Accelerating MapReduce-based Big Data Processing in Multi-GPU systems

Kuan-Ching Li
Providence University
Introduction

• Global data volume increases exponentially in explosive trend

• According to IDC, the total global data will reach to 45ZB (by 2020)

• Largest data generators: Large Hadron Collider (LHC), Medical Data, e-business, IoT, and gene sequencers
Today’s Way of Computing

- Illusion of infinite resources
- Pay only for resources used
- Quickly scale up or scale down …
It is needed an easy way for programmers to utilize resources of large distributed systems ...
MapReduce

• Well-known programming framework used to process the ever-growing data sets collected
• Although widely adopted in a number of data centers, more improvements are still needed to meet the huge demands of Big Data computing
• Three phases, and can be conducted in parallel
  — Map phase is CPU-intensive and the
  — Shuffle phase is I/O-intensive
MapReduce
GPUs

• Explosion of demand for sophisticated AI-enabled services

• Data sets are growing, networks are getting more complex, and latency requirements are to meet user expectations
FLOPS in CPUs and GPUs

Theoretical GFLOP/s at base clock

~10x

~7x

Source: Nvidia
## Performance

<table>
<thead>
<tr>
<th>NVIDIA Tesla Graphics Card</th>
<th>Tesla K40 (PCI-Express)</th>
<th>Tesla M40 (PCI-Express)</th>
<th>Tesla P100 (PCI-Express)</th>
<th>Tesla P100 (SXM2)</th>
<th>Tesla P100 (SXM2)</th>
<th>Tesla V100 (PCI-Express)</th>
<th>Tesla V100 (SXM2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>GK110 (Kepler)</td>
<td>GM200 (Maxwell)</td>
<td>GP100 (Pascal)</td>
<td>GP100 (Pascal)</td>
<td>GP100 (Pascal)</td>
<td>GV100 (Volta)</td>
<td>GV100 (Volta)</td>
</tr>
<tr>
<td>Transistors</td>
<td>7.1 Billion</td>
<td>8 Billion</td>
<td>15.3 Billion</td>
<td>15.3 Billion</td>
<td>15.3 Billion</td>
<td>21.1 Billion</td>
<td>21.1 Billion</td>
</tr>
<tr>
<td>CUDA Cores (Total)</td>
<td>2880</td>
<td>3072</td>
<td>3584</td>
<td>3584</td>
<td>3584</td>
<td>5120</td>
<td>5120</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>384-bit GDDR5</td>
<td>384-bit GDDR5</td>
<td>4096-bit HBM2</td>
<td>4096-bit HBM2</td>
<td>4096-bit HBM2</td>
<td>4096-bit HBM2</td>
<td>4096-bit HBM2</td>
</tr>
<tr>
<td>Memory Size</td>
<td>12 GB GDDR5 @ 288 GB/s</td>
<td>24 GB GDDR5 @ 288 GB/s</td>
<td>12 GB HBM2 @ 549 GB/s</td>
<td>16 GB HBM2 @ 732 GB/s</td>
<td>16 GB HBM2 @ 900 GB/s</td>
<td>16 GB HBM2 @ 900 GB/s</td>
<td>16 GB HBM2 @ 900 GB/s</td>
</tr>
<tr>
<td>L2 Cache Size</td>
<td>1536 KB</td>
<td>3072 KB</td>
<td>4096 KB</td>
<td>4096 KB</td>
<td>4096 KB</td>
<td>6144 KB</td>
<td>6144 KB</td>
</tr>
<tr>
<td>TDP</td>
<td>235W</td>
<td>250W</td>
<td>250W</td>
<td>250W</td>
<td>300W</td>
<td>250W</td>
<td>300W</td>
</tr>
</tbody>
</table>

Source: Nvidia
<table>
<thead>
<tr>
<th>Model</th>
<th>NVIDIA Tesla V100&lt;sup&gt;TM&lt;/sup&gt;</th>
<th>NVIDIA Tesla P100</th>
<th>NVIDIA Titan Xp</th>
<th>AMD Radeon Instinct MI25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core name</td>
<td>GV100</td>
<td>GP100</td>
<td>GP102</td>
<td>?</td>
</tr>
<tr>
<td>Transistor Count</td>
<td>21.1B</td>
<td>15.3B</td>
<td>128</td>
<td>?</td>
</tr>
<tr>
<td>Die Size</td>
<td>815mm&lt;sup&gt;2&lt;/sup&gt;</td>
<td>610mm&lt;sup&gt;2&lt;/sup&gt;</td>
<td>471mm&lt;sup&gt;2&lt;/sup&gt;</td>
<td>?</td>
</tr>
<tr>
<td>Manufacturing Process</td>
<td>TSMC 12nm FFR</td>
<td>TSMC 16nm FF+</td>
<td>TSMC 16nm FF+</td>
<td>GF 14nm LPP</td>
</tr>
<tr>
<td>Stream Processors</td>
<td>5120</td>
<td>3584</td>
<td>3840</td>
<td>4096</td>
</tr>
<tr>
<td>SM or CUs</td>
<td>80 SM</td>
<td>56 SM</td>
<td>60 SM</td>
<td>64 CU</td>
</tr>
<tr>
<td>FP32 Cores</td>
<td>5120</td>
<td>3584</td>
<td>3840</td>
<td>4096</td>
</tr>
<tr>
<td>FP64 Cores</td>
<td>2560 (1:2)</td>
<td>1792 (1:2)</td>
<td>120 (1:32)</td>
<td>? (1:16?)</td>
</tr>
<tr>
<td>Tensor Cores</td>
<td>840</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TATF</td>
<td>320</td>
<td>224</td>
<td>240</td>
<td>2567</td>
</tr>
<tr>
<td>ROIRBE</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boost Clock</td>
<td>1455MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP16 TFLOPS</td>
<td>30TFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32 TFLOPS</td>
<td>15TFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP64 TFLOPS</td>
<td>7.5TFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tensor TFLOPS</td>
<td>120TFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 Cache</td>
<td>6144KB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Size</td>
<td>16GB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Clock</td>
<td>1.6Gbps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Interface</td>
<td>4096GB HBM2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>9000GBs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCIE/AVLink</td>
<td>NVLink x12/PCI-E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDP</td>
<td>300W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**3X Faster on Deep Learning Training**

<table>
<thead>
<tr>
<th>GPU Model</th>
<th>Time to Solution (In Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100</td>
<td>6.5 Hours</td>
</tr>
<tr>
<td>P100</td>
<td>18 Hours</td>
</tr>
<tr>
<td>K80</td>
<td>34 Hours</td>
</tr>
<tr>
<td>CPU</td>
<td>361 Hours</td>
</tr>
</tbody>
</table>

CPU Server: Dual Xeon E5-2699 v4, 2.6GHz | GPU Servers add 8X Tesla K80, Tesla P100 or Tesla V100 | V100 measured on pre-production hardware | Workload: NMT, 13 epochs to solution.
Why Multi-GPU?

Source: Nvidia
Programmability and Performance

- Better programmability, scalability and acceleration of computationally demanding tasks
- GPUs' throughput-oriented computing design closely matches the characteristics of large-scale data parallel applications
- Large number of applications have been accelerated by GPU-related programming paradigms
  - CUDA
  - OpenCL
  - OpenCV
  - Matlab
Facts

• MapReduce have been implemented on parallel platforms such as cluster of servers, MC CPUs, MICs, Cell processors

• Implementations of MapReduce systems have put their efforts on a single GPU attached to a server, neglecting the multi-GPU platforms supported by advanced techniques of GPUs

• Many systems tend to use atomic operations (原子操作) in GPU global memory to handle the concurrent writes among multiple threads
  
  – such atomic operations may cause serialization of accesses to GPU memory → decrease overall performance dramatically
Map-Shuffle-Reduce Jobs

• As known, only 7% of jobs in MapReduce are reduce-heavy

• Map and Shuffle
  - CPU-intensive and I/O-intensive (can overlap)

• Centralized scheduler
  - Determine an execution order of jobs on Map pipeline and Shuffle pipeline

• Dependency relationship
  - The Map emits data at a given rate
  - Shuffle waits for the data emitted by Map and may be delayed by the scheduling policy
Proposed Research - Motivation

- Phases of MapReduce programming model with high degree of parallelism

- In average, more than 60% of execution time is spent in these phases

- GPU memory bw > \(10x \times \text{CPU memory bw}\) (bw = bandwidth)

- Motivation: parallelization
Goals

• Acceleration using multiple GPUs

• Joint scheduling optimization of overlapping map and shuffle phases

• To minimize the average job makespan
How about MapReduce over Multi-GPUs?
Proposed Research

- **Using multiple GPUs** to accelerate MapReduce
  - Computing power increases **linearly** with the number of GPUs

- Enhancements to **system bottlenecks** in the design using GPUs

- Keeping simple the MapReduce application design and implementation, **without** adding new instructions
Approach

- The inputs of all stages are processed in multiple GPUs simultaneously.
- Work stealing algorithm is used to assign input blocks onto GPUs (for load balancing).
- The Shuffle stage incurs all-to-all communication among workers.
  - For workers within one GPU, the communication is accomplished through shared memory and global memory.
  - For workers from different GPUs, GPUDirect enables remote GPU memory access without going through CPU memory.

To exploit inner features of GPUs!
Multi-GPU Utilization

Workflow on multiple GPUs
1. Multi-GPU Utilization
2. Multi-GPU Utilization
3. Multi-GPU Utilization
Machine Learning

To speed up machine learning applications by making available "more resources" at the problem in MapReduce-style processing
• Original algorithms do not utilize all CPU cycles
• Extensive research in numerical linear algebra on parallelizing these numerical operations and adapt them to the programming model
• Fine Tuning of MapReduce Applications, and seek for acceleration opportunities, with better exploitation of
  – types of memories
  – parallel kernel execution
The system may offer …

- Multi-GPUs are utilized to speed up MapReduce
- Load balancing is achieved by distributing computations based on the capacity of all GPUs
- Big data issue is addressed through CPU memory
  - Normally bigger and more extensible than GPU memory
  - Aggregate GPU memory is not the bottleneck anymore
- Serial atomic operations are replaced by a parallel alternative, the parallel prefix sum operation, for maximum performance gains
Experiments

- Testing applications
  - GPU version are implemented with CUDA and compiled with NVCC compiler
  - CPU versions are implemented with OpenMP to utilize all 24 CPU cores (full capacity)
K-means Clustering

• Characteristics
  o Computation intensive

• Testing data
  o Randomly generated from a $10k \times 10k$ square area with floating-point coordinates

• Options
  o Partial Reduce is used to reduce I/O
  o The test is set to 3 rounds for measuring performance
  o The cluster number is set to 24 for fair comparison between CPU and GPUs, since there are totally 24 CPU cores
K-means Clustering

- Generated up to 520 millions points
- Double-GPU version achieves 91.7x speedup over CPU version and 1.7x speedup over single-GPU version
- Shuffle can almost be ignored because of the Partial Reduce
Integer Series Pattern

- Problem description
  - Find out the most common pattern in an integer series
  - Example
    - The maximum length of a pattern is limited to 2
    - Input: 1, 2, 3, 2, 3, 1, 3
    - Output: the most common pattern is (2, 3) which is appeared 2 times

- Usage example
  - For an article with all the words hashed into integer, it can be used to find out the most common phrase pattern
Integer Series Pattern

- Characteristic
  - I/O intensive

- Testing data
  - Randomly generated integers

- Options
  - Partial Reduce is used to reduce I/O
  - Developed as a three-pass MapReduce
    - The first two passes count 2-word and 3-word phrases separately.
    - Both results are used as the input for the third pass
Int Series Pattern

- Generated up to 5375 MB integer data
- Double-GPU version achieves 12.6x speedup over CPU version and 1.5x speedup over single-GPU version
- Shuffle can almost be ignored because of the Partial Reduce
Conclusions

- GPU matches MapReduce
  - Both designed for data parallelism

- Utilize Multi-GPU
  - Increase computing power
  - Reduce I/O among machines in the network

- Improve resource utilization and performance acceleration