Comparing CPU and GPU in OLAP Cube Creation
SOFSEM 2011

Krzysztof Kaczmarski

Faculty of Mathematics and Information Science
Warsaw University of Technology

22-28 January 2011
Introduction

Comparison of CPU and GPU

Summary
Preliminaries
General Purpose Computation on Graphics Processing Unit – GPGPU

Owens et al. (“A Survey of General-Purpose Computation on Graphics Hardware”)
GPGPU – General Architecture
GPGPU – General Architecture

NVIDIA (CUDA whitepapers)
Can we utilize this power in databases?
Known papers in Databases

- „Scaling PostgreSQL Using CUDA” Todd Hoff
- „Accelerating SQL Database Operations on a GPU with CUDA” Peter Bakkum and Kevin Skadron
- „Fast Computation of Database Operations using Graphics Processors” Naga K. Govindaraju Brandon Lloyd, Wei Wang, Ming Lin, Dinesh Manocha
- „Efficient Relational Database Management using Graphics Processors” Naga K. Govindaraju, Dinesh Manocha
- „Exploring Utilisation of GPU for Database Applications”, Slawomir Walkowiak, Konrad Wawruch, Marita Nowotka, Lukasz Ligowski, Witold Rudnicki
Known papers in Databases

- "Scaling PostgreSQL Using CUDA" Todd Hoff
- "Accelerating SQL Database Operations on a GPU with CUDA" Peter Bakkum and Kevin Skadron
- "Fast Computation of Database Operations using Graphics Processors" Naga K. Govindaraju Brandon Lloyd, Wei Wang, Ming Lin, Dinesh Manocha
- "Efficient Relational Database Management using Graphics Processors" Naga K. Govindaraju, Dinesh Manocha

As far as I know this is the first publication of GPGPU in OLAP.
OLAP (On-Line Analytical Processing) is a popular, industry-accepted technology, that takes advantage of pre-summing in order to provide fast and efficient access to analysis and reports on large data volumes.
Introduction

Comparison of CPU and GPU

Summary

Experiment set up

Database:

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
<th>Day</th>
<th>Group</th>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td>1</td>
<td>69</td>
</tr>
<tr>
<td>2009</td>
<td>12</td>
<td>30</td>
<td>1</td>
<td>4</td>
<td>79</td>
</tr>
<tr>
<td>2009</td>
<td>12</td>
<td>31</td>
<td>3</td>
<td>6</td>
<td>97</td>
</tr>
<tr>
<td>2009</td>
<td>12</td>
<td>31</td>
<td>1</td>
<td>4</td>
<td>79</td>
</tr>
<tr>
<td>2010</td>
<td>01</td>
<td>11</td>
<td>3</td>
<td>6</td>
<td>97</td>
</tr>
<tr>
<td>2009</td>
<td>01</td>
<td>12</td>
<td>1</td>
<td>4</td>
<td>79</td>
</tr>
<tr>
<td>2009</td>
<td>01</td>
<td>12</td>
<td>3</td>
<td>6</td>
<td>97</td>
</tr>
</tbody>
</table>

- 2 dimensions:
  - time with 3 levels (year, month, day)
  - 2 levels in the product dimension (group, product)
- The measure used was a sum of sales amount.
Outline

1. Introduction

2. Comparison of CPU and GPU
   - Programming ease
   - Computation times
   - Scalability

3. Summary
Comparison I – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
Comparison I – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
- GPU – complicated and depending on low level techniques:
Comparison I – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
- GPU – complicated and depending on low level techniques:
  - coalesced memory read and write
Comparison 1 – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
- GPU – complicated and depending on low level techniques:
  - coalesced memory read and write
  - shared memory utilization (bank conflicts)
Comparison I – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
- GPU – complicated and depending on low level techniques:
  - coalesced memory read and write
  - shared memory utilization (bank conflicts)
Comparison I – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
- GPU – complicated and depending on low level techniques:
  - coalesced memory read and write
  - shared memory utilization (bank conflicts)
- CPU coding time < one day
Comparison 1 – Programming ease

- CPU – very simple implementation – most of optimizations done by compiler and hardware
- GPU – complicated and depending on low level techniques:
  - coalesced memory read and write
  - shared memory utilization (bank conflicts)
- CPU coding time < one day
- GPU coding time > one week
## Comparison II – Computation time

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intell’s 3Ghz CPU Core2 Duo, 4GB of RAM</td>
<td>NVIDIA GTX 295 (1.242 GHz, 240 cores, 896MB)</td>
</tr>
</tbody>
</table>

1. Simply iterate over all records

1. Copy records to the device
2. Find ranges
3. Calculate aggregations
4. Copy results back

Input: 25M records (filled available devices memory completely)
Output: Cube of 1.424,016 aggregations

Including copying, GPU is approximately 2 times slower.
## Comparison II – Computation time

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel’s 3Ghz CPU Core2 Duo, 4GB of RAM</td>
<td>NVIDIA GTX 295 (1.242 GHz, 240 cores, 896MB)</td>
</tr>
</tbody>
</table>

1. Simply iterate over all records
2. Copy records to the device
3. Find ranges
4. Calculate aggregations
5. Copy results back

Input: 25M records (filled available devices memory completely)
Output: Cube of 1,424,016 aggregations
### Comparison II – Computation time

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intell’s 3Ghz CPU Core2 Duo, 4GB of RAM</td>
<td>NVIDIA GTX 295 (1.242 GHz, 240 cores, 896MB)</td>
</tr>
</tbody>
</table>

1. Simply iterate over all records
2. Copy records to the device
3. Find ranges
4. Calculate aggregations
5. Copy results back

Input: 25M records (filled available devices memory completely)
Output: Cube of 1.424.016 aggregations
Including copying, GPU is approximately 2 times slower.
Comparison III – Pure computation time (no copying included)

Input: 25M records (filled available devices memory completely)
Output: Cube of 1.424.016 aggregations
Comparison III – Pure computation time (no copying included)

Input: 25M records (filled available devices memory completely)  
Output: Cube of 1.424.016 aggregations

Depending on optimizations GPU from 10 to 18 times faster.
Comparison IV – Scalability I

Input: 25M records (filled available devices memory completely)
Output: Cube of 1,424,016 aggregations

- CPU: Tests for 1, 2 and 4 threads
- GPU: Tests for 1, 2 devices (GTX 295 dual-device)
Comparison IV – Scalability I

Input: 25M records (filled available devices memory completely)
Output: Cube of 1,424,016 aggregations

- CPU: Tests for 1, 2 and 4 threads
- GPU: Tests for 1, 2 devices (GTX 295 dual-device)
Comparison IV – Scalability I

Input: 25M records (filled available devices memory completely)
Output: Cube of 1,424,016 aggregations

- CPU: Tests for 1, 2 and 4 threads
- GPU: Tests for 1, 2 devices (GTX 295 dual-device)
Comparison IV – Scalability II
Comparison IV – Scalability II

Why performance of two devices is worse than one device?
Comparison IV – Scalability II

Why performance of two devices is worse than one device?
Memory copying over PCI-Express may achieve max 3GB/s (x16)
Memory copying over PCI-Express may achieve max 3GB/s (x16)
- Double devices cards degrade this to x8 performance.
- PCI cannot send data concurrently to two devices in one slot.
Outline

1 Introduction

2 Comparison of CPU and GPU

3 Summary
Conclusions – Massively Parallel Algorithm

Performance:

- Much faster than CPU counterpart
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)
Conclusions – Massively Parallel Algorithm

Performance:

- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)

Implementation:
- Inflexible, data dependent code, hard to debug
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)

Implementation:
- Inflexible, data dependent code, hard to debug
- Difficult and time consuming optimizations
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)

Implementation:
- Inflexible, data dependent code, hard to debug
- Difficult and time consuming optimizations
- Low level dependency
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)

Implementation:
- Inflexible, data dependent code, hard to debug
- Difficult and time consuming optimizations
- Low level dependency
- This technology is still under heavy development:
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)

Implementation:
- Inflexible, data dependent code, hard to debug
- Difficult and time consuming optimizations
- Low level dependency
- This technology is still under heavy development:
  - No guaranteed industrial standards (OpenCL?)
Conclusions – Massively Parallel Algorithm

Performance:
- Much faster than CPU counterpart
- Excellent scalability for better devices
  (binary code automatically scales to all available cores)

Implementation:
- Inflexible, data dependent code, hard to debug
- Difficult and time consuming optimizations
- Low level dependency
- This technology is still under heavy development:
  - No guaranteed industrial standards (OpenCL?)
  - Unknown directions in future
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side
  (time costly copying reduced to minimum)
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side
  (time costly copying reduced to minimum)
- Requires data stored in appropriate form
  (columns not rows)
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side (time costly copying reduced to minimum)
- Requires data stored in appropriate form (columns not rows)
- Parallelism needs new structures and algorithms (often not possible in databases)
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side  
  (time costly copying reduced to minimum)
- Requires data stored in appropriate form  
  (columns not rows)
- Parallelism needs new structures and algorithms  
  (often not possible in databases)
- Database limited by number of devices and available memory
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side
  (time costly copying reduced to minimum)
- Requires data stored in appropriate form
  (columns not rows)
- Parallelism needs new structures and algorithms
  (often not possible in databases)
- Database limited by number of devices and available memory
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side
  (time costly copying reduced to minimum)
- Requires data stored in appropriate form
  (columns not rows)
- Parallelism needs new structures and algorithms
  (often not possible in databases)
- Database limited by number of devices and available memory

Benefits:

- Faster computations
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side
  (time costly copying reduced to minimum)
- Requires data stored in appropriate form
  (columns not rows)
- Parallelism needs new structures and algorithms
  (often not possible in databases)
- Database limited by number of devices and available memory

Benefits:

- Faster computations
- Good scalability (cheaper speed-up)
Conclusions – GPU in databases

Requirements:

- Requires data storage on GPU side
  (time costly copying reduced to minimum)
- Requires data stored in appropriate form
  (columns not rows)
- Parallelism needs new structures and algorithms
  (often not possible in databases)
- Database limited by number of devices and available memory

Benefits:

- Faster computations
- Good scalability (cheaper speed-up)
- Smaller power consumption
Thank you.