CSC290/420 Machine Learning Systems for Efficient AI
Bulk Data Transfers / Communication

Sreepathi Pai

October 22, 2025

URCS

Outline

Single-Machine Communication

Networked Communication

Collective Communication Algorithms

Communication in ML Programs

Outline

Non-bulk data transfers: Loads and Stores

- Load and Store instructions can be used to transfer data
- Special loads: non-temporal loads and stores
  - Hint to the processor that these should not be cached
- Most computers are stall-on-use for loads
  - Other instructions (or hardware threads) are executed without waiting for a load

Computation and Communication

- Communication is seen as I/O
  - traditionally considered "slow" compared to CPU
- Using loads/stores uses CPU for communication
- Perform communication using a dedicated unit
  - e.g., DMA engine
  - Still slow
- Goal is not make CPU wait for communications
  - overlap communication with computation

Asynchronous Notifications: Interrupts

- When a DMA engine completes a data transfer, it raises an interrupt that alerts the CPU
- The OS then translates that interrupt into a signal for a program
- A signal causes the program to run a pre-registered signal handler
  - interrupts whatever it was doing
  - resumes after signal has been handled

Synchronous Notification: Polling

- The CPU can also periodically check if the data transfer has completed
  - possibly by examining some register or flags on the DMA engine
- Polling continously can be slow and can waste CPU cycles
- However, at high enough communication rates, polling is usually better than interrupts
  - rise of user-space networking (i.e., OS is not involved)

Pinned Memory

- DMA engines have traditionally required physical addresses
  - however, newer DMA engines can deal with virtual addresses as well
- The mapping between virtual to physical memory must remain unchanged during the DMA operation
- Achieved by "pinning" the page
  - prevents the OS from swapping the page out
- These pinned pages are sometimes called "bounce buffers"
- Aside: GPUs can load/store directly from pinned pages in CPU memory

Zero Copy

- Traditional data transfer: device to bounce buffer (OS kernel memory) to user buffer
  - or in the reverse direction
- Can a device directly send data to user buffer bypassing the OS kernel?
  - or can a program send data directly to device?
  - devices: network cards, disks, GPUs
- Example: copying a file using `fread` and `fwrite` vs `sendfile` (on Linux)

Cross-Device Transfers

- Data from a source device to a destination device doesn't need to go through the CPU
- Devices: Network cards (NICs), Disks, GPU [memory]
- Various proprietary technologies
  - e.g. NVIDIA's GPUDirect

Outline

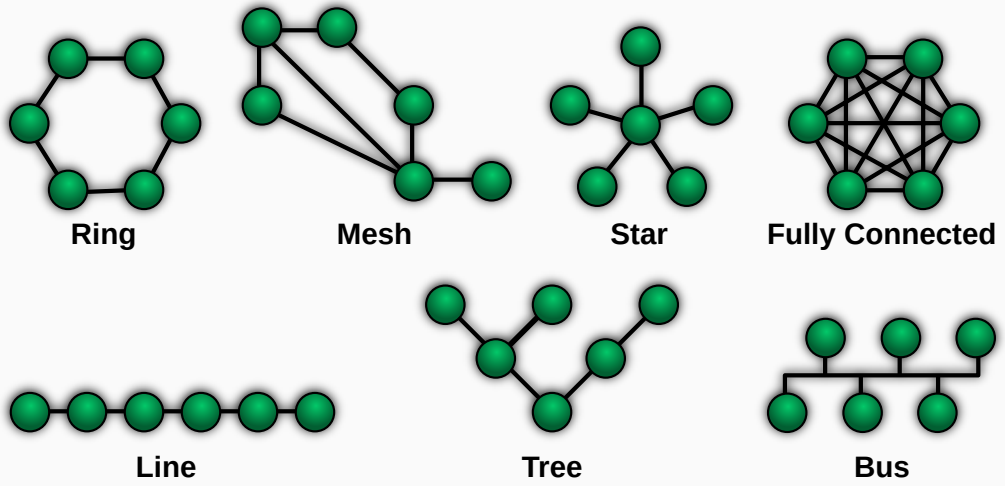
Multi-machine Communication

- Physical Layer: Electrical, Radio, Optical Fibre
  - Different bandwidth, latency, routing characteristics
- Data Link Layer: Ethernet, 802.11, InfiniBand
- Network layer: IPv4, IPv6
- Transport layer: e.g, TCP, SCTP
- Application layer: e.g., HTTP

Physical Network Topologies

- How different nodes of the network are wired up
  - alternatively, physical paths between nodes

NetworkTopologies.png: Maksim / derivative work: Malyszkz, Public domain, via Wikimedia Commons



Topology Characteristics

- Diameter
  - Max number of hops between two nodes
- Bisection Width
  - Minimum number of links to be cut to bisect the network into two
- Bisection Bandwidth
  - The aggregate bandwidth of the the bisecting links
  - How much bandwidth can one half of nodes use to send data to the other half?

Logical Network Topologies

- The topology of a network is of interest to communication algorithm designers
- Often, a logical topology is overlaid on the physical network as an abstraction
- Example: logical ring even if the underlying physical network is not a ring

Communication Modes

- Unicast (or Point to Point)
  - One-to-one
  - Sent once, received once
- Broadcast
  - One-to-all
  - Sent once, received by each device once
- Multicast
  - One-to-many
  - Many-to-many
  - Sent once, Received once by each device in a multicast group
- Anycast
  - One-to-one (of many)
  - Sent once, received once by one member of a group

What are the performance implications?

Protocol Characteristics

- Connected / Non-connected
  - must a connection be set up before sending data?
- Reliable / Unreliable
  - are errors in transmission detected and reported?
- Ordered / Unordered
  - is send order respected by receiver?
- Stream / Datagram
  - is data broken up into chunks or sent as a continuous stream of bytes?

What are the performance implications?

Low-Level Programmer Interfaces

- (Unix) Sockets
  - available on non-Unices like Windows too
  - send, recv, poll, events
- Device-specific mechanisms
  - RDMA (a moniker for a large range of networking technologies)
  - cudaMemcpy[Async] (and its equivalents on other GPU programming models)

Outline

Collective Communication Algorithms

- Basic Communication Primitives: send, recv
  - synchronous and asynchronous
- Collective Communication Algorithms
  - an often multi-step data transfer algorithm performed by all communicating devices
  - some collective communication algorithm also perform computation (e.g., $+$)
- Collective communication can be used by parallel algorithms
  - often supported by hardware (esp. in supercomputers)

AllReduce

```
% executed by all machines
% not an efficient implementation!
% assuming async send
for m in other_machines:
   send(m, mydata)

output = mydata

for m in other_machines:
    data = recv(m)
    output += data

return output
```

With support for `AllReduce`, this code becomes:

```
output = AllReduce(mydata)
```

Other CC Algorithms

- Broadcast*
- Reduce*, All-Reduce
- Gather*, All-Gather
- Scatter*
- Barrier

Operators marked with a * have a distinguished participant whose behaviour is different from others. e.g., the sender of the broadcast

Using CC algorithms

- MPI (traditional)
  - Doesn't support GPUs very well
- NCCL
  - NVIDIA-specific, built only for GPU-to-GPU communication
- RCCL
  - AMD's equivalent of NCCL (based on it)
- MSCCL
  - Microsoft's own library
- many others, strong area of interest because 40% of training costs is communication
  - and this cost is increasing

Illustration of many CC algorithm behaviours

See: NVIDIA NCCL's Collective Operations page

Outline

Training and Inference

- Distributed Parameter Training
  - Data is split up among different machines which train on their own subset
  - A parameter server is a central location for these individual subsets to be combined
- Pipeline Parallelism
  - Data from one stage to another esp. if the stages are on different GPUs
- Model Parallelism
  - Data from one partition of the graph to another

ML-specific Communication Optimizations

- ML computations can be approximated
  - without significant loss in accuracy
- ML communications can be reduced by not communicating every step
  - e.g., in pipelined algorithms
  - i.e., operate on stale data, see Li et al., DistriFusion: Distributed Parallel Inference for High-Resolution Diffusion Models
- Asynchronous communication seems not to hurt
  - e.g., Hogwild Stochastic SGD
- Compression of data
  - Different compression schemes for parameters, activations, inputs, outputs