GPGPU for Real-Time Data Analytics:
Case Studies– MapReduce (On-Demand Analytics)

Bingsheng He¹, Huynh Phung Huynh², Rick Siow Mong Goh²

¹Nanyang Technological University, Singapore
²A*STAR Institute of High Performance Computing, Singapore
Scope and Goals

• We use MapReduce as an example to demonstrate how GPGPU accelerates on-demand RTDA with MapReduce.
  – Single-GPU acceleration of MapReduce for speedup.
  – GPU integrations into Hadoop for scalability.
Outline

• Why MapReduce on GPUs
• Mars
  – Single-GPU implementation [He et al., PACT’08]
  – Multi-GPU implementation [Fang et al., TPDS’11]
• Other MapReduce Variants
• Summary
GPGPU Programming

• “Assembly languages”
  – DirectX, OpenGL
    • Graphics rendering pipelines

• “C/C++”
  – NVIDIA CUDA, OpenCL
    • Need to think in parallel.
    • Hardware specific optimizations on users.

• “Functional language”?
Without worrying about hardware details—
• Make GPGPU programming much easier.
• Well harness high parallelism and high computational capability of GPUs.

MapReduce
MapReduce Functions

• Process lots of data to produce other data
• Input & Output: a set of records in the form of key/value pair
• Programmer specifies two functions
  – map (in_key, in_value) -> emit list(intermediate_key, intermediate_value)
  – reduce (out_key, list(intermediate_value)) -> emit list(out_key, out_value)
MapReduce Workflow

From http://labs.google.com/papers/mapreduce.html
Outline

• Why MapReduce on GPUs

• Mars
  – Single-GPU implementation
  – Multi-GPU implementation

• Other MapReduce Variants

• Summary
MapReduce on GPU

Web Analysis | Data Mining | ...

**MapReduce (Mars)**

Rendering APIs (DirectX) | GPGPU languages (CUDA, Brook+)

GPU Drivers
Limitations on GPUs

• Rely on the CPU to allocate memory
  – How to support variable length data?
  – How to allocate output buffer on GPUs?

• Lack of lock support, and/or synchronization can be costly.
  – How to synchronize to avoid write conflict?
Data Structure for Mars

Support variable length record!

A Record = <Key, Value, Index entry>

<table>
<thead>
<tr>
<th>Key1</th>
<th>Key2</th>
<th>Key3</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Value2</td>
<td>Value3</td>
<td>…</td>
</tr>
<tr>
<td>Index entry1</td>
<td>Index entry2</td>
<td>Index entry3</td>
<td>…</td>
</tr>
</tbody>
</table>

An index entry = <key size, key offset, val size, val offset>
Lock-free scheme for result output

Basic idea:

Calculate the offset for each thread on the output buffer.
Lock-free scheme

1. Histogram on key size, value size, and record count.
2. Prefix sum on key size, value size, and record count.
3. Allocate output buffer on GPU memory.
4. Perform computing.

Avoid write conflict.
Allocate output buffer exactly once.
Mars Workflow

Input

MapCount

Prefixsum

Allocate intermediate buffer on GPU

Map

Sort and Group

ReduceCount

Prefixsum

Allocate output buffer on GPU

Reduce

Output
Optimization According to CUDA features

• Coalesced Access
  – Multiple accesses to consecutive memory addresses are combined into one transfer.

• Build-in vector type (int4, char4 etc)
  – Multiple small data items are fetched in one memory request.

• Shared memory
  – Exploit shared memory in GPU-based Bitonic Sort for Group Stage.
  – Users can explicitly utilize shared memory in their Map/Reduce functions.
MarsCPU vs. Phoenix

Overhead of Phoenix:
- Always need Reduce stage.
- Lock overhead.
- Re-allocate buffer on the fly.
- Insertion sort on static arrays.
- Call memmove() frequently.

There is a newer version of Phoenix.
MarsCUDA vs. MarsCPU

**Speedup**

\[ \text{Speedup} = \frac{T_{\text{MarsCPU}}}{T_{\text{MarsCUDA}}} \]

- Massive thread parallelism
- Memory bandwidth
Outline

• Why MapReduce on GPUs
• Mars
  – Single-GPU implementation
  – Multi-GPU implementation
• Other MapReduce Variants
• Summary
MarsHadoop: Integrating Mars into Hadoop

* Hadoop streaming
Outline

• Why MapReduce on GPUs
• Mars
  – Single-GPU implementation
  – Multi-GPU implementation
• Other MapReduce Variants
• Summary
Further Enhancements

• A number of projects that improve Mars and MarsHadoop
  – Single-GPU systems
    • MapCG [Hong et al. 10]
    • Shared memory usage [Ji et al. 11; Chen et al. 12]
    • StreamMR [Elteir et al. 11]
  – Multi-GPU systems
    • GPMR [Stuart et al. 11]
    • Pamar [Tan et al. 12]
    • Surena [Kruijf et al. 12]
    • Hybrid map scheduling [Shirahata et al. 10]
MapCG: MapReduce on CPUs and GPUs

- Map and Reduce
  - Based upon CUDA spec
  - Portable for both CPUs and GPUs
- MapCG generates CPU and GPU code with source code generation
- Runtime executes the Map Reduce code
  - CPU (OpenMP)
  - GPU (CUDA)
MapCG: MapReduce on CPUs and GPUs (Cont’)

• Specialized memory allocator
  – On CPU
    • One big malloc
    • Pheonix-2 uses allot of small mallocs: Inefficient, each intermediate key/value requires malloc
  – On GPU
    • CUDA has no support for on-device malloc (note: when MapCG was developed)
    • Mars uses a “count phase” (mapping is done twice)
    • MapCG uses one big buffer in global memory, and moves a “free space pointer” atomicAdd

• More memory efficient hash table
  – Phoenix and Mars sort the key/value pairs
  – MapCG hashes key/value pairs: More efficient
MapCG: MapReduce on CPUs and GPUs (Cont’)

Speedup of MapCG over Mars on GTX280 GPU, using Small, Medium, and Large datasets.

Chuntao Hong et al. Mapcg: writing parallel program portable between cpu and gpu. PACT ’10.
GPMR: Multi-GPU MapReduce on GPU Clusters

• A stand-alone library: not a fully-featured MapReduce implementation.
• Handle data movement, out-of-core data management, and maintain full GPU access.

Jeff A. Stuart and John D. Owens. Multi-gpu mapreduce on gpu clusters. IPDPS ’11.
GPU-enabled Hadoop

• Pamar [Tan et al. 12]
  – Uses JCUDA to automatically identify whether the task requires GPU resources.
  – Implements a FCFS scheduler that is aware of heterogeneous environments.

• Surena [Kruijf et al. 12]
  – Monitors and maximizes the GPU utilization.
  – Task resizing: select appropriate task granularity for a particular architecture.
Summary

• Heterogeneous platforms are common in current and future data analytics paradigms.
• GPUs are able to effectively accelerate MapReduce, which is the best-practice for on-demand RTDA.
• GPU-enabled Hadoop can be used in solving the on-demand RTDA problems in big data era.
Thank you and Q&A

Feedbacks are welcome:
Bingsheng He, bshe@ntu.edu.sg
Huynh Phung Huynh, huynhph@ihpc.a-star.edu.sg
Rick Goh Siow Mong, gohsm@ihpc.a-star.edu.sg
Tutorial site: http://www3.ntu.edu.sg/home/bshe/GPGPUTut.html