Accelerating Distributed Inference of Sparse Deep Neural Networks via Mitigating the Straggler Effect

Mohammad Hasanzadeh Mofrad, Rami Melhem,
Yousuf Ahmad and Mohammad Hammoud

Github Repository: https://github.com/hmofrad/DistSparseDNN

UNIVERSITY OF PITTSBURGH

Carnegie Mellon University Qatar

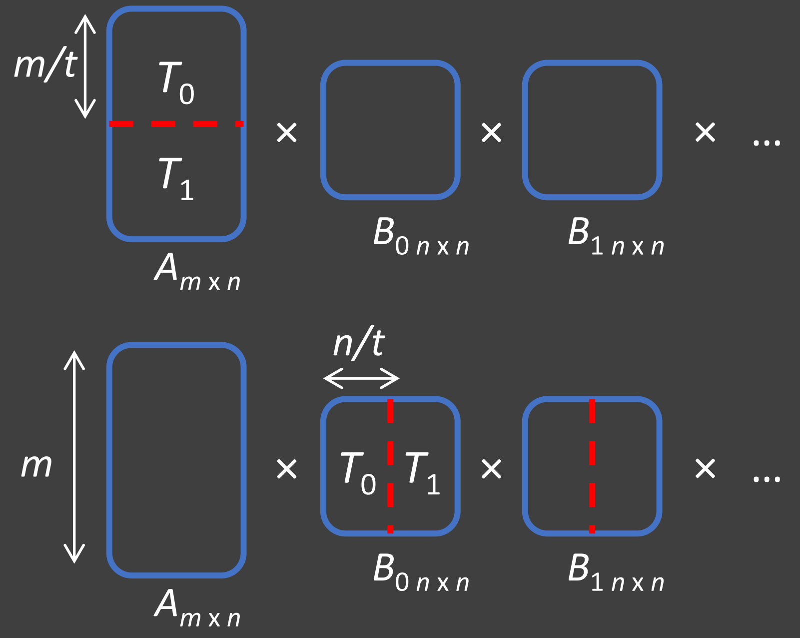
IEEE HPEC

Dense and Sparse Neural Networks

- **Deep Neural Networks (DNNs)** are pervasive
  - Speech processing
  - Friend suggestion
  - Autonomous driving
  - Item recommendation
- The core kernel behind inference/training of DNNs is **Dense matrix-matrix multiplication**
  - $C_{m \times n} = A_{m \times n} \times B_{n \times n}$
- **Sparse DNNs** are new alternative to dense DNNs with
  - Less time and space complexities
  - Comparable accuracy
- Sparse DNNs core kernel is **Sparse Matrix-matrix Multiplication (SpMM)**

Multithreaded Single Machine (Sparse) DNN Inference

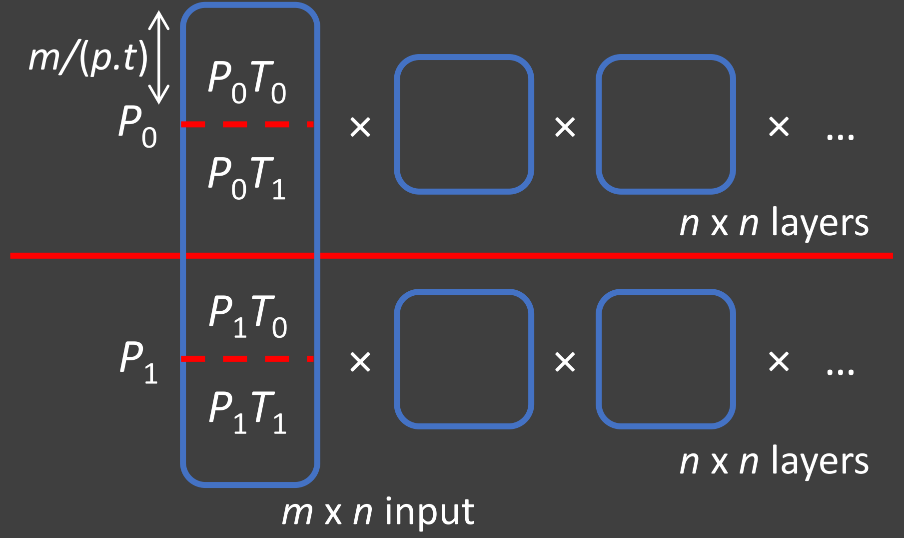
- **Data Parallelism**
  - Horizontal 1D-Row partitioning of input (A)
    - *t* input partitions where *t* is the number of threads
  - + No synchronization, – stragglers, – bad L3 utilization
- **Model Parallelism**
  - Vertical 1D-Column partitioning of network (B)
    - *t* network partitions
  - – Strict synchronization, + better L1 and L3 utilization
- **Manager-worker** (single queue of *m*t* i.e. *m>>t*)
- **Work-Stealing** (*t* queues of *m* tiles)



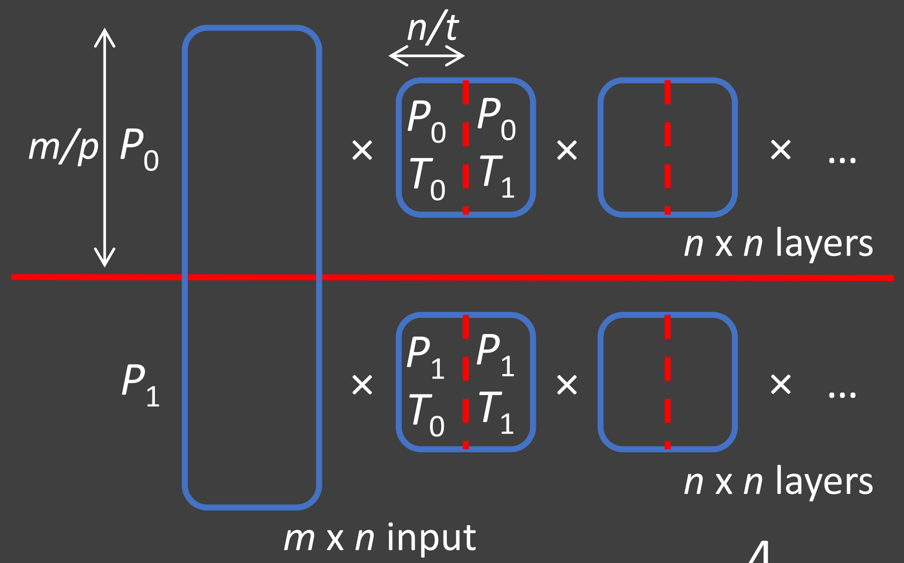
- SpMM algorithm: Two-step right SpMM with CSC

Distributed (Sparse) DNN Inference

- **Data * Data parallelism**
  - Horizontal 1D-Row partitioning of input (A)
  - *p.t* input partitions
  - *p* is the number of processes



- **Data * Model Parallelism**
  - vertical 1D-Column partitioning of network (B)
  - *p.t* network partitions



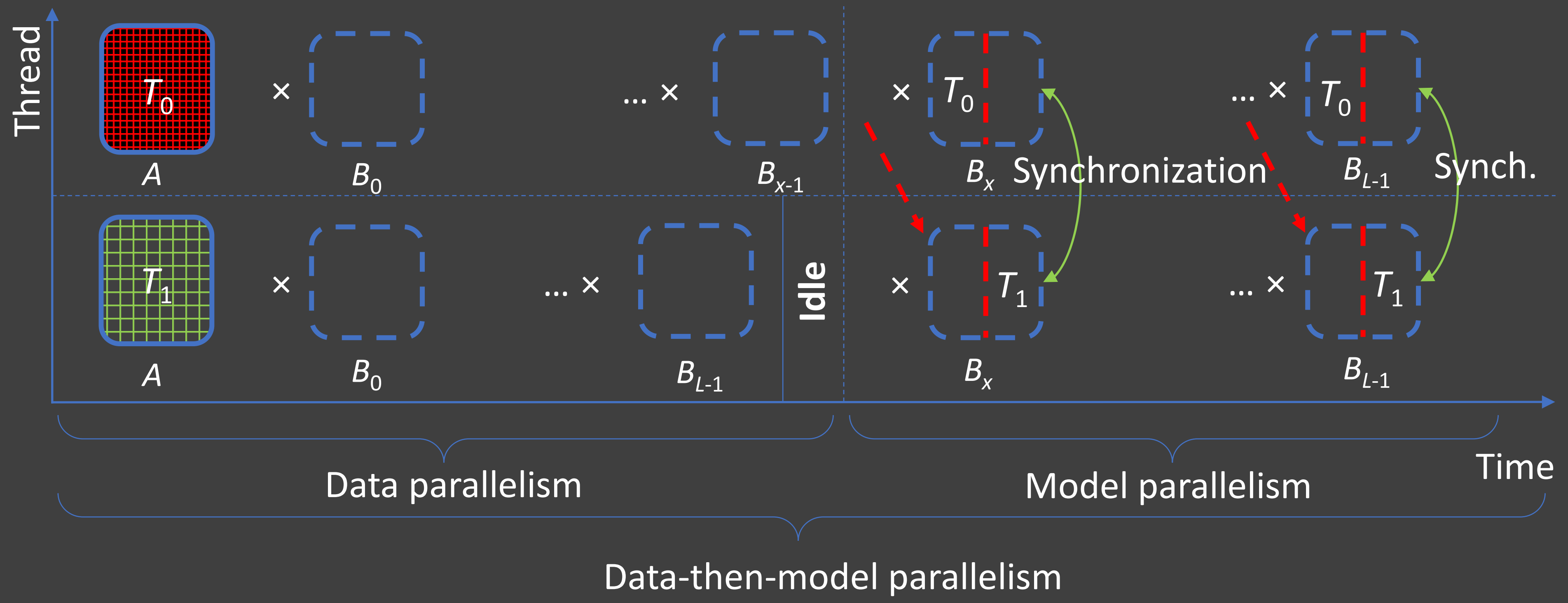
- Data * Manager-worker, and Data * Work-stealing
- Network is replicated for each process
- No communication is happening among processes

Data-then-model Parallelism

- Due to imbalance Data parallelism suffers from straggler effect
- Imperfect solution is **hashing**

- Switch from data to model parallelism and turning idle threads into additional processing power to mitigate the effect of stragglers

- Lazy load balancing by reusing idle threads
- Zero data movement
- No contention
- Less synchronization cost



Experiments

Dataset (RadixNet Sparse DNN, MNIST)					
Input		Network			
		Each Layer		All Layers	
Size $_{m \times n}$	NNZ	Size $_{n \times n}$	NNZ	L	NNZ
60K x 1K	6.3M	1K x 1K	32K	120	3.9M
				480	15.7M
				1920	62.9M
60K x 4K	25M	4K x 4K	131K	120	15.7M
				480	62.7M
				1920	251M
60K x 16K	99M	16K x 16K	524K	120	62.9M
				480	251M
				1920	1B
60K x 65K	392M	65K x 65K	21 M	120	251M
				480	1B
				1920	4B

Node Specification (16)	
CPU	28-core @ 2.6 GHZ
Memory	192 GB
OS	Linux
MPI	Intel
Network	Intel Omni-path Fabric

Parallelisms
Data Parallelism
Model Parallelism
Data-then-model Parallelism
Manager-worker Parallelism
Work-stealing Parallelism

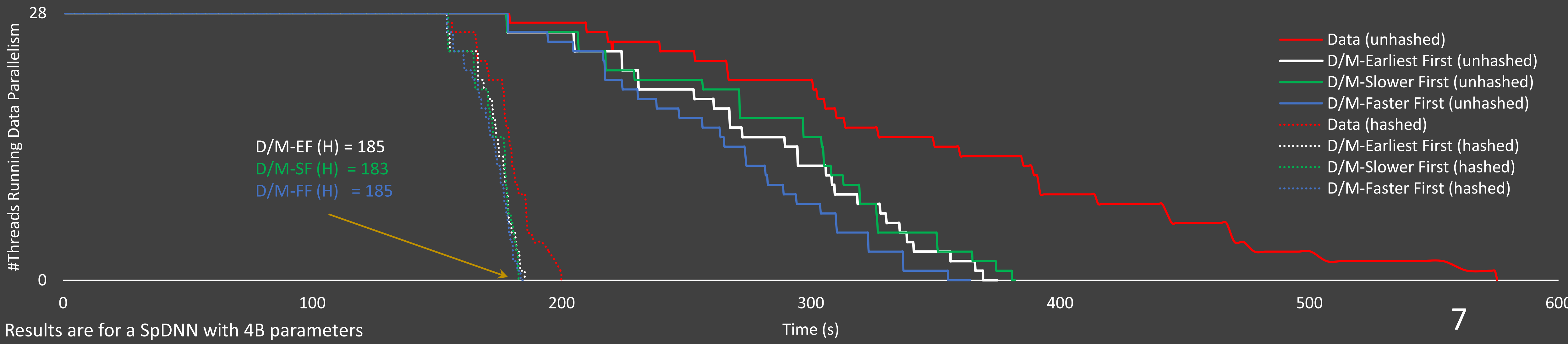
Experiments: Data-then-model Parallelism Thread Scheduling Algorithms

Locking Mechanism Runtime

- Threads are either
  - Worker (active) or helper (newly idle)
- An idle thread enlists into an **idle queue**
- A **working thread** probes the idle queue
  - **Helper threads** get recruited by a working one

Advantages

+ Overloadable with scheduling strategies
  - Earliest first, slower first, and faster first
+ Fully decentralized and asynchronous
+ Minimal lock contention
  - condition variables + locks
+ Elastic: adding/removing threads on the fly
  - zero data movement



Results are for a SpDNN with 4B parameters

Experiments: Scalability

Questions?