Exploiting Graphic Card Processor Technology to Accelerate Data Mining in SAP NetWeaver BIA

Christoph Weyerhaeuser, Tobias Mindnich, Franz Färber
SAP NW EIM TREX

Wolfgang Lehner
TU Dresden

Pisa, Italy, 2008-12-15

Outline

- SAP BIA Architecture – Big Picture
- Why Graphic Cards
- Scenario: Optimizing DBColScan
- Evaluation
- Summary and Conclusion
Overview

- Vertical Decomposition – TREX stores tables by column, not row
- Smart Compression – TREX uses dictionaries and integer coding
- Horizontal Partitioning – TREX partitions indexes for parallel lookup
**BI Accelerator Availability and Scalability**

Master 1 is active, masters 2 and 3 are backups: each master pings all masters to check availability.

Run as many blades as required for current index and query load.

For higher availability install one or more extra blades and configure them to serve as backups for the other blades.

**Motivation: Why Graphic Cards**

Begin of “Desktop parallel computing age”

- GPUs are the first widespread parallel architecture
- Commodity hardware → GPUs are cheap (~ 500 €)
- GPUs are everywhere (desktop, notebook, PDA, cell phone)

**Nvidia GeForce 8800 GTX**

- G80 GPU: Chip Core (1350 MHz)
- 16 Multiprocessors x 8 ALUs (Arithmetic and Logic Unit)
- 768 MB GDDR3 RAM (900 MHz)
- Memory Bus Width: 384 bit
- PCIe x16 Interface
- Acquisition Costs: ~ 500 €
Thread Model - Overview

- 128 ALUs
- Multiprocessor 1
  - ALU1 to ALU8
  - 8192 registers
- Multiprocessor 16
  - Warp size: 32
- in parallel
- Thread Block
  - max 512 threads
- in sequence
- Thread Grid: max 65535 blocks

Motivation by Example

Data Mining Query

- predict 20 values
- real SAP cust data
  - 1,093,470,000 fact entries
  - 80 parts distributed over 10 blades with 80 cores in total
Dictionary-based Compression Scheme

Attribute Table

<table>
<thead>
<tr>
<th>RowId</th>
<th>ValueId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Dictionary

<table>
<thead>
<tr>
<th>ValueId</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM</td>
</tr>
<tr>
<td>2</td>
<td>Microsoft</td>
</tr>
<tr>
<td>3</td>
<td>Google</td>
</tr>
<tr>
<td>4</td>
<td>Sun</td>
</tr>
<tr>
<td>5</td>
<td>Novell</td>
</tr>
</tbody>
</table>

4 x 8-bit values can be concatenated to one 32-bit integer

Basic Memory Scan – Memory Layout

Example: 15 bit
Input: vector of 64 bit values (concatenated 15 bit values)
Output: 32 bit integer

- Unrolled extraction loop to extract 64 values at once
- 15 x 64 bit input integer → 64 x 15 bit values

- Extracted values are compared (rangeFrom <= value < rangeTo)
- Index of hits (docids) are stored in ValueVector / BitVector
Value Extraction – 15bit case

- Rowid 1 → mask vector[0] with 0x7FFF
- Rowid 2 → shift vector[0] by 15 and mask with 0x7FFF
- Rowid 3 → shift vector[0] by 30 and mask with 0x7FFF
- Rowid 5 → shift vector[0] by 60; mask vector[1] with 0x7FF and shift by 4; concatenate both by OR

Value Extraction – 15bit case
→ 1st position

vector[0]:

Mask (0x7FFF):

0000.................................0000 111...111

42
Scan Configuration

Two directions of parameterization
- The number of threads per block which alters the number of usable registers per thread
- The size of data blocks (k*32 bit with k being the coding parameter) per thread

Experimental Setup
- 8-bit coding scheme
- 512MByte raw data

Observation
- Query runtimes do not show any correlation with the thread/block configuration.
Scan Configuration (cont.)

Two directions of parameterization
- The number of threads per block which alters the number of usable registers per thread
- the size of data blocks (k*32 bit with k being the coding parameter) per thread

Experiments
- thread factor values
  1, 8, 16, 32, 64, and 128
  for different raw data sizes.

Observation
- the more threads
  (with smaller data sets),
  the better the query runtime
  - surprising,
  extremely light-weight thread model

Overhead of non-aligned kernels

Runtime comparison of 8-bit and 15-bit scenario
- Retrieve same number of values (same number of value extraction operations)
  - memory size varies
- Selectivity was 1/255 over uniformly distributed data set

Observation
- query runtimes increase proportionally with the data volume
- 15-bit case requires between 19%-56% more time by processing 187.5% data volume

<table>
<thead>
<tr>
<th>no. of extracted values</th>
<th>raw data size</th>
<th>query runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8-bit case</td>
<td>15-bit case</td>
</tr>
<tr>
<td>67, 108, 864</td>
<td>64 MB</td>
<td>120 MB</td>
</tr>
<tr>
<td>134, 217, 728</td>
<td>128 MB</td>
<td>240 MB</td>
</tr>
<tr>
<td>268, 435, 456</td>
<td>256 MB</td>
<td>480 MB</td>
</tr>
</tbody>
</table>
### Overhead per Filter Criterion

<table>
<thead>
<tr>
<th>input lines</th>
<th>Bit-Vector (0/5 Hits)</th>
<th>Bit-Vector (1/5 Hits)</th>
<th>Bit-Vector (5/5 Hits)</th>
<th>Bit-Vector (0/5 Hits)</th>
<th>Bit-Vector (1/5 Hits)</th>
<th>Bit-Vector (5/5 Hits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.388.608</td>
<td>8.73</td>
<td>9.84</td>
<td>12.87</td>
<td>11.59</td>
<td>13.49</td>
<td>17.28</td>
</tr>
<tr>
<td>33.554.432</td>
<td>31.78</td>
<td>36.15</td>
<td>47.91</td>
<td>42.78</td>
<td>47.06</td>
<td>65.14</td>
</tr>
<tr>
<td>67.108.864</td>
<td>62.36</td>
<td>71.22</td>
<td>95.11</td>
<td>85.27</td>
<td>88.92</td>
<td>128.53</td>
</tr>
</tbody>
</table>

### Overhead per Hit

<table>
<thead>
<tr>
<th>input lines</th>
<th>Bit-Vector (0/1 Hits)</th>
<th>Bit-Vector (1/1 Hits)</th>
<th>Bit-Vector (5/5 Hits)</th>
<th>Bit-Vector (0/1 Hits)</th>
<th>Bit-Vector (1/1 Hits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.388.608</td>
<td>5.77</td>
<td>6.34</td>
<td>9.08</td>
<td>10.76</td>
<td></td>
</tr>
<tr>
<td>16.777.216</td>
<td>10.48</td>
<td>11.19</td>
<td>17.10</td>
<td>20.53</td>
<td></td>
</tr>
<tr>
<td>33.554.432</td>
<td>19.06</td>
<td>21.82</td>
<td>33.21</td>
<td>39.89</td>
<td></td>
</tr>
<tr>
<td>67.108.864</td>
<td>38.78</td>
<td>43.50</td>
<td>65.24</td>
<td>78.95</td>
<td></td>
</tr>
</tbody>
</table>
GPU versus CPU

Comparison with
- CPU (Intel Xeon X5355 with 2.67GHz and 1333 MHz Front Side Bus) used in productive SAP NetWeaver BIA environments (Clovertown)
  - Single core
  - 4 cores (partitioned data set)

Data Set
- uniformly distributed with 255 distinct values (8bit coding scheme)
- point query on 33,554,432 Bytes comprising 134,217,728 different values

<table>
<thead>
<tr>
<th>processor</th>
<th>time in msec</th>
<th>factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon X5355 (1 core)</td>
<td>452</td>
<td>4.96</td>
</tr>
<tr>
<td>Intel Xeon X5355 (4 cores)</td>
<td>118</td>
<td>1.29</td>
</tr>
<tr>
<td>Nvidia G80</td>
<td>91</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Comment
- Dual-Xeon (2x4cores) does not yield any noticeable performance gain because of higher communication costs between the CPUs

Data Transfer Mode

- Nvidia GeForce
- 8800 GTX graphics board with the G80 GPU (1350 MHz)
- PCIe x16 interface → 4GByte/s
- Pinned mode
  - No swapping of memory
- Paged mode
  - Might be swapped to disk

Transfer cost for 512MByte

<table>
<thead>
<tr>
<th>direction</th>
<th>transfer mode</th>
<th>transfer time</th>
<th>resulting bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>THEORY (8b/10b coded)</td>
<td>62.5 ms</td>
<td>4GByte/s</td>
</tr>
<tr>
<td>Host RAM → GC RAM</td>
<td>PINNED</td>
<td>196.3 ms</td>
<td>2.54GByte/s</td>
</tr>
<tr>
<td>Host RAM → GC RAM</td>
<td>PAGED</td>
<td>290.7 ms</td>
<td>1.71GByte/s</td>
</tr>
<tr>
<td>GC RAM → Host RAM</td>
<td>PINNED</td>
<td>246.8 ms</td>
<td>2.02GByte/s</td>
</tr>
<tr>
<td>GC RAM → Host RAM</td>
<td>PAGED</td>
<td>384.3 ms</td>
<td>1.3GByte/s</td>
</tr>
</tbody>
</table>
Energy Consumption

<table>
<thead>
<tr>
<th>Desktop</th>
<th>Idle</th>
<th>1 core</th>
<th>2 cores</th>
<th>Full workload</th>
<th>GPU</th>
<th>GPU + 2 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU + Nvidia:</td>
<td>146 W</td>
<td>170 W</td>
<td>184 W</td>
<td>265 W</td>
<td>275 W</td>
<td></td>
</tr>
<tr>
<td>CPU + ATI:</td>
<td>70 W</td>
<td>91 W</td>
<td>105 W</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Difference:</td>
<td>76 W</td>
<td>79 W</td>
<td>79 W</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

→ Nvidia board doubles energy consumption: +76 W (from 70 W to 146 W)
→ 1 core (full workload): +24 W (from 146 W to 170 W)
→ 2 cores (full workload): +38 W (from 146 W to 184 W)
→ GPU (full workload): +119 W (from 146 W to 265 W)

→ GPU temperature increase from 62°C (idle) to 81°C (full workload)

System: Core 2 Duo (2.13 GHz), 2 GB RAM, 160 GB SATA HD
Nvidia: Nvidia GeForce 8800GTX, 768 MB, ASUS EN8800GTX/HTDP, actively chilled
ATI: ATI Radeon 7000, 64 MB, Sapphire, passively chilled

Summary

Pros
- Performance improvement
  - Extremely lightweight scheduling mechanism
- Moderate development effort by using CUDA API
  - Hardware architecture of graphics board is well wrapped by CUDA
  - Not necessary to learn a new programming language

Cons
- Not yet commodity HW (blades)
- High energy consumption

Conclusion
- Hard to compete with full-scale exploitation of Intel SSE4.1 feature (CPU Xeon E5430 - Harpertown)