Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

Tal Ben-Nun and Torsten Hoefler
ETH Zurich

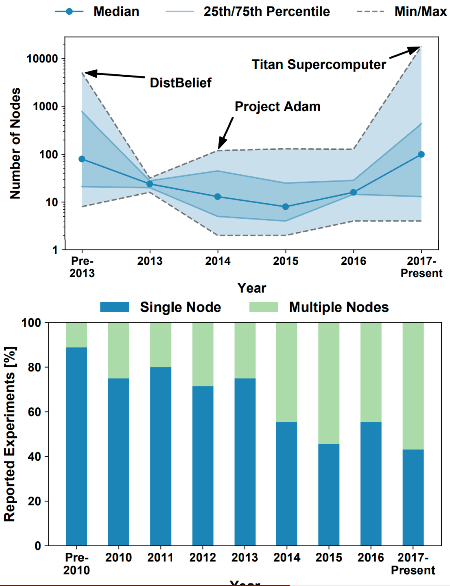
Presenter : Derrick Blakely
https://qdata.github.io/deep2Read

March 6, 2019

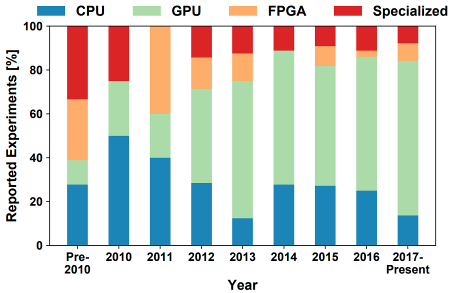
Outline

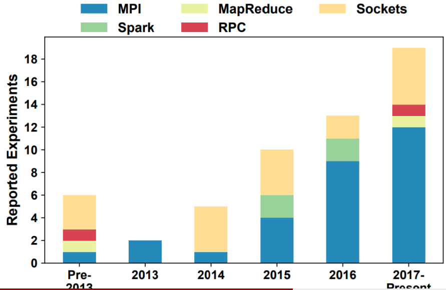
Outline

Deep Learning is Using More Nodes



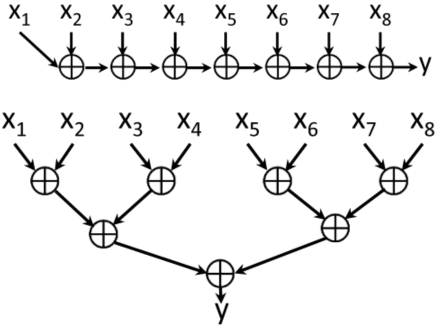
More GPUs, More MPI





Outline

Parallel Computing Basics: Computation Graphs



- Critical path: causes the earliest possible completion time
- Depth D: time needed to execute critical path
- $T_1 = W$: sequential execution; same as total work
- $T_\infty = D$: with unlimited processors
- Average parallelism: $T_1 / T_\infty = W / D$

Parallel Computing Basics: Granularity

- Granularity: $G \approx T_{comp}/T_{comm}$
- Alternatively:

$$G = \frac{min_{n \in V} w(n)}{max_{e \in E} c(e)}$$

Parallel Computing Basics: Granularity

- Granularity: $G \approx T_{comp} / T_{comm}$
- Alternatively:

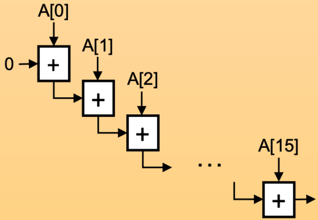
$$G = \frac{min_{n \in V} w(n)}{max_{e \in E} c(e)}$$

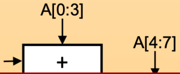
Parallel Computing Basics: Granularity

- Granularity: $G \approx T_{comp} / T_{comm}$
- Alternatively:

$$G = \frac{min_{n \in V} w(n)}{max_{e \in E} c(e)}$$

```
c = 0;
For (i=0; i<16; i++)
    c = c + A[i]
```





Goals

- Maximize parallelism
- Minimize communication and load imbalance
- Tradeoff: high parallelism, low communication
- Tradeoff: high load balance vs low communication
- High parallelism and high load balance are often compatible

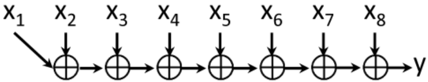
Communication

Want to perform the following AllReduce:

$$y = x_1 \oplus x_2 \oplus \ldots \oplus x_{P-1} \oplus x_P$$

- P: num processing elements
- x_i: length-m vector of data items stored on a processing element
- γ: size of a data item (bytes)
- G: computation cost per byte
- L: network latency

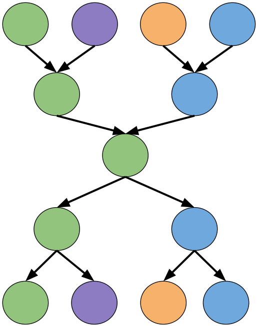
Naive: Linear-Depth Reduction

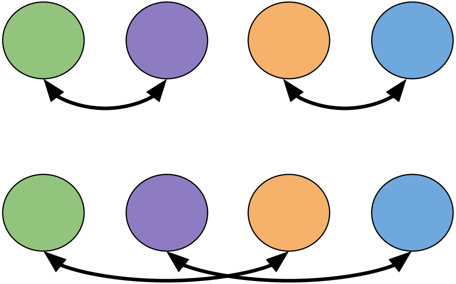


$$T = \gamma m G(P-1) + L(P-1)$$

Average Parallelism: $T_1 / T_\infty = W/D = (P-1)/(P-1) = 1$

Tree and Butterfly AllReduce

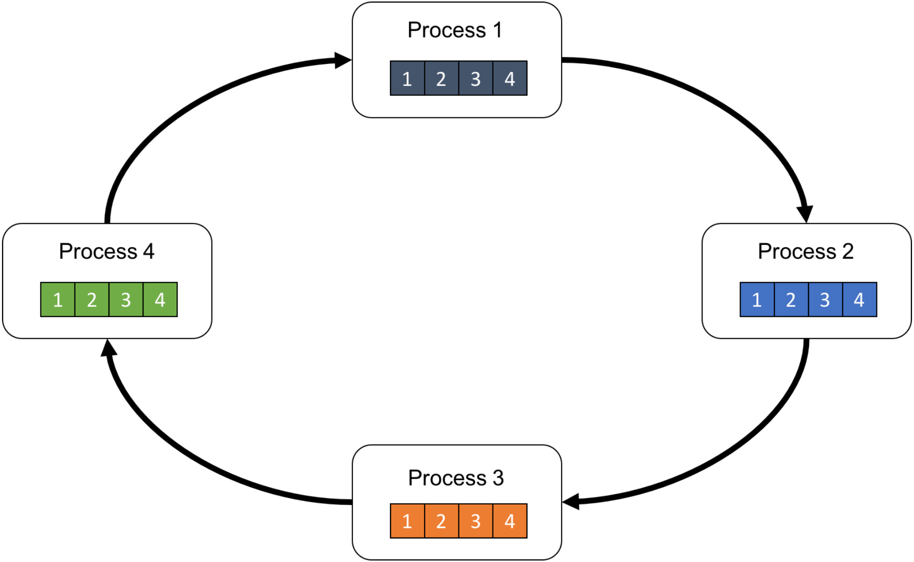




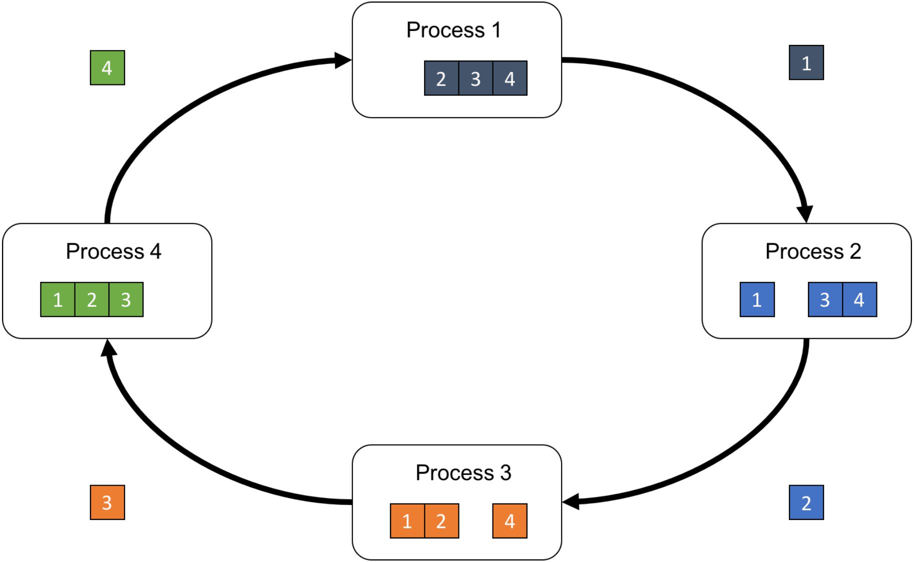
$$T = 2\gamma m G \log P + 2L \log P$$

$$T = \gamma m G \log P + L \log P$$

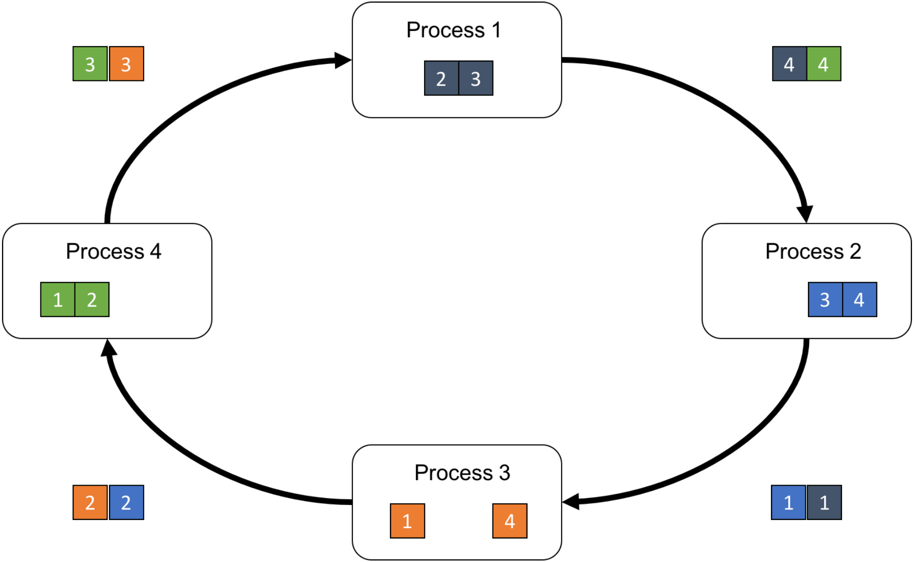
Ring (or Pipeline) AllReduce



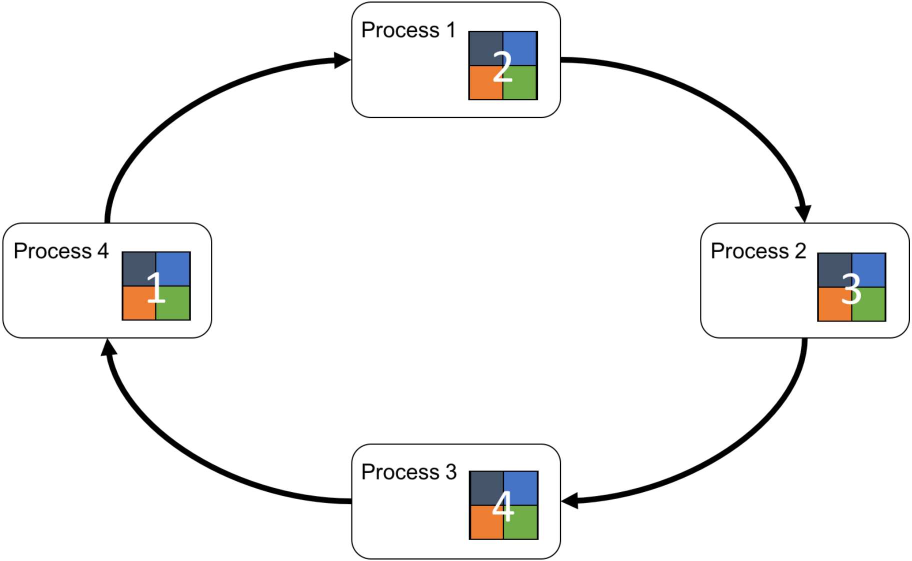
Ring (or Pipeline) AllReduce



Ring (or Pipeline) AllReduce



Ring (or Pipeline) AllReduce



AllReduce

Ring:

$$T = 2\gamma mG(P-1)/P + 2L(P-1)$$

Reduce-Scatter-Gather:

$$T = 2\gamma mG(P-1)/P + 2L\log P$$

Lower bound:

$$T \geq 2\gamma mG(P-1)/P + L\log P$$

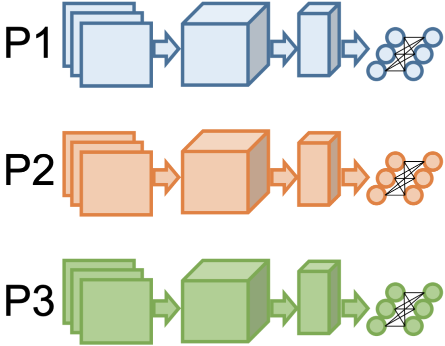
Outline

Within-Layer Concurrency

Table 4. Asymptotic Work-Depth Characteristics of DNN Operators

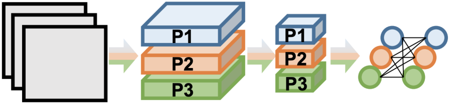
Operator Type	Eval.	Work (W)	Depth (D)
Activation	y	$O(NCHW)$	$O(1)$
	∇w	$O(NCHW)$	$O(1)$
	∇x	$O(NCHW)$	$O(1)$
Fully Connected	y	$O(C_{out} \cdot C_{in} \cdot N)$	$O(\log C_{in})$
	∇w	$O(C_{in} \cdot N \cdot C_{out})$	$O(\log N)$
	∇x	$O(C_{in} \cdot C_{out} \cdot N)$	$O(\log C_{out})$
Convolution (Direct)	y	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
	∇w	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
	∇x	$O(N \cdot C_{out} \cdot C_{in} \cdot H \cdot W \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$

Data Parallelism



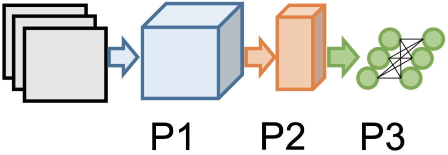
- Simple and efficient
- Must replicate models, possible GPU out of memory

Model Parallelism



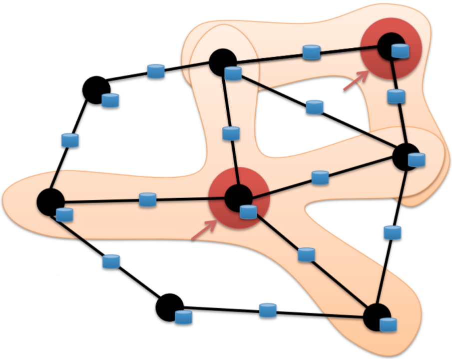
- Conserve GPU memory
- Can work well with multiple GPUs on the same system
- Must share minibatch with every worker
- Back-prop requires all-to-all communication

Layer-by-layer Concurrency: Pipelining



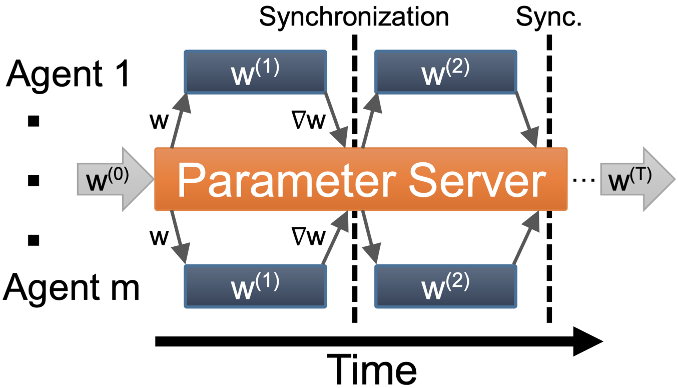
- Avoid out of memory errors
- Sparse communication: GPUs only communicate with GPU in front of them
- Have to make sure there is overlap in computation
- Latency linear in number of processors

Graph Parallelism



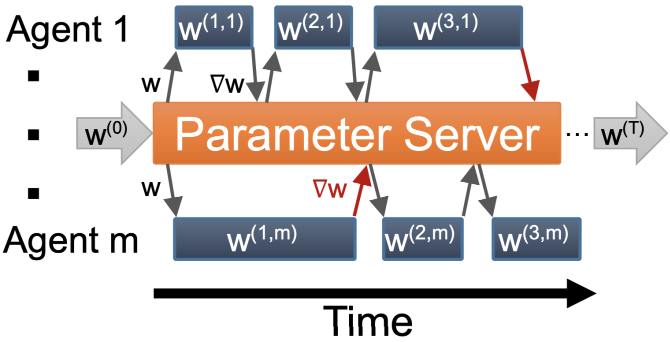
Outline

Centralized Parameter Sharing: Parameter Server



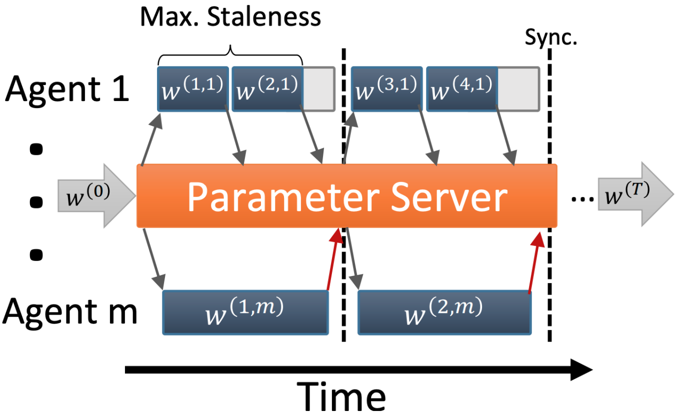
- $T = 2P\frac{\gamma m G}{s} + 2L$
- Ensures consistency
- "Stragglers" cause poor utilization

Asynchronous Parameter Server



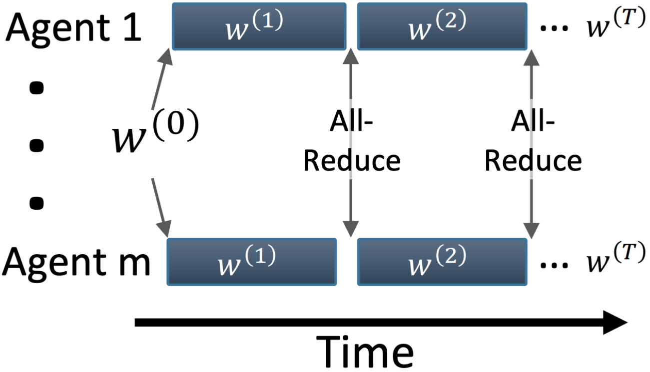
- Better utilization, faster training
- Slow agents cause parameter divergence

Stale Synchronous Parameter Server



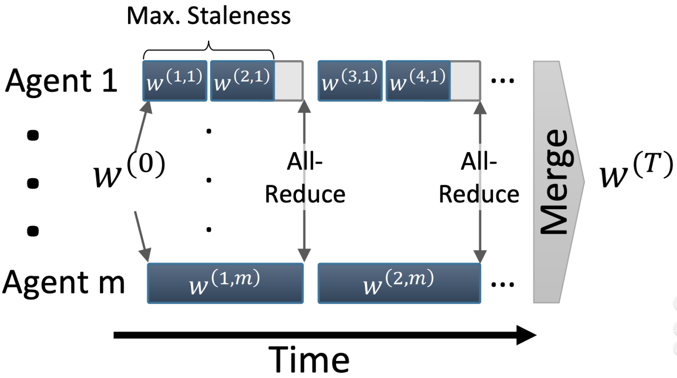
- Statistical performance vs hardware performance tradeoff
- Having a parameter server at all can bottleneck training

Decentralized Parameter Sharing

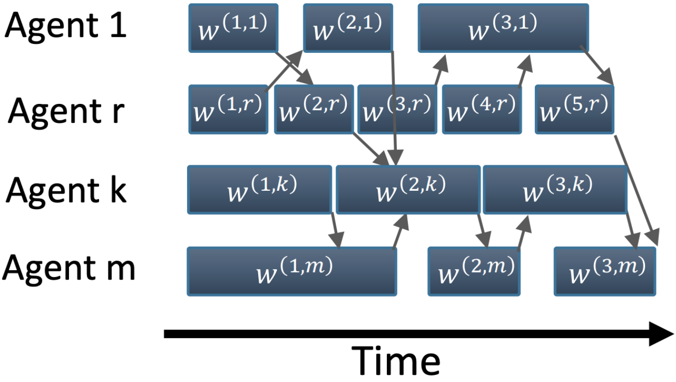


- $T = 2\gamma mG(P-1)/P + 2L \log P$
- MPI or NCCL can automatically provide a good AllReduce
- Avoids parameter server bottleneck
- "Straggler" problem

Stale Synchronous Decentralized Parameter Sharing



Asynchronous Decentralized Parameter Sharing



Outline

Neural Net Frameworks

- TensorFlow: allows Parameter Server and AllReduce (MPI, NCCL, TCP/IP)
- PyTorch: AllReduce with MPI, NCCL, or gloo

Why not use Hadoop or Spark?

- Don't natively support GPU runtimes
- Require JVM–slow
- Designed for fault-tolerance, not speed
- Only support data-parallelism
- NNs have cyclic computation graphs (must revisit working sets)
- Must be synchronous
- TF or PyTorch better optimized for deep learning

Outline

Ongoing Distributed ML Challenges

1. Optmization and Theory
2. Make algorithms more scalable
3. Better software for distributing ML
4. Develop distributed ML systems
    - Consistency
    - Fault tolerance
    - Communication latency
    - Resource management

Up-and-Coming ML Topics

- Impact of compression and quantization?
- Challenges unique to networks with dynamic control flow?
- Best ways to implement utilize graph parallelism?
- Hierarchical tasks?
- Automated architecture searches?

References

Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." arXiv preprint arXiv:1802.09941 (2018).