CSC290/420 Machine Learning Systems for Efficient AI
ML Programs as Computational Graphs

Sreepathi Pai

October 1, 2025

URCS

Outline

Outline

AI/ML

- Artificial Intelligence
  - roots go back to the 50s
- Machine Learning
  - likewise
- Neural Networks (NN)
  - "Universal" functions
  - that can "learn" (or more accurately, can be trained)
- Deep Learning (DL)
  - Neural Networks with lots of layers

ML applications of Interest

- Deep Learning Task-specific Models
  - Computer vision, speech-to-text, etc.
- Large Language Models (LLMs)
  - various linguistic tasks
  - e.g., ChatGPT, LLAMA, etc.
- Diffusion Models
  - mostly image generation (?)
  - e.g., Stable Diffusion
- These have:
  - high amounts of compute
  - high amounts of memory usage
  - large amounts of data involved
- LLMs and Diffusion models called "foundation models"

Three tasks / modalities

- Training
  - Models are trained on examples
  - "Learning" phase
  - Time-consuming, very expensive for LLMs
  - Almost entirely restricted to large, well-resourced entities
  - Done once
- Fine-tuning
  - "Low-cost" retraining for specific tasks
  - Done once per end-user application
- Inference
  - "Lowest" cost
  - Once per user request

Outline

ML Programs as Instructions

- ML programs are just programs
  - ultimately instructions that execute on a CPU or GPU
- But they possess higher-level structure that is useful to consider
  - analogous to the difference betweeen physics, chemistry, and biology
- Two important perspectives
  - ML programs are computational graphs
  - ML programs are loops

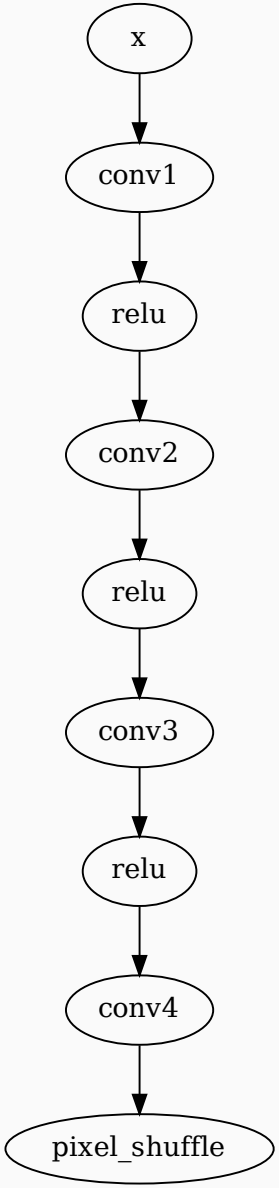
ML Programs

- A ML program will be viewed as a series of operations
  - or "primitives"
  - similar to computer instructions
- Unlike instructions, most operations consume and produce large amounts of data
  - vectors, matrices, etc. (also called tensors)
- ML programs usually have no conditional control flow
  - All operations are executed
- ML programs usually do not have loops at the operation level
  - The programs are direct acyclic graphs (DAGs)

An example

```
import torch
import torch.nn as nn
import torch.nn.init as init


class Net(nn.Module):
    def __init__(self, upscale_factor):
        super(Net, self).__init__()

        self.relu = nn.ReLU()
        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,
        self.pixel_shuffle = nn.PixelShuffle(upscale_factor)

        self._initialize_weights()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.pixel_shuffle(self.conv4(x))
        return x

    def _initialize_weights(self):
```

Graph



Classifying ML Operators

A empirical classification of operators, based on original research:

- Tensor operators, element-level/elementwise
  - operates on individual elements of tensors
  - e.g. vector addition
- Tensor operators, tensor-level
  - requires entire tensor to perform operation
  - e.g. matrix multiplication
- Tensor operators, reductions
  - produces smaller tensors
- Data Transformation operations
  - Reshape, etc.

ML Computers

- Multicore CPUs
  - most common for inference of older models
- GPUs (most common)
  - graphics processing units
  - most common for training and generative AI inference
  - NVIDIA, AMD
- Multicore CPUs + AI accelerators
  - Intel Gaudi
- Specialized Processors
  - Google's Tensor Processing Units
  - Groq "LPU"
  - Cerebras Wafer-scale
  - ...

Running a ML program

- Eager Execution
  - Each operation executes as soon as it is encountered
  - e.g. early versions of PyTorch
- Graph Execution
  - A graph of operations is first built
  - Then, the graph is compiled and optimized
  - The compiled graph is executed
  - e.g. PyTorch 2.0 (`torch.compile`)

Eager Execution

- Each operation ("convolution") maps to one or more primitives
  - here, usually just convolution
- A primitive implemention exists for each possible device (CPU, GPU, etc.)
  - e.g., as a collection of library functions, GPU kernels, etc.
- An operation decomposes into calls to these pre-existing functions
- It is possible that primitives may be generated on the fly

Graph Execution

- Each operation maps to one of more primitives
- Non-trivial transformations may be applied to graphs
  - Fusion: operations are combined (e.g., Convolution + Relu)
  - More complex: recognize attention kernels and replace with Attention operator
- Operations scheduling may change
  - Reduce memory traffic
  - Use multiple GPUs, etc.
- Primitives may generated and specialized per operation
  - A different matrix multiplication for each different matrix size

ML Primitives

- Examples
  - Convolution
  - Matrix Multiplication
- Started out as handwritten functions for each device
  - Provided by device vendors
  - NVIDIA's cuBLAS, Intel's MKL, etc.
- Now are usually generated by specialized compilers
  - Triton
  - TVM

Compiling ML Computation Primitives

- Example: Matrix Multiplication
- Many different approaches:
  - Tensor compilers: e.g., TVM, Tensor Comprehensions, etc.
  - Tile compilers: e.g., Triton
- Source language for these tools is device independent
  - Usually, some domain-specific language
- Progressively lowered to device-specific code
  - Using, for example, the MLIR framework

ML Communication Primitives

- Once ML progams run on multiple GPUs or multiple machines, some mechanism for communication is required
- Communication Primitives (not exhaustive)
  - Gather
  - Scatter
  - AllGather
  - Broadcast
  - Reduce
  - AllReduce
- Later in the course

Outline

Metrics

- Time/Latency
  - wall clock time
- Throughput
  - work per unit time
- Scalability
  - work per unit resource
- Utilization
  - busy time
- Energy
  - quantity used for doing work
- Power
  - energy per unit time
- “Loss” / Error

Multiprocessing ('Data [Level] Parallelism')

- The simplest and easiest form of parallelism
- For each input (e.g. images) which are naturally independent
  - the inputs may be from different users, for example
- Run many copies of the model, one for each input
- Always possible, as long as there are independent inputs

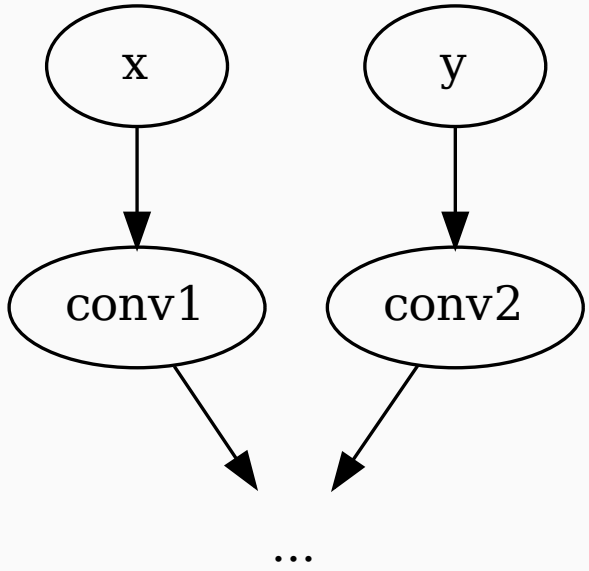
Intra-operator parallelism

- Consider matrix multiply
- Each element of the output matrix can be computed independently of the others
  - $C_{ij} = \sum_k A_{ik} * B_{kj}$
- This leads to a parallel implementation of matrix multiply

Intra-operator parallelism is exploited by building a parallel implementation of an operator.

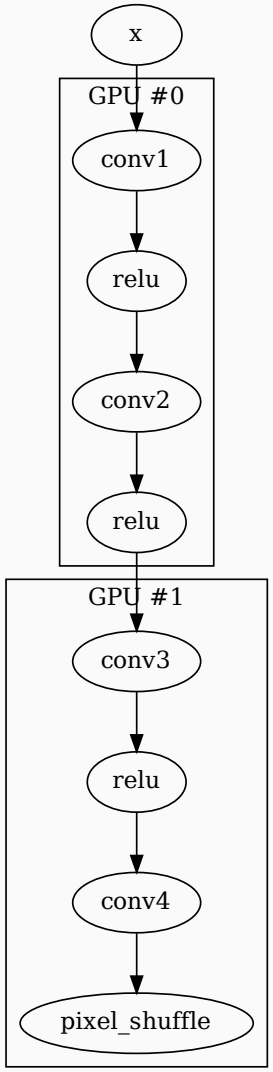
Inter-operator parallelism

- Independent operators can execute at the same time, in parallel



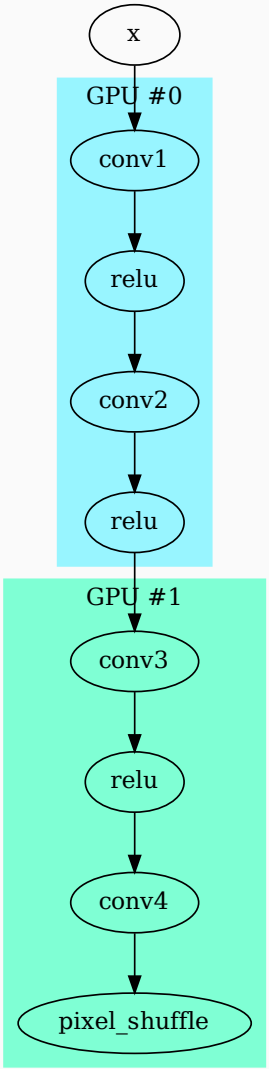
Partitioning ('Model [Level] Parallelism')

- When there are two many operators to fit on one device, the model is partitioned
- Data is communicated across devices
- Appears like a pipeline



Pipeline Parallelism

- Like partitioning, except multiple inputs are processed
- Different partitions may be processing different inputs
- Different colors indicate different inputs being processed



Tensor Parallelism

- Edges represent tensors flowing in and out of operators
- Can also split up a tensor across multiple GPUs/CPUs

Optimizing a single-node ML program

- Let's assume the only operator in a ML program is a matrix multiply
- How shall we execute it?
  - Assume matrix size of *MxM* for all matrices
- Considerations:
  - Is *MxM* large enough to utilize all compute units?
  - Is *MxM* small enough to fit in one GPU's memory?
  - Are there multiple inputs, or a single input?

Options

- Single-threaded matrix multiply (MM), small M
- Multi-threaded MM, large enough M
- Multi-GPU MM, with matrix split up into parts that can fit in one GPU
- Single-threaded MM, run as multiple processes, for many small inputs
- ...

Outline

References

- Optimizing Production PyTorch Models Performance with Graph Transformations
- Graph Optimizations in ONNX Runtime
- Google's Machine Learning Glossary: `https://developers.google.com/machine-learning/glossary`