CSC290/420 Machine Learning Systems for Efficient AI
Running ML Programs

Sreepathi Pai

October 15, 2025

URCS

Outline

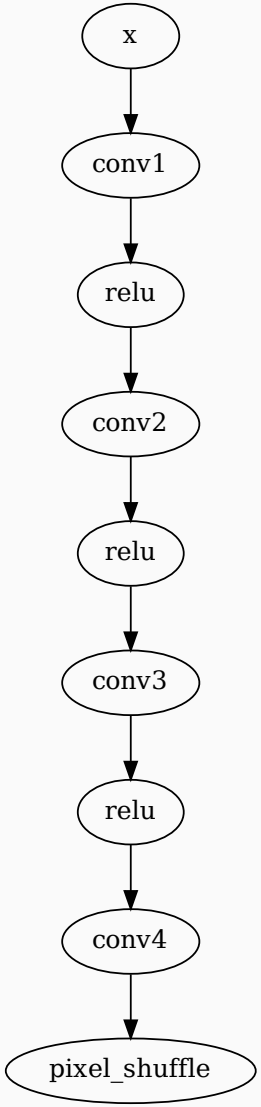
Outline

ML Machines

- Multicore CPUs
- GPUs
- Accelerators
  - TPU
  - Groq
  - Amazon Inferentia

Abstract Problem

- Map operators to compute units over time
- Ensure data flows respect data dependences
- Obtain high throughput (work / time)
    - or minimum latency per work (less important)

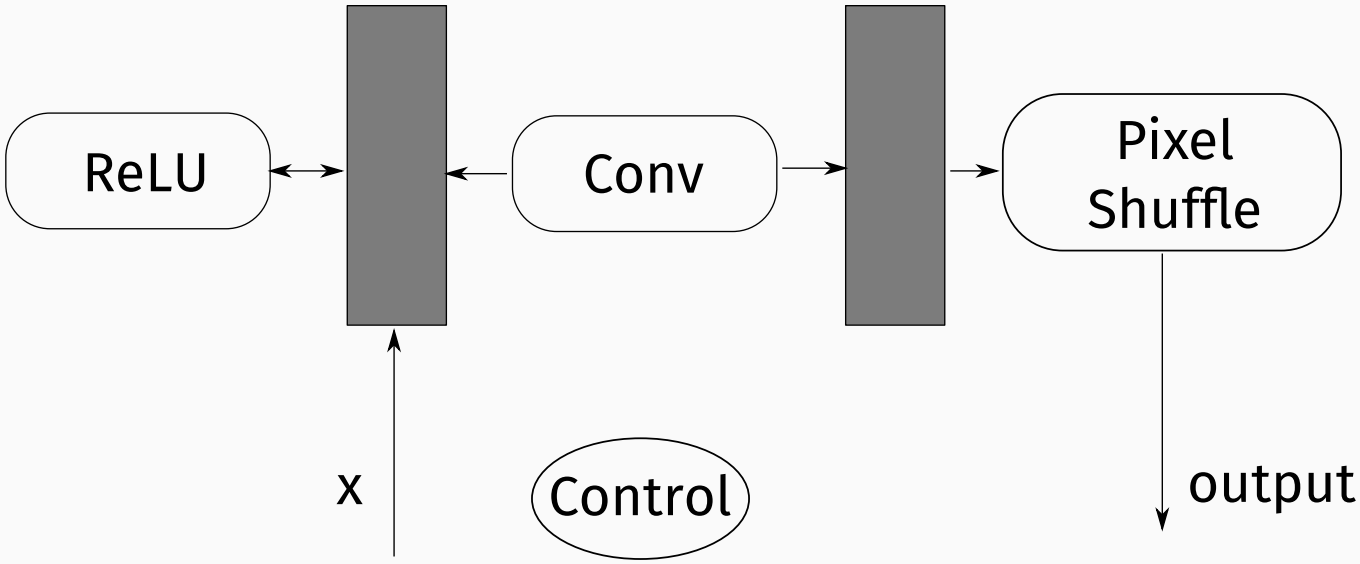


Concrete Problem

- Schedule operators and data flows on to machine ("macroscopic view")
  - Needs timing for each operator and data flow
  - Timing is "easy" when inputs and outputs have fixed, unchanging sizes (i.e., not LLMs)
- Write a high performance operator implementation ("microscopic view")
  - when not provided by hardware
  - preferably tunable
- Tune schedule and operators until you reach peak throughput

Outline

Each operator type is its own functional unit



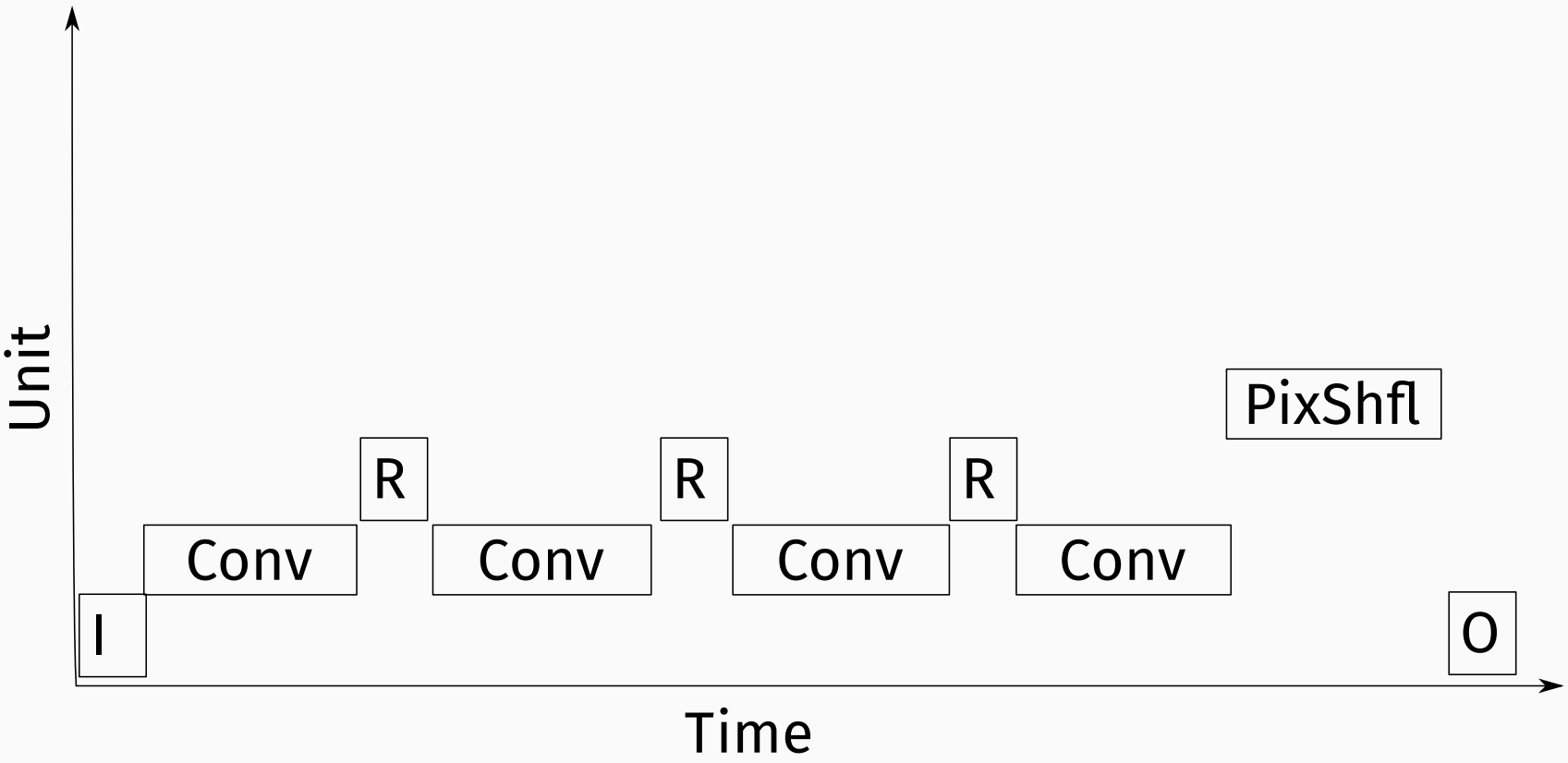
- To simplify, we can make each type of operator its own functional unit
  - With dedicated input and output buffers
- Other alternative:
  - Build exact operator needed for ML program (e.g., 5x5 convolution over 128x128 grayscale images)
  - Mapping is simplified, but wastes space

Just one convolution?

```
self.conv1 = nn.Conv2d(1, 64, (5, 5), (1, 1), (2, 2))
self.conv2 = nn.Conv2d(64, 64, (3, 3), (1, 1), (1, 1))
self.conv3 = nn.Conv2d(64, 32, (3, 3), (1, 1), (1, 1))
self.conv4 = nn.Conv2d(32, upscale_factor ** 2, (3, 3), (1, 1), (1,
```

- Different number of in channels vs out channels (first two arguments)
- Different convolution kernel sizes (5x5, 3x3)
- Different strides, padding
- Could have implemented this as different convolution units
- ReLU also operates on different size inputs

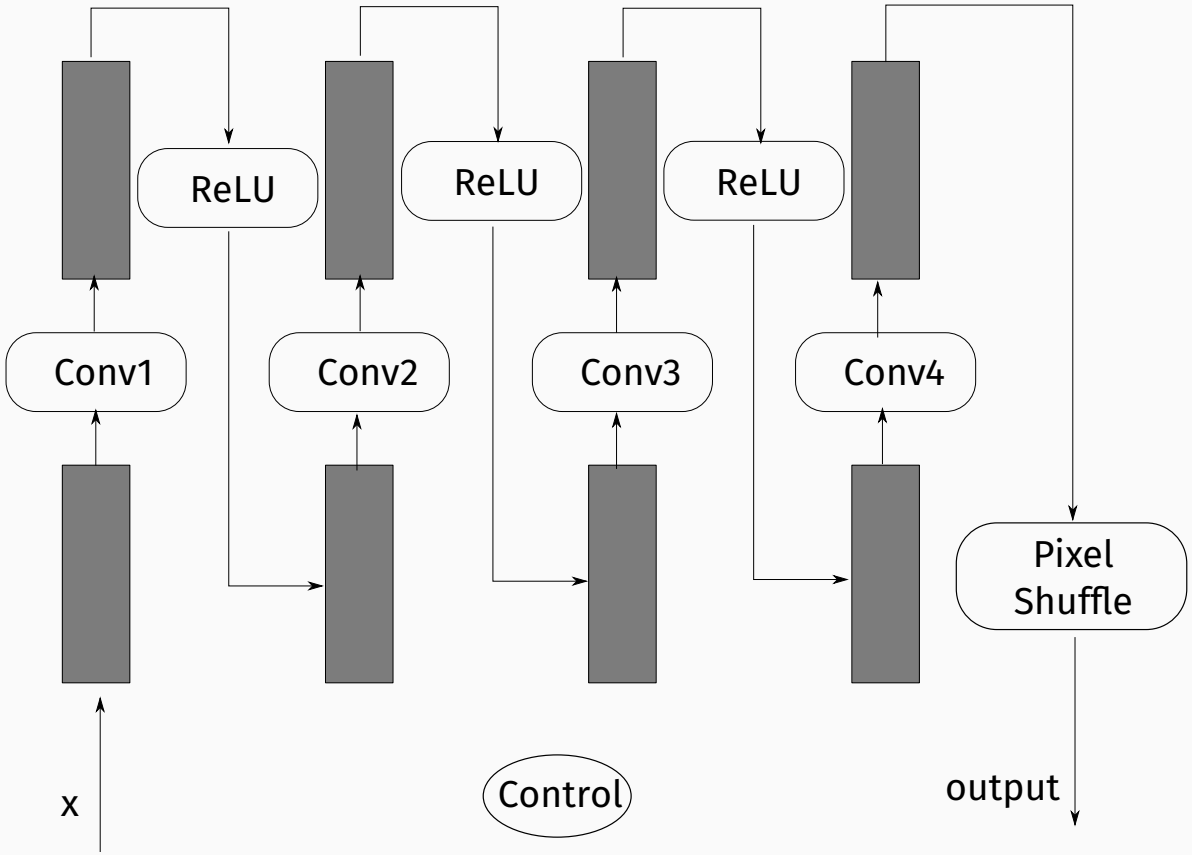
Serial Timeline



Pipeline Parallelism

- We can implement pipeline parallelism
  - Different inputs are active at each unit
- Requires re-design of accelerator
  - Need more buffers
  - Need separate convolutional units
  - Note: simple design has pipeline stages of different durations

With Pipeline Parallelism

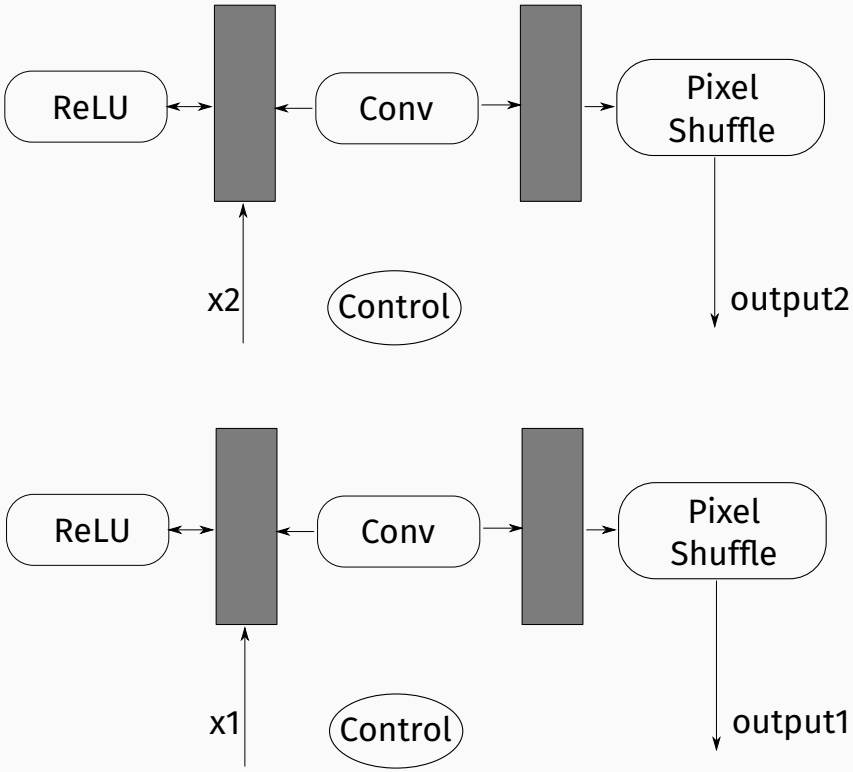


- Increased Parallelism demands increased resources
  - Memory
  - Compute
- This is true even on general purpose machines!
  - generally increased memory usage

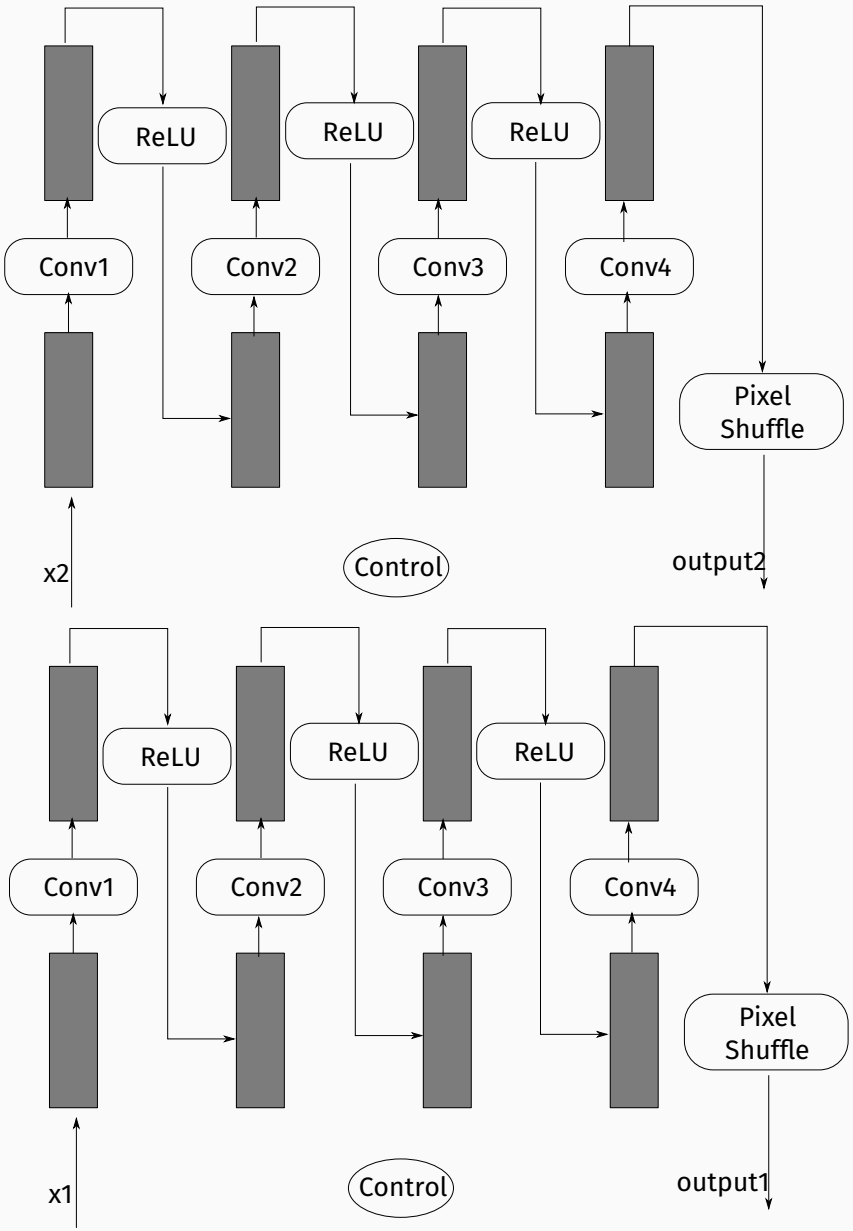
Data Parallelism

- Process multiple inputs at the same time
    - on independent hardware

Simple DP

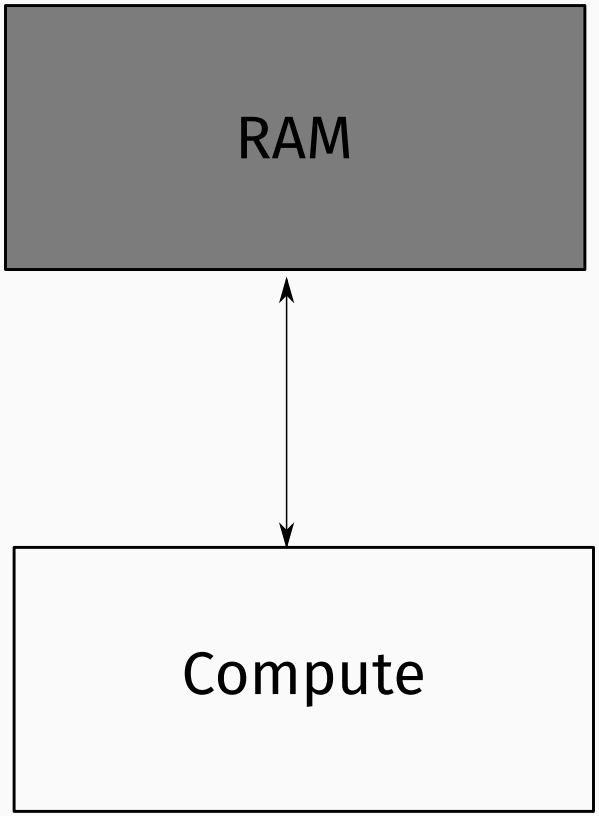

ReLU
Conv
Pixel Shuffle
x2
Control
output2
ReLU
Conv
Pixel Shuffle
x1
Control
output1

Pipelined DP



Outline

A General Purpose Architecture



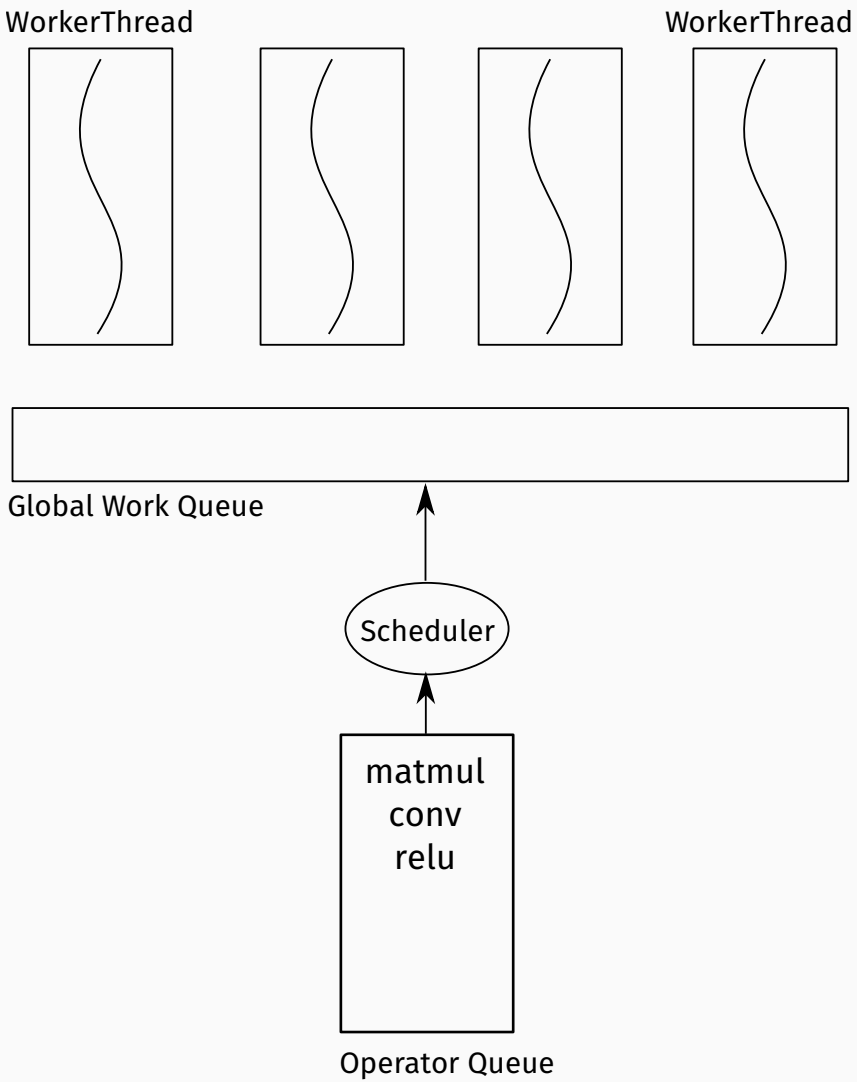
- Compute may contain multiple cores
- Memory might be NUMA

Additional Complications

- Primarily, *one size doesn't fit all*
  - Fixed hardware, shared resources
- Utilization varies across operators
- Data path is shared among operators
  - All operators read and write from the same memory
- Memory is limited
- All operators are implemented in software

Dynamic Scheduling

- Classic design: Scheduler + Worker threads
- Scheduler divides operator work into worker-size chunks and places it in a work queue/pool
- Worker threads grab chunks from queue
  - Most general method, can be specialized
- Independent operators can execute in parallel
  - if there are resources
- Dependent operators wait



Static Scheduling

- For most ML programs, the sizes of inputs and outputs are fixed
- Implies that timing of each operator can also be predicted in advance
    - Provided machine is fixed
- Timing information + Data flow can be used to generate a schedule *offline*
- Haven't seen this style of execution on CPUs/GPUs (yet?)
    - only in specialized contexts like Groq's chips where the hardware requires it

Macroscopic Optimizations

- Fusion
  - Combine two operators together
- Partitioning
  - Separate the graph into parts, with each part executing on a particular device
- Parallelism optimizations
  - Data Parallelism (run multiple copies of the graph on different user inputs)
  - Pipeline Parallelism (run multiple operators on different user inputs)
  - Tensor Parallelism (run graphs on multiple devices with tensors split across devices)
  - other forms as well ...

Outline

A Software Operator

- An operator kernel must be written for each general purpose device
  - CPUs, GPUs, etc.
- Usually multiple variants of a kernel exist
  - for different sizes, e.g.
- But all must achieve high operation throughput
  - Implies keeping the pipeline full

Balancing Operation Throughput

- A machine can perform 2 32-bit FMA/cycle
- Each FMA takes 4 cycles
- By Little's law, $n = Rt$ (with $R = 2$ and $t = 4$), there are 8 FMAs in flight
- Each FMA requires 3 operands
  - each operand in a register
- So 24 registers are needed for 8 FMAs
- Consequently, there are also 24 loads in flight
- We are consuming 12 bytes/cycle (per 32-bit FMA)
  - and if we're reading from memory, this is the bandwidth required.

Another example

- A machine can transfer 64-bits / cycle from RAM
  - assume each load is also 64-bits, so 1 load/cycle
- The latency of a load from RAM is 100 cycles
- How many loads are required to saturate bandwidth?
  - $n = Rt$, with $R = 1$ and $t = 100$, so 100 loads in flight
- Some alternatives:
  - transfer data in blocks (cache line size = 64 bytes, about 8 loads)
  - rely on a prefetcher

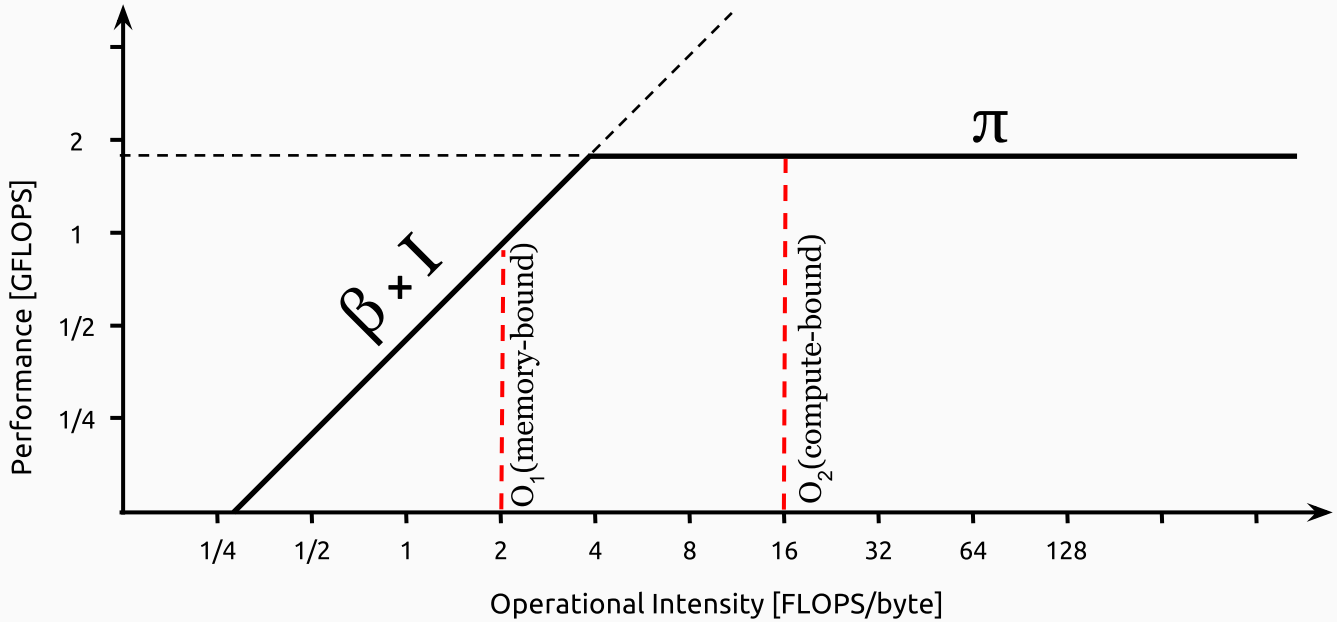
Latency Hiding through Parallelism

- Little's Law tells us n parallel operations are needed to keep the pipeline (or bandwidth) full
  - i.e. "hide" the latency of the operation
- This n dictates how many other operations need to be in flight
  - memory for compute
  - compute for memory
- If not enough operations are available (lack of parallelism) or there are not enough resources (i.e., structural hazards, e.g., registers, queue sizes, bandwidth)
  - then some of the latency will be exposed and throughput will drop

The Roofline Model

- The Roofline model applies to regular numerical codes
  - be skeptical of applications beyond regular numerical code
- Uses arithmetic intensity I, denoted in FLOPs/byte
  - Floating point operations per byte β
  - (do not confuse with FLOP/S which is Floating point operations per second)
- $P = \min(T_{\text{peak}}, \beta \times I)$
  - T_{peak} is peak FLOPs/second of the machine
  - β is bandwidth in bytes/second
- Informally: if you performed 1 FLOP per byte of RAM read, you cannot perform more FLOP/S than β
  - Generally, T_{peak} is multiple teraflops/s, β is usually low terabytes/s
  - implies "reuse" of data required to reach high teraflop/s

Roofline graph



Outline

Optional Further Readings

- Roofline: an insightful visual performance model for multicore architectures
- All about Rooflines: Part 1 of How to Scale your Model
- Applying the Roofline Model