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
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.

Lazy load balancing by reusing idle threads.
Zero data movement.
No contention.
Less synchronization cost.
# Experiments

## Dataset (RadixNet Sparse DNN, MNIST)

<table>
<thead>
<tr>
<th>Input</th>
<th>Network</th>
<th>All Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size $m \times n$</td>
<td>NNZ</td>
<td>L</td>
</tr>
<tr>
<td>Each Layer Size $n \times n$</td>
<td>NNZ</td>
<td></td>
</tr>
<tr>
<td>60K x 1K</td>
<td>6.3M</td>
<td>120</td>
</tr>
<tr>
<td>1K x 1K</td>
<td>32K</td>
<td>480</td>
</tr>
<tr>
<td>1920</td>
<td>62.9M</td>
<td></td>
</tr>
<tr>
<td>60K x 4K</td>
<td>25M</td>
<td>120</td>
</tr>
<tr>
<td>4K x 4K</td>
<td>131K</td>
<td>480</td>
</tr>
<tr>
<td>1920</td>
<td>251M</td>
<td></td>
</tr>
<tr>
<td>60K x 16K</td>
<td>99M</td>
<td>120</td>
</tr>
<tr>
<td>16K x 16K</td>
<td>524K</td>
<td>480</td>
</tr>
<tr>
<td>1920</td>
<td>1B</td>
<td></td>
</tr>
<tr>
<td>60K x 65K</td>
<td>392M</td>
<td>120</td>
</tr>
<tr>
<td>65K x 65K</td>
<td>21 M</td>
<td>480</td>
</tr>
<tr>
<td>1920</td>
<td>4B</td>
<td></td>
</tr>
</tbody>
</table>

## 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?