CSC290/571 Topics in Systems: Machines Learning Systems
Making systems fast and scalable: A compendium

Sreepathi Pai

Aug 27, 2024

URCS

Outline

Metrics and Models

Reducing Work and Cost

Increasing Parallelism

Scalability

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

Models

- A *model* for a metric depicts a relationship between "input" variables and the (output) metric
  - usually mathematical
- Allows reasoning beyond just measuring the metric
- Helps explain the measured value of a metric
  - and, hopefully, devise strategies to control it
- Pay attention to discrepancies between a model and a metric
  - Model is wrong: may prompt upgrading the model
  - Data is wrong
  - Data is faked

A Performance Model

$$T = \frac{W \times t}{P}$$

- T - total execution time
- W - total work
- t - average time per work (cost)
- P - average parallelism

Outline

What is work?

- Algorithmic Work
  - Asymptotic complexity
  - Constant factors
- Instructions
- Operators

General strategies

- Avoid work
- Reuse work
- Amortize work
- Size work for hardware capabilities
  - i.e., utilize hardware well
- Transform work
  - Compute to Memory (memoization)
  - Memory to Compute (recomputation)

Classifying Work

- Compute
  - Work performed by the processing unit
  - Arithmetic operations, logic operations, etc.
- Memory
  - Work that interacts with memory
  - Load/store instructions
- Communication (I/O)
  - Work that interacts with peripherals and/or other machines

Examples of reducing compute work

- Use a simpler algorithm
- Eliminate redundant operations
- Combine operations
  - e.g., most processing units can combine multiply and add into a single operation

Examples of reducing memory work

- Reduce data sizes
  - use smaller data types, for example
- Reduce range/precision of data
  - quantization
- Compress data
- Avoid large temporaries

Examples of reducing communication work

- Avoid communication
  - Send data every k steps instead of every step
- Avoid synchronization
  - Use asynchronous algorithms

Cost

- Compute costs
  - Avoid computation
  - Replace computations with cheaper equivalents (e.g., shifts instead of multiplications when possible)
- Memory costs
  - Numbers every programmer should know
  - Improve cache behaviour

Outline

Data Dependence

- Two operations are data dependent if one reads as its input the output of the other
- They are independent otherwise

```
A = C * D
E = A * G
F = B * C
```

- Independent operations can execute in parallel, provided resources are available

Control Dependence

- A control dependence requires an operation to wait until a branch is resolved

```
X = C * D
if(A) {
  Y = E * F
}
```

- In this case, the operation `Y = E * F` must wait until the value of `A` is known

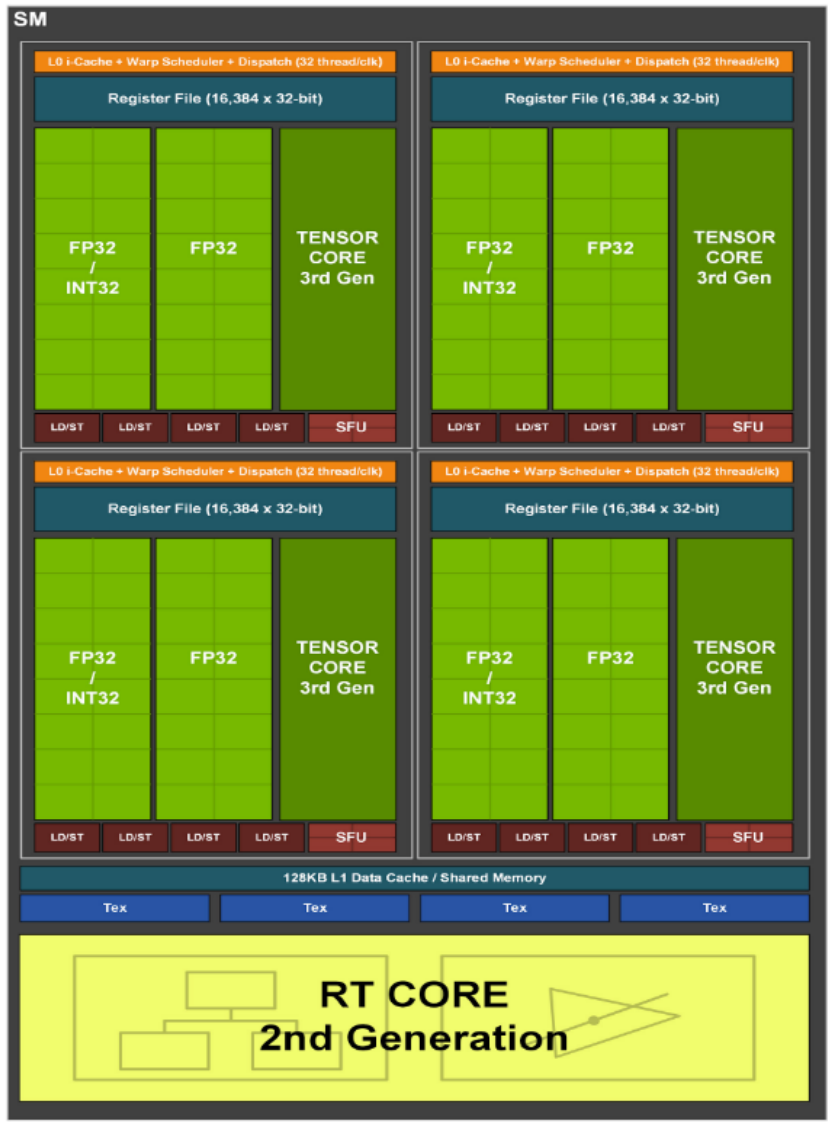
Structural Dependence/Hazard

- Two operations are independent, but must wait for each other because they use the same resource
- Assume in the following code, the machine contains only one multiplier

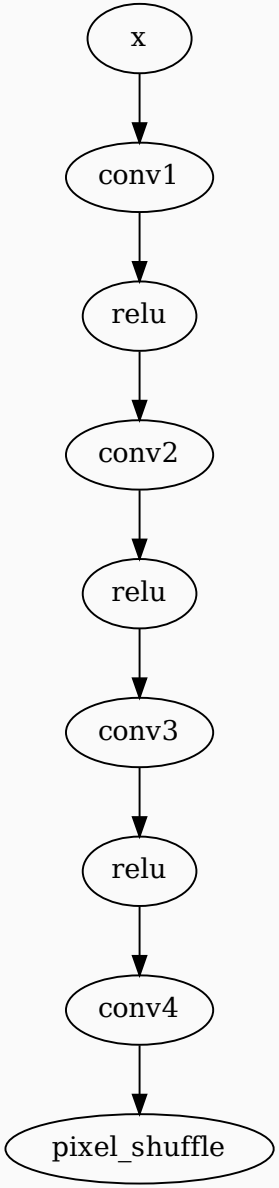
```
X = A * B
Y = C * D
```

- Only operation can execute at a time

Ampere resources



Example #1



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

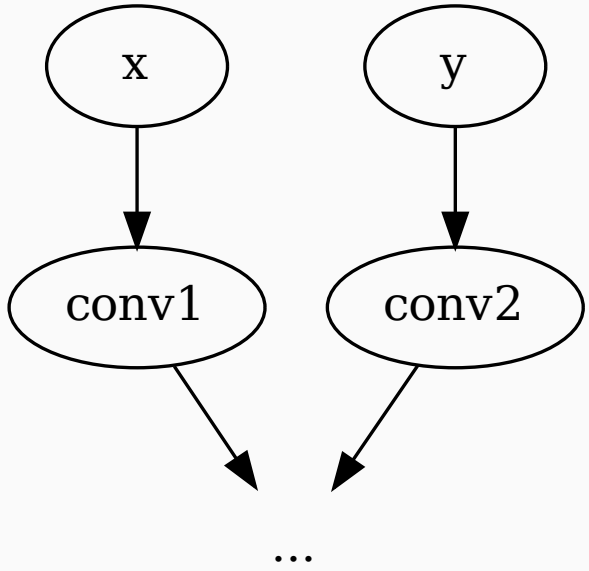
Intra-operator parallelism

- 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.

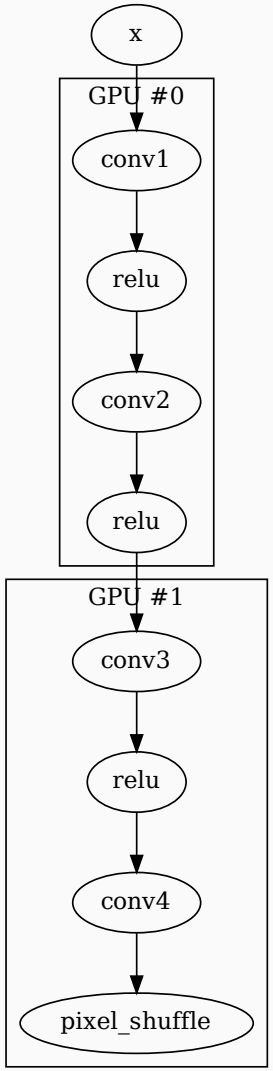
Inter-operator parallelism

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



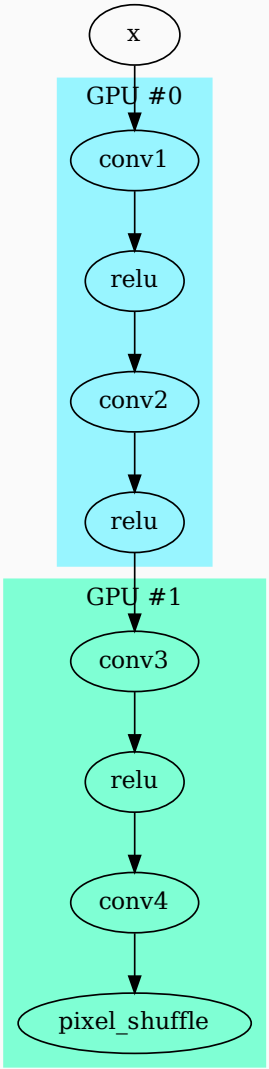
Partitioning ('Model Level Parallelism')

- 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



Breaking Data Dependences

- Dependences can be broken in many ways
  - Ignore them
  - Operate on stale data
  - Speculate on data values
- This usually recovers recovery mechanisms

Outline

What is scalability?

- A service has to respond to increases in "load", for example:
  - the number of customers, or
  - the number of inputs, or
  - the size of inputs
- Usually this increase is handled by increasing the resources available to a service
- A service is scalable if increases in load can be handled:
  - through proportional linear or sublinear increases in resources

Strong Scaling: Amdahl's Law

$$\text{Speedup} = \frac{T_1}{T_P} = \frac{1}{\frac{\alpha}{P} + (1 - \alpha)}$$

where:

- α is the fraction of the program that can parallelized
- P is the number of processors (or parallel units).
- T_1 is the time taken on 1 processor (single-processor runtime)
- T_P is the time on P processors

Serial Bottleneck

As $P \to \infty$, we have:

$$\text{Speedup}_{\infty} = \frac{1}{(1-\alpha)}$$

- A program that has a serial portion of 20% (i.e. $\alpha = 0.8$), has a maximum speedup of 5.

Where do the serial portions come from?

- Essentially, due to queueing (or serialization)
  - I/O: loading data from a file before handing it off to threads
  - Synchronization: locks implicitly introduce serialization
  - Resource contention: e.g., all data requests go to the same machine

Weak Scaling: Gustafson's Law

Let T_N be time for N processors, with s the execution time for the serial portion and p the time for parallel execution. Assume the times are normalized so $s + p = 1$.

Serial execution time $T_1 = s + Np$ (assuming linear slowdown).

$$\mathrm{Speedup(N)} = s + Np$$

Implications of Gustafson's Law

- Amdahl's law keeps the problem size constant
  - Keeps increasing machine size
- Gustafson's law increases problem size
  - And also increases machine size
- Informally, a bigger machine can solve a bigger problem in the same amount of time
  - I.e., if you want to use more parallelism, increase the problem size
  - E.g, by batching
- Key assumption: serial portion takes the same time regardless of number of processors