

Mars: Accelerating MapReduce on Graphics Processors

Wenbin Fang
wenbin@cse.ust.hk

HKUST

25 June, 2010

Mars: Accelerating MapReduce on Graphics Processors

What is Mars?

- A MapReduce Programming System, Map + Reduce.
- A Parallel Processing System accelerated by Graphics Processors (GPUs).
- Mars Modules running on:
 - An NVIDIA GPU: MarsCUDA
 - An AMD GPU: MarsBrook
 - A Multi-core CPU: MarsCPU
 - Multi-core CPUs + GPUs: Co-processing
 - Distributed System: MarsHadoop

Mars: Accelerating MapReduce on Graphics Processors

How Good?

- Ease of use. Up to **7 times** code saving.
- High performance. **An order of magnitude** speedup over a state-of-the-art CPU-based MapReduce system.

Agenda

- 1 Why Mars
 - GPGPU
 - MapReduce
- 2 How it works
 - Design
 - Implementation
- 3 Evaluation
 - Ease of use
 - High Performance
- 4 Conclusion

Agenda

- 1 Why Mars
 - GPGPU
 - MapReduce
- 2 How it works
 - Design
 - Implementation
- 3 Evaluation
 - Ease of use
 - High Performance
- 4 Conclusion

GPU Hardware Trend (1)

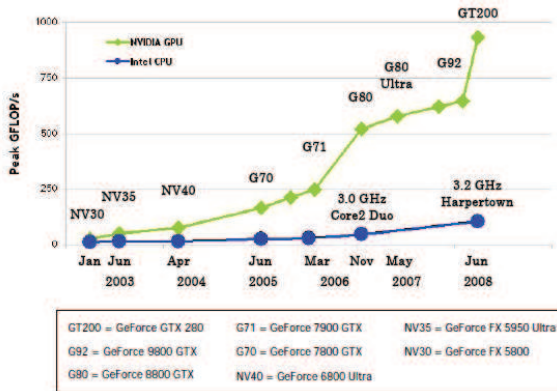


Figure: Floating-Point Operations per Second on NVIDIA GPUs and Intel CPUs.

Source: NVIDIA CUDA Programming Guide [4].

GPU Chip

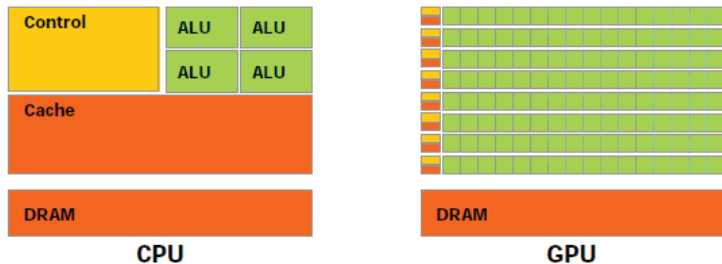


Figure: GPUs devote more transistors to data processing.

Source: NVIDIA CUDA Programming Guide [4].

GPU Hardware Trend (2)

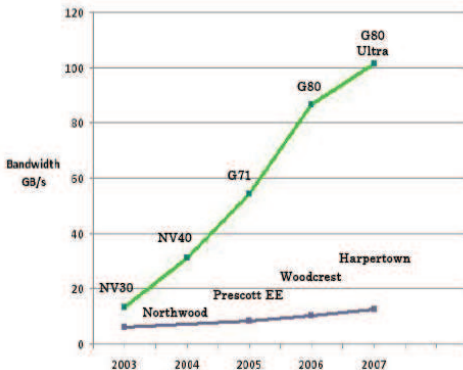
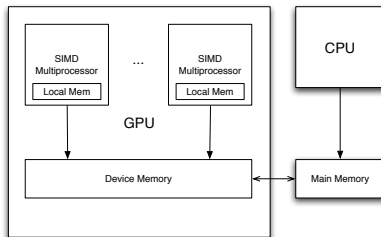


Figure: Bandwidth of NVIDIA GPU memory and CPU memory.

General Purpose GPU Computing

Many-core Arch for GPUs



Programability

- NVIDIA CUDA
- AMD Brook+
- OpenCL
- More...

Non-Graphics Workloads on GPUs

Owens et al. [5] **A Survey of General-Purpose Computation on Graphics Hardware**

- Linear algebra
- Finance
- Database query
- Machine Learning
- More...
- **Data Parallel** programs on SIMD multiprocessors.

Map Function and Reduce Function

Jeffrey Dean and Sanjay Ghemawat, **MapReduce: Simplified Data Processing on Large Clusters. OSDI'04.** [2]

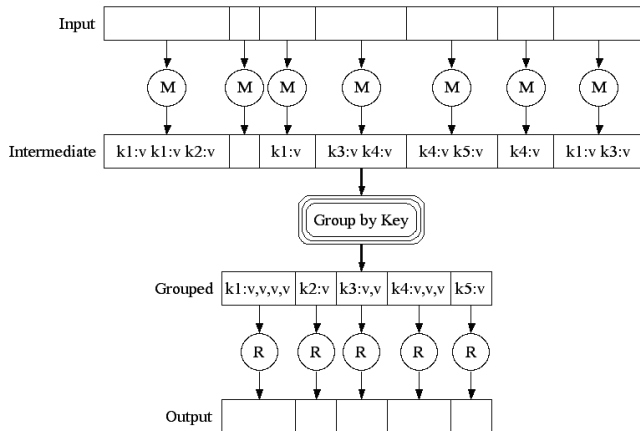
```

Map(void *doc) {
  1: for each word w in doc
  2:   EmitIntermediate(w, 1); // count each word once
}

Reduce(void *word, Iterator values) {
  1: int result = 0;
  2: for each v in values
  3:   result += v;
  4: Emit(word, result); // output word and its count
}

```

MapReduce Workflow



Source - <http://labs.google.com/papers/mapreduce-osdi04-slides/index-auto-0007.html>

Implementations of MapReduce

- Distributed Environment
 - Google MapReduce
 - Apache Hadoop (Yahoo, Facebook, ...)
 - MySpace Qizmt
- Multicore CPU
 - Phoenix from Stanford, HPCA'07 [6]/IISWC'09 [7].
- Cell BE
- FPGA
- GPUs
 - From UC-Berkeley, STMCS'08 [1]
 - Merge, from Intel, ASPLOS'08 [3]

Agenda

- 1 Why Mars
 - GPGPU
 - MapReduce
- 2 How it works
 - Design
 - Implementation
- 3 Evaluation
 - Ease of use
 - High Performance
- 4 Conclusion

Goals and Challenges

Design Goals

- Programmability. Ease of use.
- Flexibility. Support various multi/many core processors.
- High Performance.

Challenges

Result output.

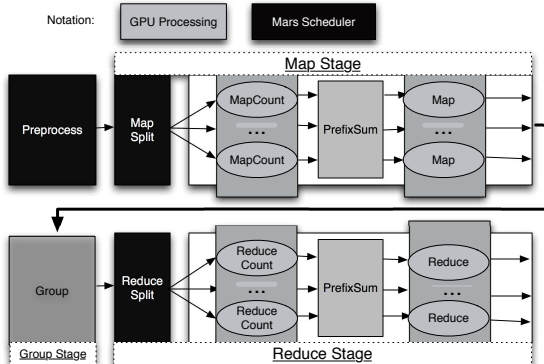
- Write conflicts among GPU threads.
- Unknown output buffer size.

Solution

Lock-free scheme

Workflow

Workflow of Mars



Customizing Workflow

- Map Only.
- Map→Group.
- Map→Group
→Reduce.
- ~~Group→Reduce.~~
- ~~Group.~~
- ~~Map→Reduce.~~

Data Structure

Records

Input Records →

Map Stage → Intermediate Records I → **Group Stage** →
Intermediate Records II → **Reduce Stage**
→ Output Records

Structure of Arrays

- Key array
- Value array
- Directory index array – Variable-sized record
 - <Key size, Key offset, Value size, Value offset>
- Chained MapReduce:
Map1 → Group1 → Map2 → Map3 → Map4 → Group4

Lock-Free Output

Lock Free

- MapCount
 - Call User defined MapCount function
 - Each function emits intermediate key size and value size
- Prefix sum on intermediate key sizes and value sizes
 - The size of intermediate buffer, allocate at one time
 - The deterministic write position for each Map, lock-free
- Allocate intermediate buffer
- Map
 - Call User defined Map function
 - Output records according to the write position

Lock-Free Output, Example

Map1 \rightarrow "123456789", Map2 \rightarrow "abcd", Map3 \rightarrow "ABCDEDED"

MapCount

- MapCount1 \rightarrow 9
- MapCount2 \rightarrow 4
- MapCount3 \rightarrow 6

Prefix Sum, Allocate buffer, and Map

- 9, 4, 6 – size array
- 0, 9, 13 – write position array
- 19 – output buffer size
- Allocate a buffer of size 19
- "123456789abcdABCDED"

MarsCUDA

Building blocks

- NVIDIA CUDA
- Prefix Sum: CUDPP Library, GPU-based Prefix Sum
- Group: GPU-based Bitonic Sort

MarsCUDA – Memory Optimization (1)

Coalesced Access

For a half-warp of threads, simultaneous device memory accesses to consecutive device memory addresses can be coalesced into one transaction. → Reduce # of device memory accesses.

Local memory

- Programmable on-chip memory (shared memory in NVIDIA's term).
- Exploit local memory in GPU-based Bitonic Sort for Group Stage.
- Users can explicitly utilize local memory in their Map/Reduce functions.

MarsCUDA – Memory Optimization (2)

Built-in Vector type

- Address Alignment
- float4 and int4
- One load instruction to read data of built-in type, of size up to 16 bytes → Reduce # of memory load instructions, compared with reading scalar type

Page-lock host memory

Prevent OS from paging the locked memory buffer → High PCI-E bandwidth

MarsCUDA – Task distribution

Map/Reduce

$\lceil N/B \rceil$ thread blocks

- N : the number of Map or Reduce tasks
- B : the number of GPU threads per thread block, which is practically set to 256
- 1 task per GPU thread

Special case for Reduce

- Communicative and Associative. For example, Integer Addition.
- Parallel reduction for load balanced reduce task distribution.

MarsCPU

Building blocks

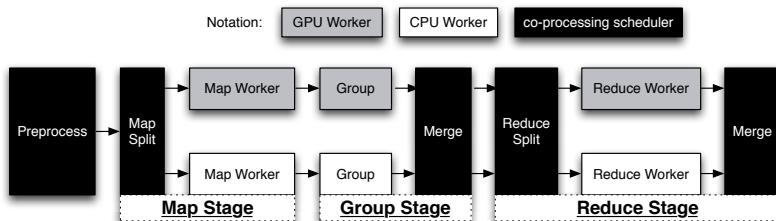
- pthreads
- Group: Parallel Merge Sort

General Mars Design

- Lock Free
- $\lceil N/T \rceil$ tasks per CPU thread.
 - N : the number of Map or Reduce tasks
 - T : the number of CPU threads
 - N is usually much larger than T

GPU/CPU Co-processing

The workflow of GPU/CPU co-processing



- I : Total size of input data
- S : Speedup of GPU Worker over CPU Worker
- Workload for GPU Worker: $\frac{SI}{1+S}$
- Workload for CPU Worker: $\frac{I}{1+S}$

MarsHadoop

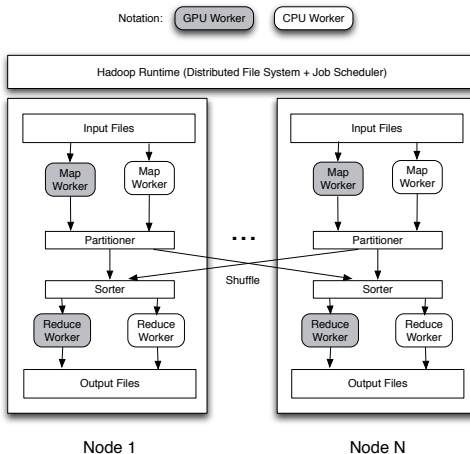


Figure: MarsHadoop. Using Hadoop Streaming.

Agenda

- 1 Why Mars
 - GPGPU
 - MapReduce
- 2 How it works
 - Design
 - Implementation
- 3 **Evaluation**
 - Ease of use
 - High Performance
- 4 Conclusion

Experimental Setup

Machine	PC A	PC B	PC C
GPU	NVIDIA GTX280	NVIDIA 8800GTX	ATI Radeon HD 3870
# GPU core	240	128	320
GPU Core Clock	602 MHz	575 MHz	775 MHz
GPU Memory Clock	1107 MHz	900 MHz	2250 MHz
GPU Memory Band-width	141.7 GB/s	86.4 GB/s	72.0 GB/s
GPU Memory size	1024 MB	768 MB	512 MB
CPU	Intel Core2 Quad Q6600	Intel Core2 Quad Q6600	Intel Pentium 4 540
CPU Clock	2400 MHz	2400 MHz	3200 MHz
# CPU core	4	4	2
CPU Memory size	2048 MB	2048 MB	1024 MB
OS	32-bit CentOS Linux	32-bit CentOS Linux	32-bit Windows XP

Applications

Applications	Small	Medium	Large
String Match (SM)	size: 55MB	size: 105MB	size: 160MB
Matrix Multiplication (MM)	256x256	512x512	1024x1024
Black-Scholes (BS)	# option: 1,000,000	# option: 3,000,000	# option: 5,000,000
Similarity Score (SS)	# feature: 128, # documents: 512	# feature: 128, # documents: 1024	# feature: 128, # documents: 2048
PCA	1000x256	2000x256	4000x256
Monte Carlo (MC)	# option: 500, # samples per option: 500	# option: 500, # samples per option: 2500	# option: 500, # samples per option: 5000

GPU Implementation: MarsCUDA, CUDA

CPU Implementation: MarsCPU, Phoenix, pthreads

GPUCPU Coprocessing: MarsCUDA + MarsCPU

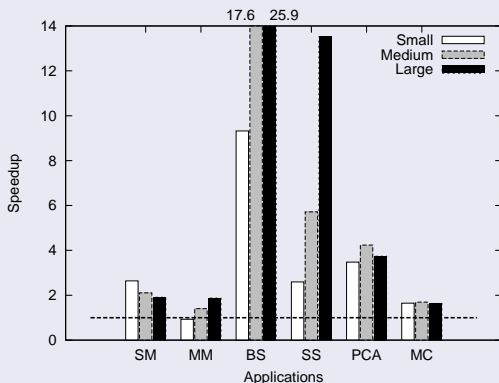
Code size saving

In lines:

Applications	Phoenix	MarsCUDA/MarsCPU	CUDA
String Match	206	147	157
Matrix Multiplication	178	72	68
Black-Scholes	199	147	721
Similarity Score	125	82	615
Principal component analysis	297	168	583
Monte Carlo	251	203	359

MarsCPU vs Phoenix

$$\text{Speedup} = T_{\text{Phoenix}} / T_{\text{MarsCPU}}$$

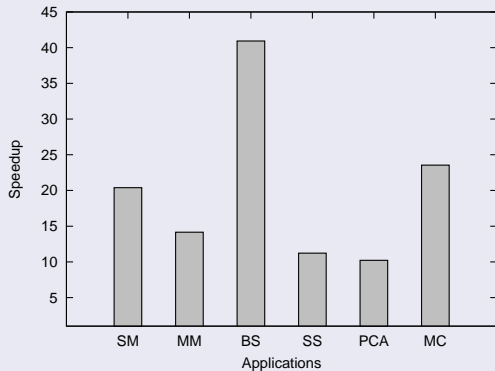


Overhead of Phoenix

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

MarsCUDA vs MarsCPU on Kernel

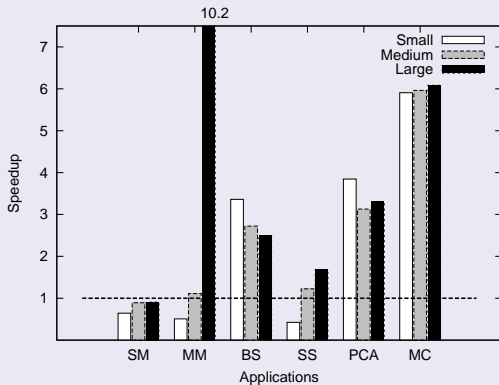
$$\text{Speedup} = T_{\text{MarsCPU}} / T_{\text{MarsCUDA}}$$



- Preprocess +
**Map + Group +
Reduce**

MarsCUDA vs MarsCPU

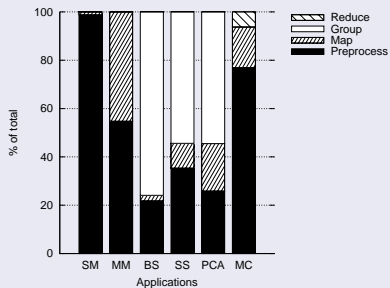
$$\text{Speedup} = T_{\text{MarsCPU}} / T_{\text{MarsCUDA}}$$



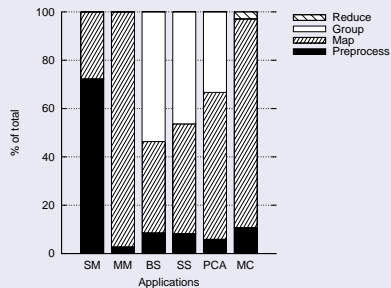
- Preprocess + Map
+ Group +
Reduce

Time Breakdown

MarsCUDA



MarsCPU



Amdahl's Law

Amdahl's Law

$$\text{Speedup} = \frac{1}{(1-P) + P/S}$$

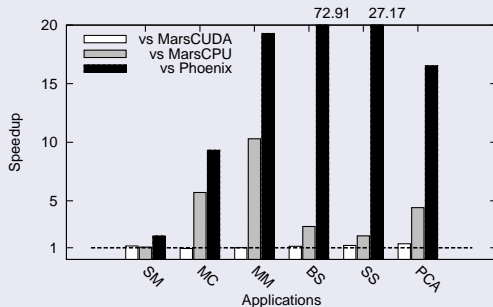
- P : The proportion that is parallelized
- $(1 - P)$: The proportion that is not parallelized
- S : Speedup by parallelism

For MarsCUDA

- P : Map + Reduce
- $(1 - P)$: Preprocess
- Example: String Match
 - Parallelized: Map stage
 - $P = 25\%$
 - $S = 20$
 - Speedup
$$= \frac{1}{(1-25\%) + 25\%/20} = 1.3$$

GPU/CPU Co-processing

$$\text{Speedup} = T_{\text{Standalone}} / T_{\text{Coproprocessing}}$$

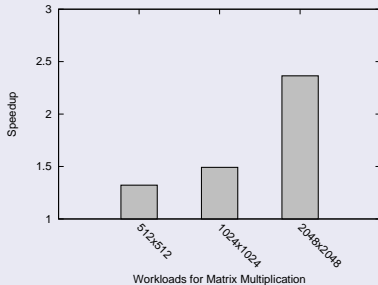


Co-processing over MarsCUDA:

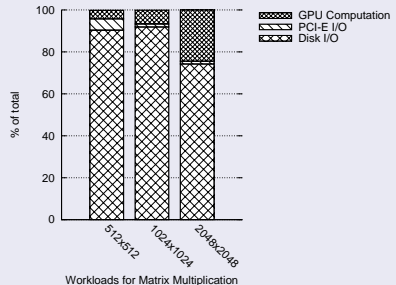
- $\text{Speedup} = \frac{S+1}{S}$
- S : Speedup of MarsCUDA over MarsCPU

MarsHadoop

$$\text{Speedup} = T_{Hadoop} / T_{MarsHadoop}$$



Time Breakdown



Two slave nodes: PC A and PC B
One master node: PC A

Reference



Marc de Kruijf and Karthikeyan Sankaralingam.
Mapreduce for the cell b.e. architecture.
[Technical report, University of WisconsinMadison, 2007.](#)



Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.
[OSDI, 2004.](#)



Michael D. Linderman, Jamison D. Collins, Hong Wang, and Teresa H. Meng.
Merge: a programming model for heterogeneous multi-core systems.
[ASPLOS, 2008.](#)



NVIDIA corp.
[NVIDIA CUDA Programming Guide 2.0, 2008.](#)



John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell.
A survey of general-purpose computation on graphics hardware.
[Computer Graphics Forum, 2007.](#)



Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis.
Evaluating mapreduce for multi-core and multiprocessor systems.
[HPCA, 2007.](#)



Richard Yoo, Anthony Romano, and Christos Kozyrakis.
Phoenix rebirth: Scalable mapreduce on a numa system.
In [IISWC, 2009.](#)

Conclusion

- Mars
 - MarsCUDA for NVIDIA GPU
 - MarsBrook for AMD GPU
 - MarsCPU for multicore CPU
 - GPU/CPU Co-processing
 - MarsHadoop for clusters
- Ease of programming
- High performance

Thanks! Q&A?

<http://www.cse.ust.hk/gpuqp/Mars.html>