Hybrid Map Task Scheduling for GPU-based Heterogeneous Clusters

Koichi Shirahata*  Hitoshi Sato*  Satoshi Matsuoka*†‡
*Tokyo Institute of Technology
†CREST, Japan Science and technology Agency
‡National Institute of informatics
The data generated by human activities is rapidly increasing

• Massive data processing
  – Various scientific computation (biology, astronomy, ...)
• MapReduce
  – Programming model for massive data processing
  – Scalable parallel data processing
• GPGPU
  – Better performance compared to CPU
  → Emerging of GPU-based Hybrid Supercomputer and Cloud
    ex) TSUBAME 2.0 : NVIDIA Fermi "Tesla M2050" x3 in a node

MapReduce acceleration by using GPUs
Problems of MapReduce on CPU-GPU Hybrid Clusters

- Scheduling Map tasks onto CPUs and GPUs efficiently is difficult

- Dependence on computational resource
  - # of CPU cores, GPUs, amount of memory, memory bandwidth, I/O bandwidth to storage

- Dependence on applications
  - GPU computation characteristic
    - Pros. Peak performance, memory bandwidth
    - Cons. Complex instructions

Hybrid Scheduling with CPUs and GPUs to make use of each excellence → Exploit computing resources
Goal and Achievement

• Goal
  – Acceleration of MapReduce in hybrid environment with CPUs and GPUs

• Achievement
  – Hybrid Map Task execution
    • Implemented on Hadoop, MapReduce OSS
  – Map Task Scheduling technique
    • Minimize total job execution time
  – Evaluation by K-means
    • Job execution time: 1.93 times faster by using multiple GPUs and proposed scheduling technique than CPU-only at 64 nodes.
Table of Contents

1. Background
2. MapReduce and GPGPU
3. Proposal
4. Design and Implementation
5. Experiments
6. Related work
7. Conclusion and Future work
MapReduce

• Data analysis, Machine learning applications
• Implementations
  – Hadoop: OSS of HDFS, MapReduce, HBase
  – Mars: framework for GPU
  → We implemented in Hadoop, widely used in many companies and institutes
GPGPU

• Graphic processors are used as SIMD
• Higher peak performance than CPUs
• Integrated developing environment
  – NVIDIA: CUDA
    • High level abstraction in a SIMD-style
    • Specific to NVIDIA GPUs
  – AMD: OpenCL
    • An open standard that can be used to program CPUs, GPUs from different vendors

→ We use CUDA, which provides C- and C++-based programming environment for NVIDIA GPUs
Table of Contents

1. Background
2. MapReduce and GPGPU
3. Proposal
4. Design and Implementation
5. Experiments
6. Related work
7. Conclusion and Future work
Structure of Hadoop

- Master/Worker model
  - Master: JobTracker
    - Manages submitted jobs
  - Worker: TaskTrackers
    - Execute map and reduce tasks
Idea: Hybrid Map task scheduling onto CPUs and GPUs

• Automatic scheduling onto CPUs and GPUs
  – A runtime environment, Computing resources
  – Application characteristics

→ Minimize the job execution time

• CUDA invocation from Hadoop
• Hybrid scheduling algorithm
CUDA invocation strategy from Hadoop

• Translation of a Java program to a CUDA code
  – Hadoop → Java (Middleware, Application)
  – CUDA → C or C++ library

• How to translate CUDA in Hadoop environment
  – Hadoop Streaming: Standard I/O
  – Hadoop Pipes: C++ library, Socket connection,
    – JNI, JNI-based CUDA wrapper (JCUDA)
  → We use Hadoop Pipes for our proposed technique
    • MapReduce applications/CUDA kernel → written in C++
CUDA invocation strategy from Hadoop (cont’d)

• Management of Map tasks, idle slots
  – Which slot each Map task should run on
  – Which CPU/GPU slots are idle

• Map task contention to GPU
  – When a TaskTracker node has Multiple GPUs
  – Management of which Map tasks run on which GPU devices
    • We set the GPU device number by cudaSetDevice() at the invocation of a GPU binary program
Hybrid scheduling strategy

• **Minimization of total job execution time**
  – Allocation of Map tasks by performance ratio of CPU and GPU map task execution (acceleration factor)

• **Dynamic monitoring**
  – Execution on both CPU and GPU map tasks simultaneously to collect profiles
  – Getting profiles of finished Map tasks on CPUs and GPUs periodically (e.g. execution time)
  – Calculation of the acceleration factor
  – Monitoring of the Job progress
Scheduling algorithm

• Goal
  – Minimize the time all the Map tasks are assigned
  – Calculate # of Map tasks to assign to CPUs, GPUs

• Acceleration factor
  \[ \alpha = \frac{\text{mean map task execution time on CPU cores}}{\text{mean map task execution time on GPU devices}} \]

• Scheduling algorithm
  Minimize
  \[ f(x, y) \]
  Subject to
  \[ f(x, y) = \max\left\{ \frac{x}{n} \cdot \alpha \cdot t, \frac{y}{m} \cdot t \right\} \]
  \[ x + y = N \]
  \[ x, y \geq 0 \]

Input
• CPU cores: \( n \), GPUs: \( m \)

Monitoring
• Remaining Maps tasks: \( N \)
• Runtime of 1 GPU task: \( t \)
• Acceleration factor: \( \alpha \)

Output
• Total Map tasks to run on CPUs: \( x \), on GPUs: \( y \)
Table of Contents

1. Background
2. MapReduce and GPGPU
3. Proposal
4. Design and Implementation
5. Experiments
6. Related work
7. Conclusion and Future work
How Hadoop Pipes works

- **Users**
  - write map/reduce functions (with C++ wrapper library)
  - specify compiled binary, and run the job

- **Execution overflow**
  - A Child JVM invokes map or reduce tasks on a TaskTracker
  - A C++ wrapper process send/receive key-value pairs to/from C++ binary via a socket
Hybrid execution by using Hadoop Pipes

• Specification of CPU/GPU binary when a job is launched
• TaskTrackers monitor the behavior of running map tasks
  – The elapsed time of a map task
  – The used CPU cores and GPU devices

- Specifying of CPU/GPU binary when a job is launched
- TaskTrackers monitor the behavior of running map tasks
  - The elapsed time of a map task
  - The used CPU cores and GPU devices
The map task scheduling workflow

1. Send profiles of Available slots, GPU devices, Mean runtime
2. Get # of Map Tasks, TaskTrackers, CPUs, and GPUs
3. Calculate $\alpha$
4. Allocate Map Tasks
5. Run Map Tasks
Table of Contents

1. Background
2. MapReduce and GPGPU
3. Proposal
4. Design and Implementation
5. Experiments
6. Related work
7. Conclusion and Future work
Experiments setting

• Measurement of the Job execution time on Hadoop-based environment
  – Comparison between
    • CPU-only and CPU + GPU hybrid execution
    • Hadoop original and proposed scheduling

• K-means application
  – Running Map tasks with C++ binary and CUDA binary
  – 20GB of files with 4000 sets of 262,144 floating points and 128 clusters
    • Map: executes K-means for each file
    • Reduce: collects the result of each K-means
Experimental environment

- **TSUBAME1.2**
  - We use up to 64 nodes (1024 CPU cores and 128 GPU devices)
  - Lustre file system as DFS (stripes: 4, Chunk size: 32 MB)
    - I/O performance: Write 180MB/s, Read 610MB/s (with 32 MB file)
  - Hadoop 0.20.1, Java 1.6.0, CUDA 2.3

### Specification of a single compute node

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td><strong>Dual Core AMD Opteron 880 (2.4 GHz)</strong></td>
</tr>
<tr>
<td># of cores</td>
<td>16</td>
</tr>
<tr>
<td>Main MEM</td>
<td>32GB</td>
</tr>
</tbody>
</table>

| GPU        | **NVIDIA Tesla S1070 (T10-based card × 4)** |
| # of cards | 2                                      |
| # of cores per card | 240 cores (1.29 - 1.44 GHz) |
| Global MEM per card | 4GB                                   |

| Interconnect | SDR Infiniband × 2 |
| PCI-Express Bandwidth | 2GB/s            |
| OS           | Linux 2.6.16      |
Comparison of Job execution time

Total MapReduce job execution time

- hadoop: Hadoop original scheduling
- hadoop-gpu2: Hadoop original scheduling with 2 GPU devices / node
- proposal-gpu2: Proposed scheduling with 2 GPU devices / node
Comparison of Job execution time

Although there are slight performance gain by using GPUs, it is almost equal to non-GPU version due to the small acceleration factor (x1.13)

- hadoop: Hadoop original scheduling
- hadoop-gpu2: Hadoop original scheduling with 2 GPU devices / node
- proposal-gpu2: Proposed scheduling with 2 GPU devices / node
Comparison of Job execution time

Further performance gain by applying proposed algorithm

- hadoop: Hadoop original scheduling
- hadoop-gpu2: Hadoop original scheduling with 2 GPU devices / node
- proposal-gpu2: Proposed scheduling with 2 GPU devices / node
Comparison of Job execution time

Performance degradation by adding nodes → Due to increase of Map task runtime

Total MapReduce job execution time

- hadoop: Hadoop original scheduling
- hadoop-gpu2: Hadoop original scheduling with 2 GPU devices / node
- proposal-gpu2: Proposed scheduling with 2 GPU devices / node
Increase of Map task runtime

• Map task runtime increases in proportion to # of nodes
  – Degradation of I/O performance
  – Since Lustre is configured with separated compute and storage node connected with shared networks
Comparison of job execution time
1 GPU / node with 2 GPUs / node

- hadoop: Hadoop original scheduling
- proposal-gpu1: Proposed scheduling with 1 GPU device / node
- proposal-gpu2: Proposed scheduling with 2 GPU devices / node
Comparison of job execution time
1 GPU / node with 2 GPUs / node

Our proposal achieves high application performance by using many GPUs

Total MapReduce job execution time
- hadoop: Hadoop original scheduling
- proposal-gpu1: Proposed scheduling with 1 GPU device / node
- proposal-gpu2: Proposed scheduling with 2 GPU devices / node
Overhead of process launching Experiment with 1 node

- Compare Map task binary runtime and Map task (total) runtime
  - Binary time: C++ or CUDA Map function execution time
  - Map task runtime: from Map task allocation to finish of the task

→ Significant overheads dependent on original Hadoop implementation

CPU-binary, GPU-binary : binary execution time
CPU-task, GPU-task : Map task runtime
Table of Contents

1. Background
2. MapReduce and GPGPU
3. Proposal
4. Design and Implementation
5. Experiments
6. Related work
7. Conclusion and Future work
Related work

• Several studies related to task scheduling or hybrid execution for heterogeneous environment
  – CPU/GPU task scheduling by learning mechanism [Chi-Keung Lu et al. `09]
  – Accelerate reduction computation with CPU/GPU hybrid execution by changing chunk size [T. Ravi Vifnesh et al. `10]
  – MapReduce task scheduling in heterogeneous environment [Zaharia et al. `08]

→ Massive data processing by CPU/GPU hybrid execution according to computing resource/application
  - Consider resource contention (e.g. memory, storage)
  - Auto-scheduling during execution
Table of Contents

1. Background
2. MapReduce and GPGPU
3. Proposal
4. Design and Implementation
5. Experiments
6. Related work
7. Conclusion and Future work
Conclusion and Future work

• Conclusion
  – Invocation of Map task on GPU from Hadoop
  – Task scheduling technique for GPU-based heterogeneous environment
  – Experiment by K-means application
    • 1.02-1.93 times faster by 2GPUs / node and proposed technique
    • Significant overhead dependent on Hadoop implementation

• Future work
  – Bottleneck analyses
    • TSUBAME 2.0, a new supercomputer in Tokyo Tech.
    • Comparison of Lustre and HDFS
  – Improvement of scheduling model
    • Resource contention issue including memory/disk access
    • I/O performance to storage system