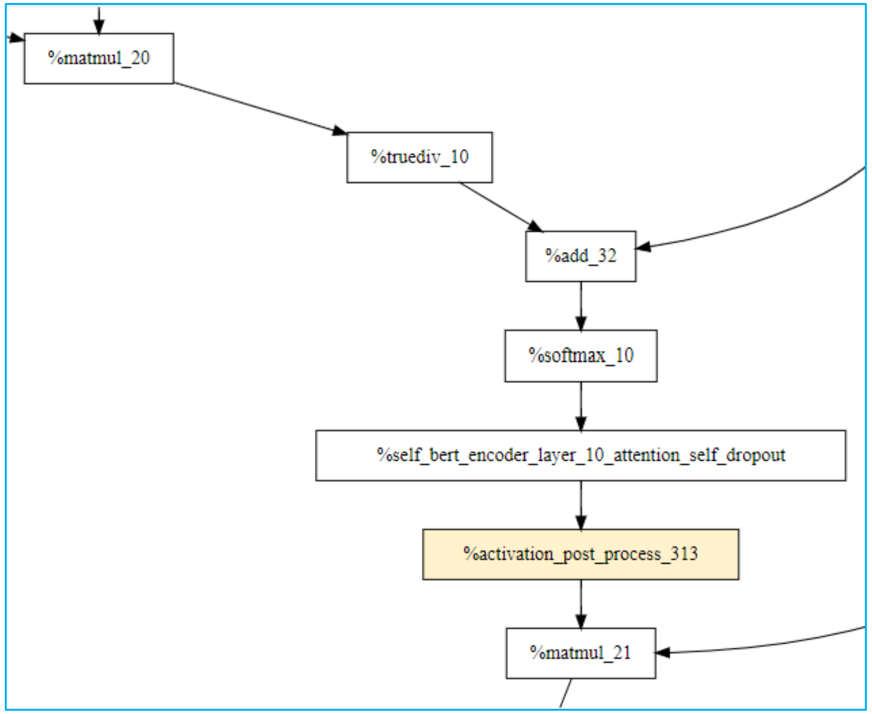
Smoothquant, MIT, June 2023

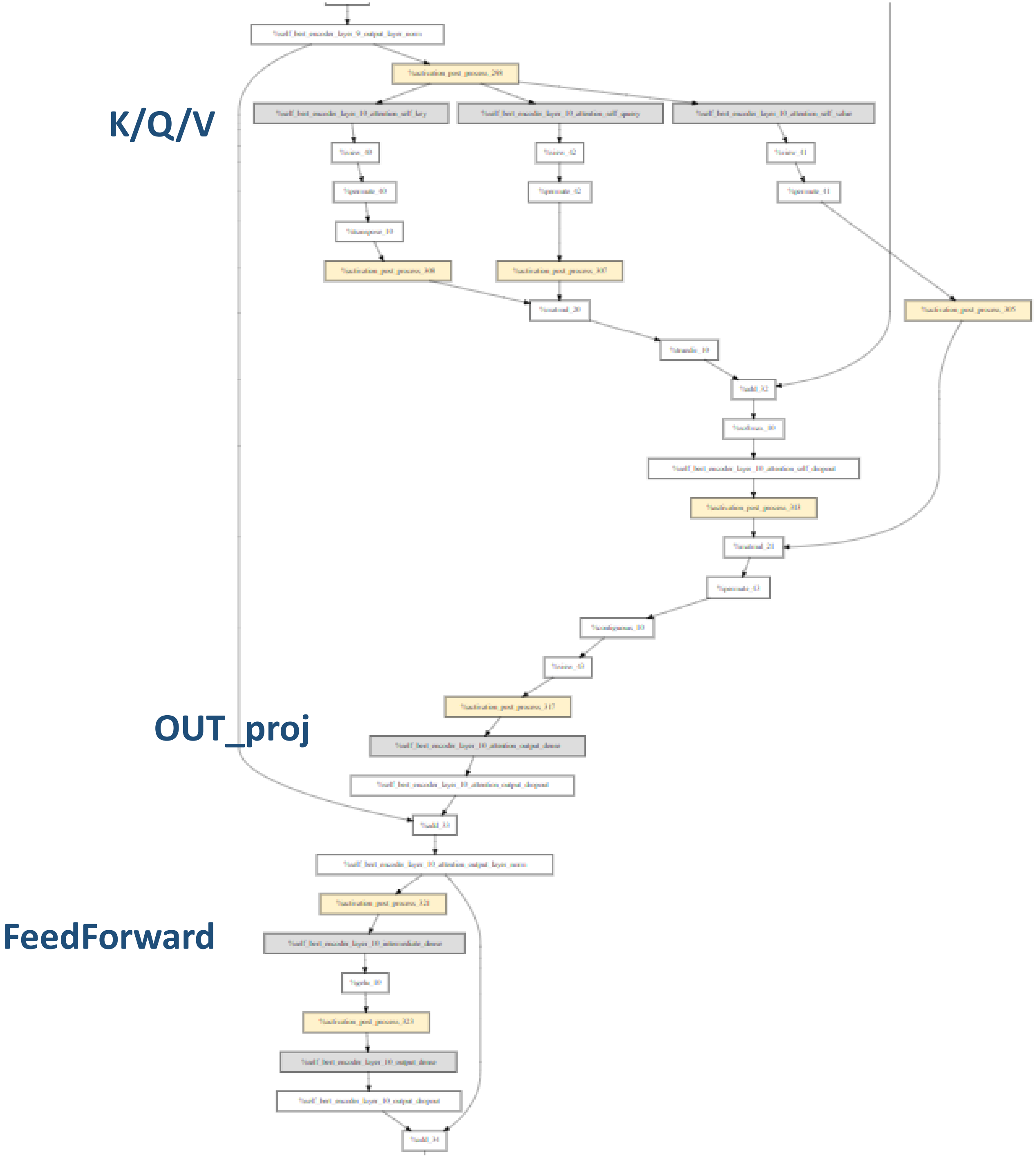
Viewpoint of Coders, Researchers, and Engineers

Make sense to torch.**compile**

[bert_model_prep_cleanup.svg]

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$





Metrics of performance

- Basic metrics - what can be measured directly
  - The duration of some time interval
    - Time spent in a function
    - Time to transmit a file
  - The count of an event
    - Number of L1 cache misses
    - Number of messages sent
  - The size of some parameter
    - Size of the memory used

Metrics of performance

- Derived metrics - calculation using measured metrics
  - CPI – cycles per instruction

  time (cycles) / # instructions
  - IPC – instructions per cycle

  # instructions / time (cycle)
  - Bandwidth utilization

  # bytes / time
  - FLOPS - floating pint operations executed per second

  #float_point_operations / time

Execution time

- Real/Wall clock/Elapsed time:

Actual elapsed time from a point in the past

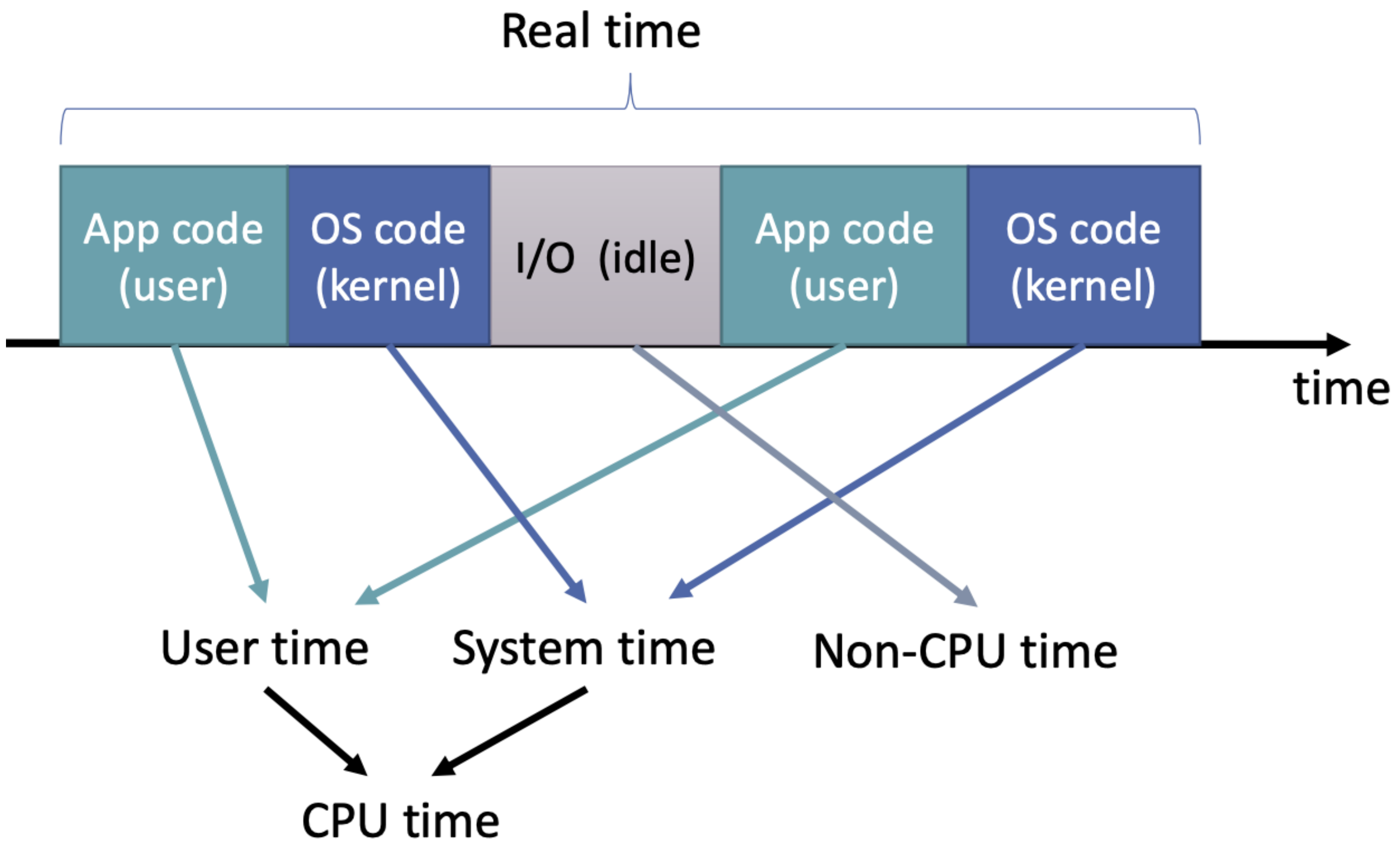
- CPU/Process time:

Time spent executing CPU instructions

  - User time: time spent in user space
  - System time: time spent in kernel (OS) space

- Non-CPU time:

Time spent waiting (CPU is idling) for I/O, virtualization, etc.



Morari & Domeniconi - NYU & IBM

Time Measurement - Linux

- time command - Real, User and System times

  $ time ./a.out

  real 0m2.450s

  user 0m0.430s

  sys 0m0.000s

- millisecond granularity, accuracy may vary between systems!
- real >= user + sys + non-CPU time

Time measurement - Python

- Real Time:

```
import time
start=time.monotonic()
....
end=time.monotonic()
print("time: “ + str(end-start))
```

- granularity fractions of seconds – printing in seconds
- time.monotonic_ns() (granularity in nanoseconds)
- https://docs.python.org/3/library/time.html

Nvidia GPU FLOPs measurement

- C:

```
#include <cuda_profiler_api.h>
cudaProfilerStart();
myKernel<<<…>>>(…);
cudaProfilerStop();
```

- Python:

```
import torch.cuda.profiler as profiler
profiler.start()
…
profiler.stop()
```

- Nsight profiler command

```
ncu --profile-from-start off –-metrics <comma separated list> --target-processes all <original job command>
```

- nvprof command (predecessor of Nsight)

```
nvprof  --profile-from-start off –metrics flop_count_sp --profile-all-processes <original job command>
```

- Why different from the estimation?

https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html

flop_count_sp (floating 16bit):
smsp__sass_thread_inst_executed_op_fp16_pred_on.sum *2

flop_count_sp (floating 32bit):
smsp__sass_thread_inst_executed_op_fadd_pred_on.sum +
smsp__sass_thread_inst_executed_op_fmul_pred_on.sum +
smsp__sass_thread_inst_executed_op_ffma_pred_on.sum * 2

Neural network memory complexity

- Memory for parameters
  - Fully connected layers
    - #weights = #outputs x #inputs
    - #biases = #outputs
- Memory for layer outputs
    - #outputs
- Backward propagation specific
  - Memory for Errors
  - Memory for parameter gradients
  - Memory for hyperparameter-related (e.g., momentum)
- Implementation overhead

What about convolution layers?
What about pooling layers?
What about batch size?

Model Summary in PyTorch

- pip install torchsummary
- MNIST

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchsummary import summary

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # PyTorch v0.4.0
model = Net().to(device)

summary(model, (1, 28, 28))
```

```
----------------------------------------
Layer (type) Output Shape          Param #
========================================
Conv2d-1 [-1, 10, 24, 24]          260
Conv2d-2 [-1, 20, 8, 8]            5,020
Dropout2d-3 [-1, 20, 8, 8]         0
Linear-4 [-1, 50]                  16,050
Linear-5 [-1, 10]                  510
========================================
Total params: 21,840
Trainable params: 21,840
Non-trainable params: 0
----------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.08
Estimated Total Size (MB): 0.15
----------------------------------------
```

Nvidia GPU memory utilization measurement

- nvidia-smi
- Pytorch CUDA API
  - cat gpumem.py

```
import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
if device.type == 'cuda':
    print(torch.cuda.get_device_name(0))
    print('Memory Usage:')
    print('Allocated:', round(torch.cuda.memory_allocated(0)/1024**3,1), 'GB')
    print('Reserved:   ', round(torch.cuda.memory_reserved(0)/1024**3,1), 'GB')
```

  - Python3 gpumem.py

```
Memory Usage:
Allocated: 0.0 GB
Reserved:    0.0 GB
```

Nvidia GPU memory utilization measurement

- GPUtil python package
  - pip3 install gputil psutil humanize
  - cat memreport.py

```
# Import packages
import torch
import os,sys,humanize,psutil,GPUtil
import torchvision.models as models

def mem_report():
  print("CPU RAM Free: " + humanize.naturalsize( psutil.virtual_memory().available ))
  GPUs = GPUtil.getGPUs()
  for i, gpu in enumerate(GPUs):
    print('GPU {:d} ... Mem Free: {:.0f}MB / {:.0f}MB | Utilization {:3.0f}%'.format(i, gpu.memoryFree, gpu.memoryTotal, 
gpu.memoryUtil*100))

wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
if torch.cuda.is_available():
  wide_resnet50_2.cuda()

mem_report()
```

- python3 memreport.py

```
CPU RAM Free: 244.9 GB
GPU 0 ... Mem Free: 44246MB / 45556MB | Utilization   3%
```

NYU Greene cluster setup

- Greene cluster info: https://sites.google.com/nyu.edu/nyu-hpc/hpc-systems/greene/getting-started?authuser=0
- Login into Greene cluster login node:

ssh greene.hpc.nyu.edu

- Launch an interactive job on a GPU node using slurm

srun -n4 -t2:00:00 --mem=4000 --gres=gpu:1 --pty /bin/bash

- Setup the env
  - Load the modules

  module load cuda/11.1.74 python/intel/3.8.6

  - Setup virtualenv

  python3 –m venv pytorch_env (only first time)

  source pytorch_env/bin/activate

  - Install torch packages (only first time)

  pip3 install torch torchvision torchsummary

  pip3 install –U numpy

NYU GCP cluster (A100, V100)

- Slurm account is csci_ga_3033_085-2024sp.
- Each user is assigned 200 GPU hours, they can access V100 and A100 GPUs. The GCP instances have CUDA profiling enabled.

To access GCP cluster

1. please first login to Greene cluster login node, then login to burst login node
2. ssh burst
3. From here to start interactive jobs, users are allowed to access these partition
   - srun --account=csci_ga_3033_085_2024sp --partition=n1s8-v100-1 --gres=gpu:1 --pty /bin/bash
   - srun --account=csci_ga_3033_085_2024sp --partition=n1s16-v100-2 --gres=gpu:2 --pty /bin/bash
   - srun --account=csci_ga_3033_085_2024sp --partition=c12m85-a100-1 --gres=gpu:1 --pty /bin/bash
   - srun --account=csci_ga_3033_085_2024sp --partition=c24m170-a100-2 --gres=gpu:2 --pty /bin/bash

Alternative: CIMS cuda[1-5].cims.nyu.edu setup

- Server info: https://cims.nyu.edu/webapps/content/systems/resources/computeservers
- Login into the cuda node:
  `ssh cuda3.cims.nyu.edu`
- Setup the env
  - Load the modules:
    `module load cuda-10.2 python-3.8`
  - Setup virtualenv:
    ```
    python3 –m venv pytorch_env # (only first time)
    source pytorch_env/bin/activate
    ```
  - Install torch packages (only first time)
    `pip3 install torch torchvision torchsummary`

Code and data

- Pytorch examples:
  git clone https://github.com/pytorch/examples
- ImageNet data
  - 1k class data set is sufficient, http://www.image-net.org/
  - Otherwise, try this:
    - Location on Greene cluster: /scratch/work/public/imagenet
    - Create a directory of your own using symbolic links for a small subset of training/test data

Performance modeling: Predicting behavior

Identify performance bottleneck

- Determine hardware limit
- Motivate algorithm improvement

Project future hw/sw performance

Simplest models: scaling properties

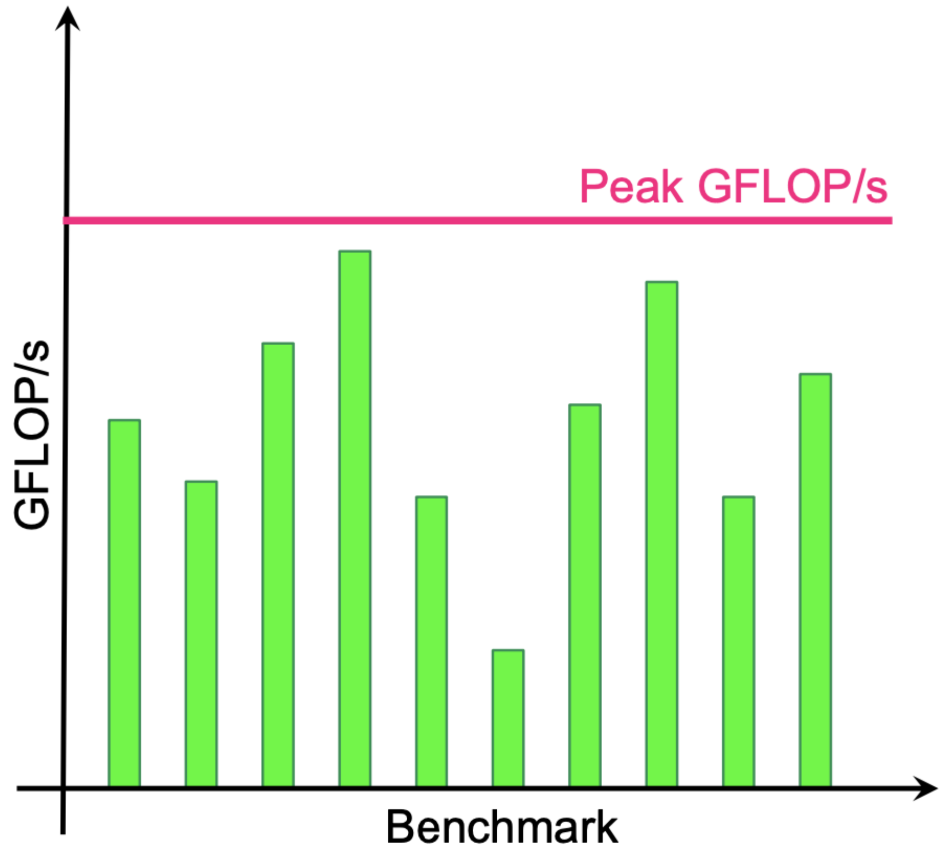
- What parts of the code are serial and parallel?
- How much time is spent in each?
- How efficient are they currently?

More complex concepts

- Roofline model (comparing throughput to theoretical maxima)
- Load balancing: what code is responsible for idle resources?
- Critical path analysis (e.g. Scalasca)

Are we getting good performance?

- When profiling GPU-accelerated applications...
- Want to know if the system is well-utilized by the
- GFLOP/s alone may not provide enough insight

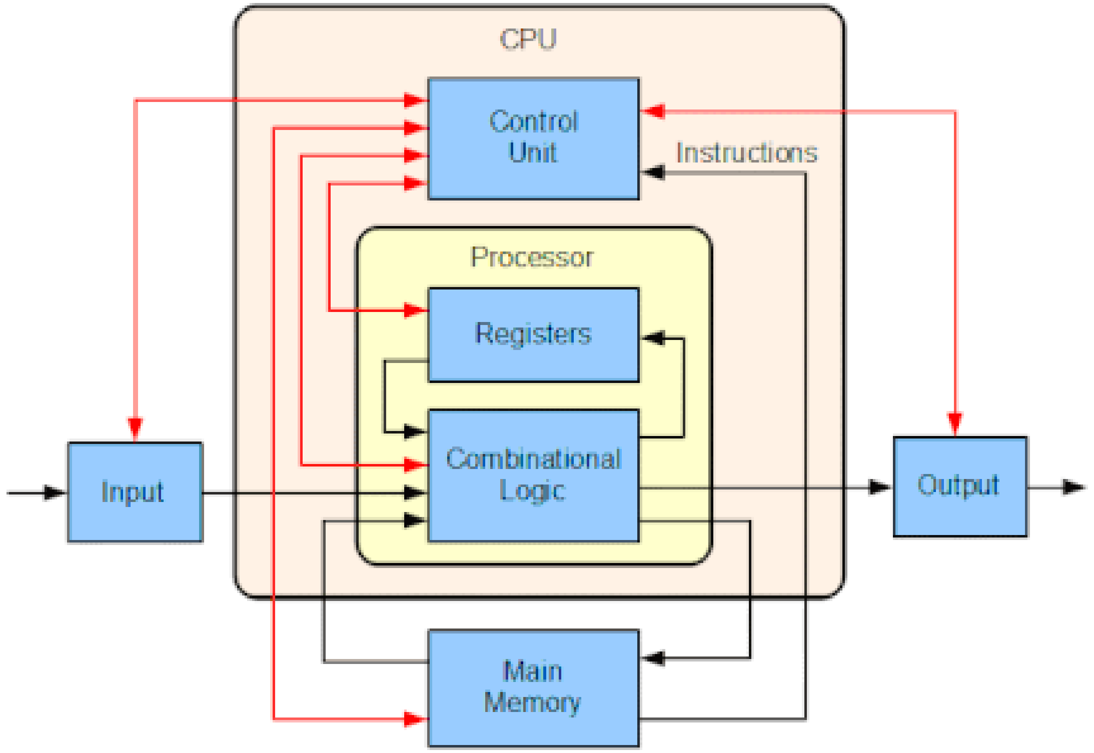


Baseline performance for comparison?

- Using the performance measurements form CPU
- Speedup may not be consistent and may be random
  - GPU isn’t always x times faster than a CPU
  - What does Speedup tell us about architecture hardware or application algorithm?
- Speedup provides not enough insights into architecture, application algorithm.
- Speedup provides no guidance system designer nor application users.

Using simulation to understand the performance?

- Modern architectures are incredibly complex
- What to simulate and what not to simulate may be challenging
- Even the simulator can perfectly reproduce the performance
  - may still be hard to interpret and reason the performance factors
  - Huge amount of simulated data, extended time to run simulation worse, (e.g., might incur 10^6x slowdowns)



Data movement versus computation

- Job execution time is the longer of
  - Data movement
  - Computation

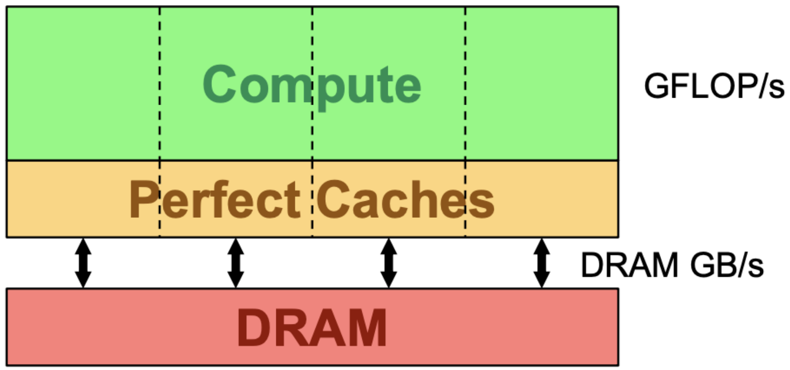
- $\text{Time} = \max\begin{cases} \dfrac{\#bytes}{GB/s} \\ \dfrac{\#\,FP\ operatoins}{GFLOP/s} \end{cases}$

Data size

System Peak Bandwidth

Number of calculations

System Peak Computation capability



How many FP ops can be completed in a given time?

- $\text{Time} = \max \begin{cases} \dfrac{\#bytes}{GB/s} \\ \dfrac{\#\ FP\ operatoins}{GFLOP/s} \end{cases} \overset{\text{Inverse}}{\Rightarrow} \dfrac{1}{Time} = min \begin{cases} \dfrac{GB/s}{\#bytes} \\ \dfrac{GFLOP/s}{\#\ FP\ operatoins} \end{cases}$

- *Multiply by # FP operations*

$$\frac{\#\ FP\ operatoins}{Time} = \min \begin{cases} \boxed{\dfrac{\#FP\ operations}{\#bytes}} * GB/s \\ GFLOP/s \end{cases}$$

How many calculations per each data byte – **Arithmetic Intensity (AI)**

System Peak Bandwidth

System Peak Computation capability

Roofline Model

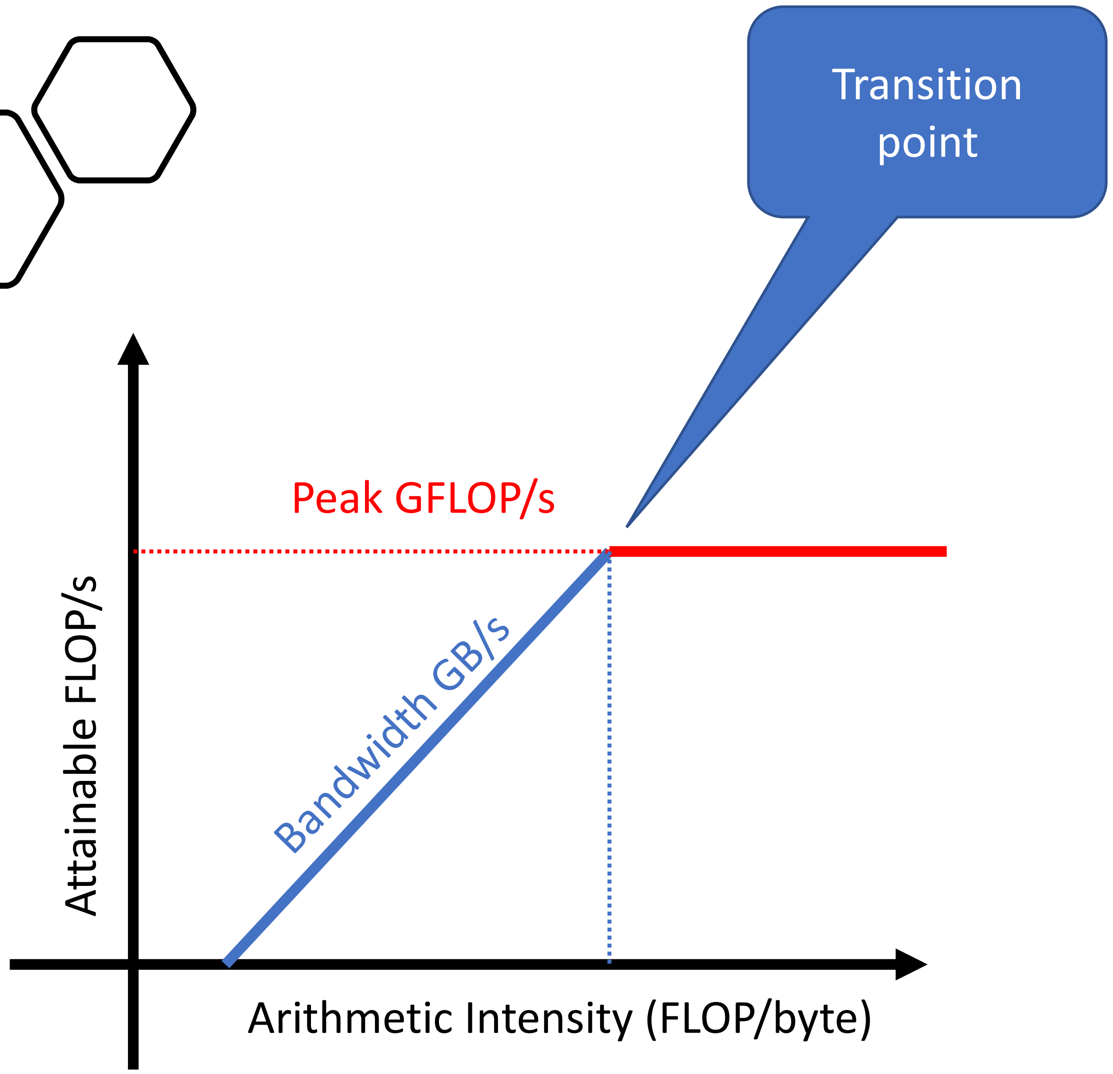
- $Attainable\ FLOP/s = \min \begin{cases} AI * peak\ GB/s \\ peak\ GFLOP/s \end{cases}$

- x axis and y axis are in log scale
- Transition point

$$AI * peak\ GB/s = peak\ GFLOP/s$$

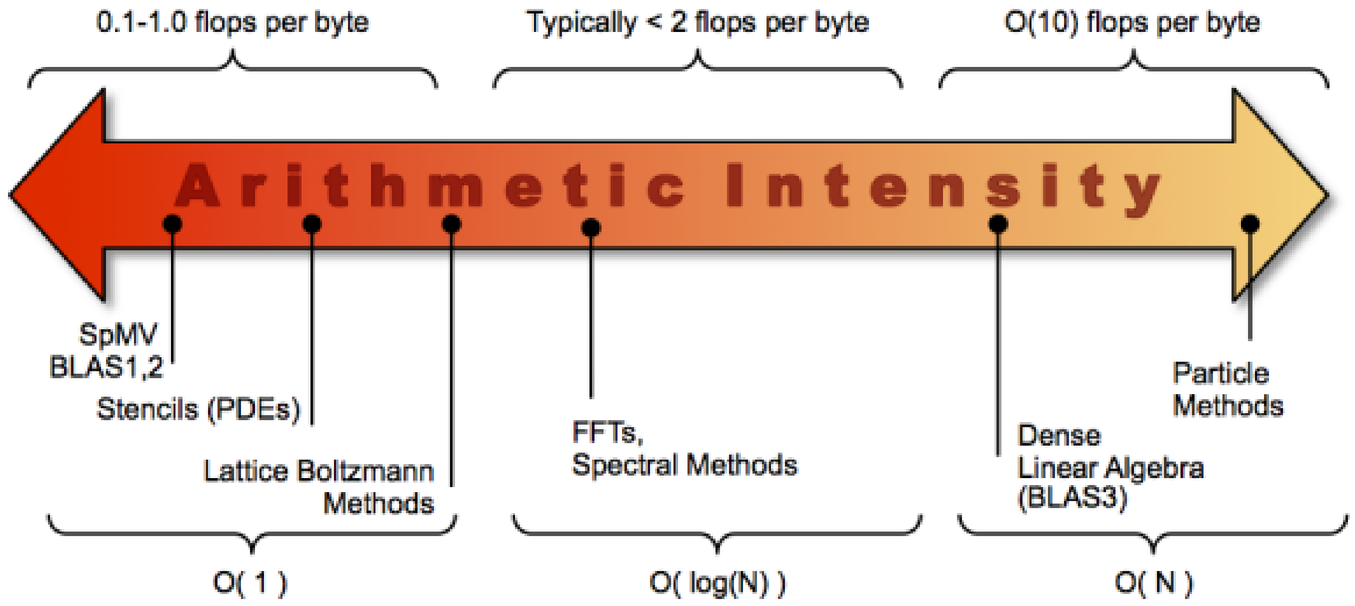
$$AI = \frac{peak\ GFLOP/s}{peak\ GB/s}$$

➔ Machine is "balanced"

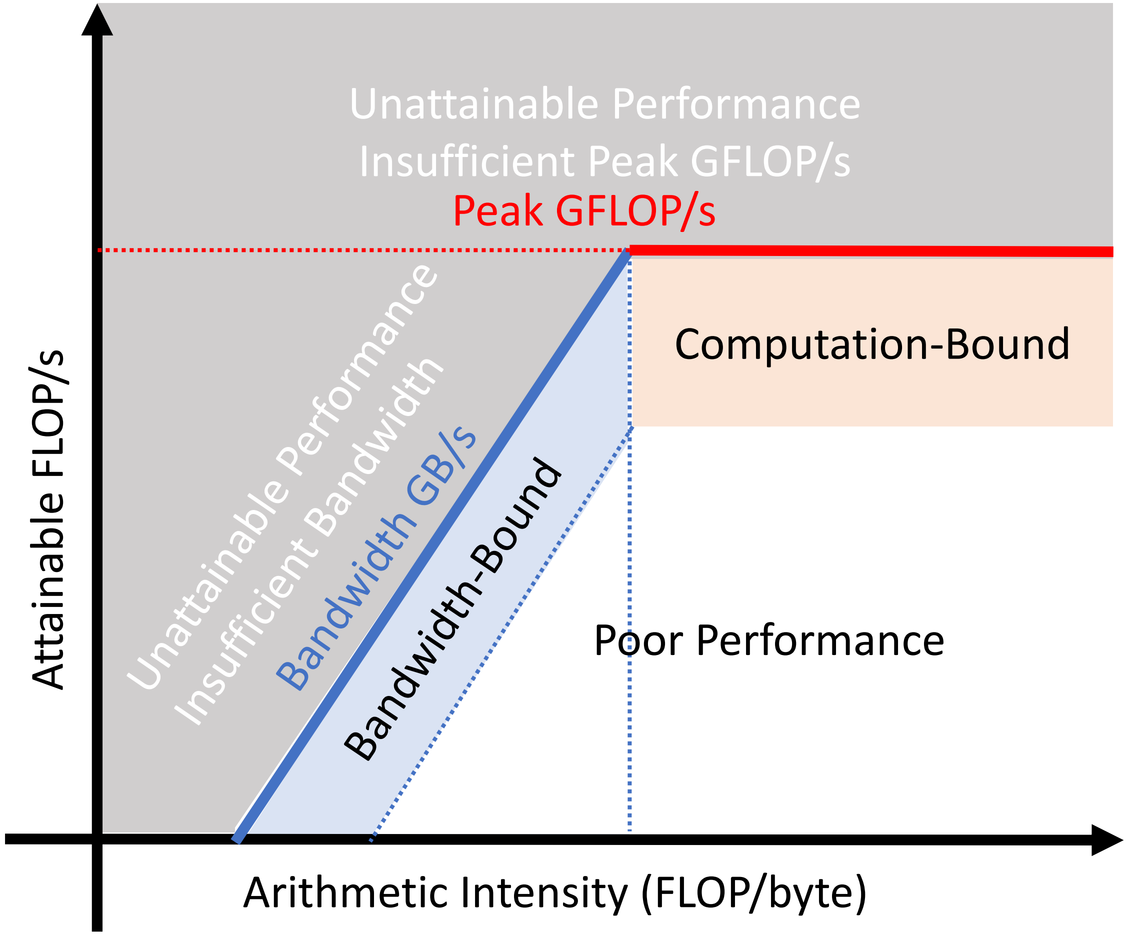


Arithmetic intensity – compute to comm. ratio

- The ratio of total floating-point operations (FLOPS) to total data movement (bytes)



Five Regions

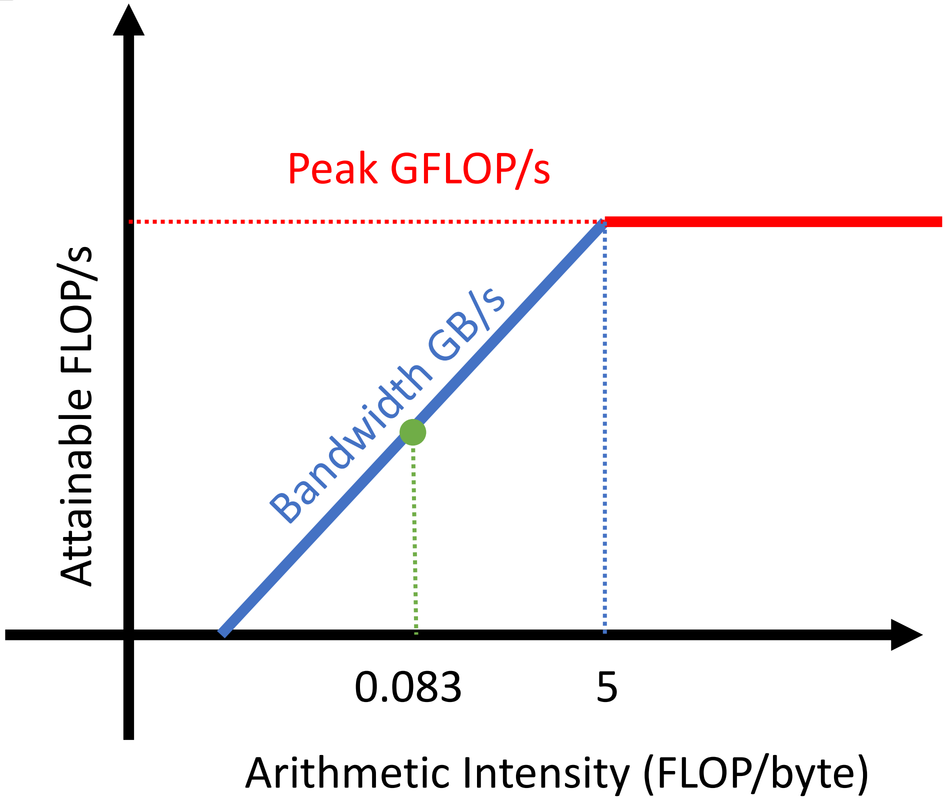


Roofline Example

- Typical machine balance is 5-10 FLOPs/byte
  - 40-80 FLOPs per double to exploit computation capability
  - Artifact of technology and money
  - It is unlikely to improve
- Consider a loop iteration

```
for(i=0;i<n;i++){
    z[i]=x[i]+alpha*y[i]
}
```

  - 2 FLOPs per iteration
  - Transfer 24 bytes per iteration (read x[i],y[i], write z[i])
  - AI = 0.083 FLOPs/byte → memory bound

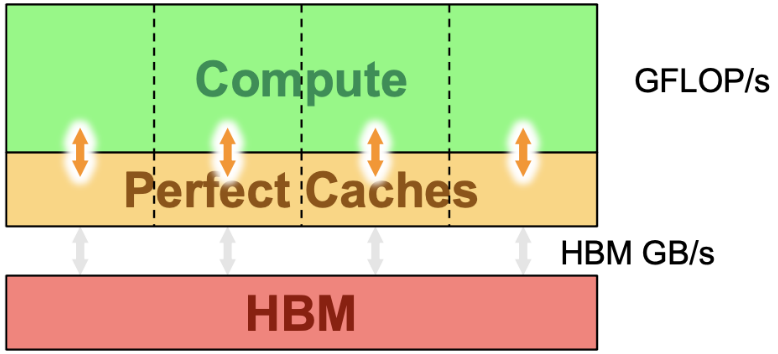


Roofline Example

- 7-point constant coefficient stencil

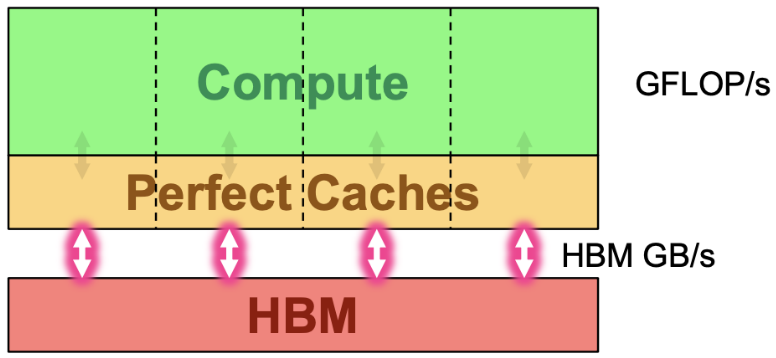
```
for (k=0;k<dim;k++) {
for (j=0;j<dim;j++) {
for (i=0;i<dim;i++) {
new[k][j][i]= -5.0 * old[k  ][j  ][i  ]
                   + old[k  ][j  ][i-1] + old[k  ][j  ][i+1]
                   + old[k  ][j-1][i  ] + old[k  ][j+1][i  ]
                   + old[k-1][j  ][i  ] + old[k+1][j  ][i  ];
}}}
```

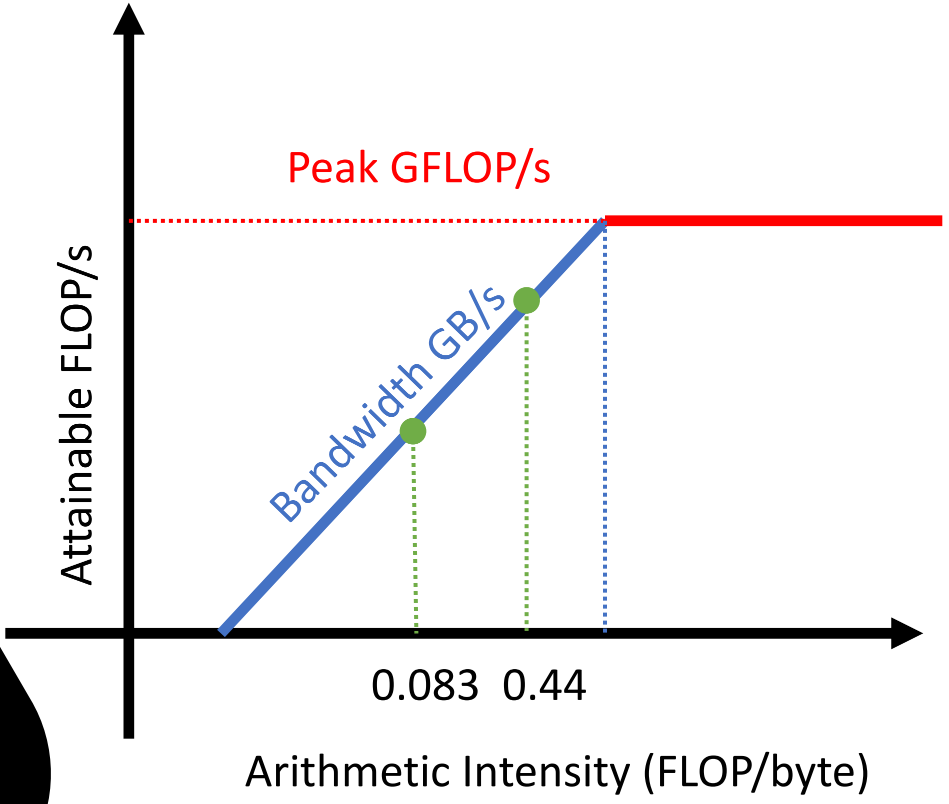
  - 7 FLOPS
  - 8 memory references (7 reads, 1 write) per point
  - AI = 7 / (8*8) = 0.11 FLOPs/byte (measured at L1)



Roofline Example

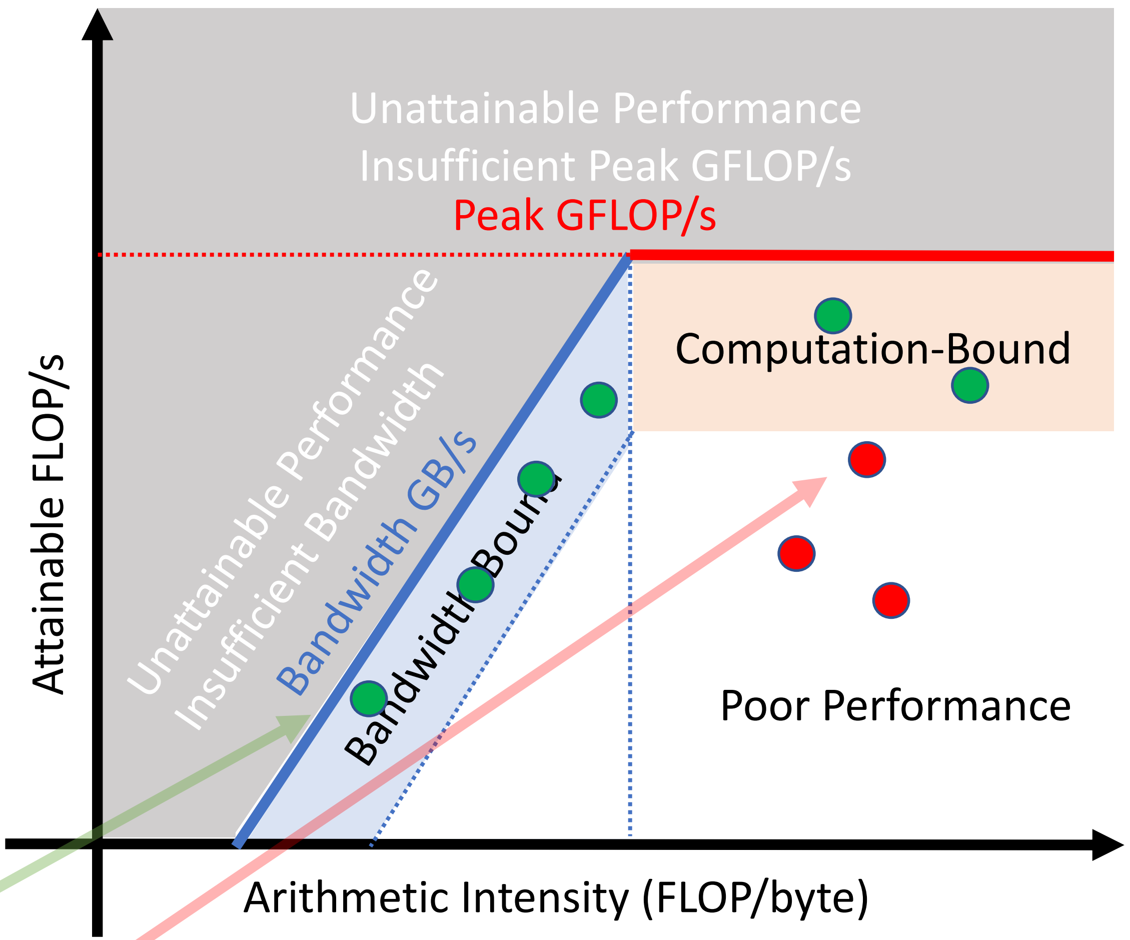
- Cache helps...
- Ideally, cache will filter out the all the memory accesses but 1 read and 1 write
- AI = 7 / (8+8) = 0.44 Flops/byte
- Still memory bound, but 5x the FLOP rate compared to the previous example





Are we getting good performance?

- Sort benchmarks by AI and plot them into the chart
- Compare the performance against machine capabilities
- Benchmarks close to the rooflines are utilizing the computation resources well
  - Benchmarks can have low performance (GFLOP/s) but make good use of the system
  - Benchmarks can have a high performance (GFLOS/s) but still make poor use of the system



Roofline Recap

Machine Model

- Lines defined by Peak GB/s (bandwidth) and GFlop/s (computation)
- System dependent – unique to each architecture
- Common to all applications running on that system

Application Characteristics

- Dots measured by application Gflop’s and GB’s
- Unique to each application
- Unique to each architecture

Project 1 – ImageNet Analysis

- Pytorch code: https://github.com/pytorch/examples/tree/master/imagenet
- ImageNet 1k data: http://www.image-net.org/
  - You don't need the whole data set - just extract some are good enough.
- May use Nsight Roofline
  - https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#roofline

Mission - comparison

- Design the experiment to show the difference
  - 2-3 Environment – Cloud vs Baremetal, different CPU/GPU flavors, etc.
  - 2-3 Neural Network models
- Short representative runs – no need to finish training as the accuracy does not matter.
- Report
  - Experiment design (10%) – what are you trying to show and what is the hypothesis
  - Complexity estimation (10%) and measurement (20%)
  - Roofline modeling (40%)
  - Discussion (20%)
- Due Mar. 27, Wed, 11:59pm

	Different environment	
Different NN models		