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.
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: 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
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
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
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
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].
Figures: 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.

Source: NVIDIA CUDA Programming Guide [4].
General Purpose GPU Computing

Many-core Arch for GPUs

- SIMD Multiprocessor
  - Local Mem
- CPU
- Device Memory
- Main Memory
General Purpose GPU Computing

Many-core Arch for GPUs

Programmability
- 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**

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

**Reduce**

```c
(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

Input

Intermediate

Grouped

Output

Implementations of MapReduce

- Distributed Environment
  - Google MapReduce
  - Apache Hadoop (Yahoo, Facebook, ...)
  - MySpace Qizmt
Implementations of MapReduce

- **Distributed Environment**
  - Google MapReduce
  - Apache Hadoop (Yahoo, Facebook, ...)
  - MySpace Qizmt

- **Multicore CPU**
  - Phoenix from Stanford, HPCA’07 [6]/IISWC’09 [7].
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
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.
## 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.
## 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 of Mars

Notation:
- GPU Processing
- Mars Scheduler

Map Stage:
- Preprocess
- Map Split
- MapCount
- PrefixSum
- Map
- Reduce Count
- PrefixSum
- Reduce

Reduce Stage:
- Reduce Count
- PrefixSum
- Reduce

Group Stage:
- Group
- Split
- Reduce Count
- Reduce
Workflow of Mars

Preprocess → Map Split → Map → PrefixSum → Map → PrefixSum → Group → Reduce Split → Reduce → PrefixSum → Reduce → PrefixSum → Reduce → Reduce Stage → Group Stage

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
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>`
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
Lock-Free Output

- 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
Lock-Free Output

- **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
## 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 $\rightarrow$ "ABCDDED"
Lock-Free Output, Example

Map1 $\rightarrow$ "123456789", Map2 $\rightarrow$ "abcd", Map3 $\rightarrow$ "ABCDED"

<table>
<thead>
<tr>
<th>MapCount</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapCount1</td>
<td>9</td>
</tr>
<tr>
<td>MapCount2</td>
<td>4</td>
</tr>
<tr>
<td>MapCount3</td>
<td>6</td>
</tr>
</tbody>
</table>
Lock-Free Output, Example

Map1 → "123456789", Map2 → "abcd", Map3 → "ABCDED"

MapCount
- MapCount1 → 9
- MapCount2 → 4
- MapCount3 → 6

Prefix Sum, Allocate buffer, and Map
- 9, 4, 6 – size array
Lock-Free Output, Example

Map1 → "123456789", Map2 → "abcd", Map3 → "ABCDED"

### MapCount
- MapCount1 → 9
- MapCount2 → 4
- MapCount3 → 6

### Prefix Sum, Allocate buffer, and Map
- 9, 4, 6 – size array
- 0, 9, 13 – write position array
- 19 – output buffer size
Lock-Free Output, Example

Map1 → "123456789", Map2 → "abcd", Map3 → "ABCDEDED"

MapCount

- MapCount1 → 9
- MapCount2 → 4
- MapCount3 → 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
Lock-Free Output, Example

Map1 → "123456789", Map2 → "abcd", Map3 → "ABCDED"

<table>
<thead>
<tr>
<th>MapCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapCount1 → 9</td>
</tr>
<tr>
<td>MapCount2 → 4</td>
</tr>
<tr>
<td>MapCount3 → 6</td>
</tr>
</tbody>
</table>

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
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.
**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
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

\[ \left\lfloor \frac{N}{B} \right\rfloor \] 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
MarsCUDA – Task distribution

Map/Reduce

\[\left\lfloor \frac{N}{B} \right\rfloor\] 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.
Building blocks

- pthreads
- Group: Parallel Merge Sort
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

Notation:
- **GPU Worker**
- **CPU Worker**
- **co-processing scheduler**

Map Stage:
- Preprocess ➔ Map Split ➔ Map Worker ➔ Group ➔ Merge

Group Stage:
- Merge ➔ Reduce Split ➔ Reduce Worker ➔ Merge

Reduce Stage:
- Reduce Worker ➔ Merge

- \( I \): 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

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Match (SM)</td>
<td>size: 55MB</td>
<td>size: 105MB</td>
<td>size: 160MB</td>
</tr>
<tr>
<td>Matrix Multiplication (MM)</td>
<td>256x256</td>
<td>512x512</td>
<td>1024x1024</td>
</tr>
<tr>
<td>Black-Scholes (BS)</td>
<td># option: 1,000,000</td>
<td># option: 3,000,000</td>
<td># option: 5,000,000</td>
</tr>
<tr>
<td>PCA</td>
<td>1000x256</td>
<td>2000x256</td>
<td>4000x256</td>
</tr>
<tr>
<td>Monte Carlo (MC)</td>
<td># option: 500, # samples per option: 500</td>
<td># option: 500, # samples per option: 2500</td>
<td># option: 500, # samples per option: 5000</td>
</tr>
</tbody>
</table>

**GPU Implementation:** MarsCUDA, CUDA  
**CPU Implementation:** MarsCPU, Phoenix, pthreads  
**GPUCPU Coprocessing:** MarsCUDA + MarsCPU
## Code size saving

### In lines:

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

Speedup $= \frac{T_{Phoenix}}{T_{MarsCPU}}$

Overhead of Phoenix

Applications

SM  MM  BS  SS  PCA  MC

Speedup

Overhead

0  2  4  6  8  10  12  14

Small

Medium

Large

17.6  25.9
MarsCPU vs Phoenix

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

### Overhead of Phoenix
- Always need
- Reduce stage.
MarsCPU vs Phoenix

**Ease of use**

**High Performance**

**Why Mars**

**How it works**

**Evaluation**

**Conclusion**

**Speedup**

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

- **Small**
- **Medium**
- **Large**

**Applications**

- SM
- MM
- BS
- SS
- PCA
- MC

**Overhead of Phoenix**

- Always need
- Reduce stage.
- Lock overhead.
MarsCPU vs Phoenix

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

- Always need Reduce stage.
- Lock overhead.
- Re-allocate buffer on the fly.
MarsCPU vs Phoenix

**Speedup** = \( \frac{T_{Phoenix}}{T_{MarsCPU}} \)

### 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

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

- Preprocess + Map + Group + Reduce
MarsCUDA vs MarsCPU

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

- Preprocess + Map
- + Group + Reduce
Time Breakdown

MarsCUDA

MarsCPU

Applications

% of total

SM MM BS SS PCA MC

SM MM BS SS PCA MC

Reduce Group Map Preprocess

Reduce Group Map Preprocess
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
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
    \[
    = \frac{1}{(1-25\%)+25\%/20} = 1.3
    \]
Preprocess is a bottleneck?

Real world applications in Chained MapReduce:
Preprocess → Map1 → Group1 → Reduce1 → Map2 → Map3 → Map4 → Group4

- Prepare key/value pairs
- Transfer input key/value pairs from main memory to device memory
Co-processing over MarsCUDA:

- Speedup = \( \frac{S+1}{S} \)
- \( S \): Speedup of MarsCUDA over MarsCPU
MarsHadoop

Speedup $= \frac{T_{Hadoop}}{T_{MarsHadoop}}$

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

Workloads for Matrix Multiplication

<table>
<thead>
<tr>
<th>Workload</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x512</td>
<td>1.16</td>
</tr>
<tr>
<td>1024x1024</td>
<td>1.39</td>
</tr>
<tr>
<td>2048x2048</td>
<td>2.97</td>
</tr>
</tbody>
</table>

Time Breakdown

<table>
<thead>
<tr>
<th>Workload</th>
<th>GPU Computation</th>
<th>PCI-E I/O</th>
<th>Disk I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x512</td>
<td>25%</td>
<td>60%</td>
<td>15%</td>
</tr>
<tr>
<td>1024x1024</td>
<td>30%</td>
<td>50%</td>
<td>20%</td>
</tr>
<tr>
<td>2048x2048</td>
<td>35%</td>
<td>45%</td>
<td>20%</td>
</tr>
</tbody>
</table>
Reference

Marc de Kruijf and Karthikeyan Sankaralingam.
Mapreduce for the cell b.e. architecture.

Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.

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.

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.

Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis.
evaluating mapreduce for multi-core and multiprocessor systems.

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
Backup 1: GPU Workload and design trade-off

Why GPUs Have High Memory bandwidth?

Memory Bandwidth $\propto (\text{Clock Rate} \times \text{Memory Bus width})$
<table>
<thead>
<tr>
<th>Why GPUs Have High Memory bandwidth?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth $\propto (\text{Clock Rate} \times \text{Memory Bus width})$</td>
</tr>
</tbody>
</table>

| Why such design? |
Why GPUs Have High Memory bandwidth?

Memory Bandwidth $\propto (\text{Clock Rate} \times \text{Memory Bus width})$

Why such design?

- CPU: Use Cache to improve memory performance.
Why GPUs Have High Memory bandwidth?

Memory Bandwidth ∝ (Clock Rate × Memory Bus width)

Why such design?

- CPU: Use Cache to improve memory performance.
- GPU Workload: 3D rendering, large dataset of polygons and textures, too large working set to fit in cache.
Why GPUs Have High Memory bandwidth?

Memory Bandwidth \( \propto (\text{Clock Rate} \times \text{Memory Bus width}) \)

Why such design?

- CPU: Use Cache to improve memory performance.
- GPU Workload: 3D rendering, large dataset of polygons and textures, too large working set to fit in cache.
- GPU: the only way – wider memory bus + faster clock rate
Backup 1: GPU Workload and design trade-off

Why GPUs Have High Memory bandwidth?

Memory Bandwidth $\propto (\text{Clock Rate} \times \text{Memory Bus width})$

Why such design?

- CPU: Use Cache to improve memory performance.
- GPU Workload: 3D rendering, large dataset of polygons and textures, too large working set to fit in cache.
- GPU: the only way – wider memory bus + faster clock rate
- Price( NVIDIA GTX 285 GPU with 1 GB memory ) $\approx$ Price (Intel Core i7 CPU with 6 GB memory ).
Backup 2: Performance Slowdown Over Native Implementations

**MarsCUDA vs CUDA. Slowdown**

\[ \frac{T_{MarsCUDA}}{T_{CUDA}} \]

<table>
<thead>
<tr>
<th>SM</th>
<th>MM</th>
<th>BS</th>
<th>SS</th>
<th>PCA</th>
<th>MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MarsCPU vs pthreads. Slowdown**

\[ \frac{T_{MarsCPU}}{T_{pthread}} \]

<table>
<thead>
<tr>
<th>SM</th>
<th>MM</th>
<th>BS</th>
<th>SS</th>
<th>PCA</th>
<th>MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Applications: SM, MM, BS, SS, PCA, MC