CS 764: Topics in Database Management Systems
Lecture 27: GPU Databases

Xiangyao Yu
12/7/2022
Announcements

DAWN workshop
  – Reserve a presentation slot using the following google sheet
    https://docs.google.com/spreadsheets/d/1Re1M9FmJwl_YkidhNgeV0iKn-clsFrK_J1PMidaAuw/edit?usp=sharing
  – 8-min per group (presentation + QA)

Project report (DDL: Dec. 19)
  – Submit to the hotcrp website (like the proposal)

Submit course evaluation on aefis.wisc.edu
System Architecture

![System Architecture Diagram](image)
Advantages of GPU for Data Analytics

Advantage 1: **High computational power**
- GPU has massive parallelism using SIMT model
Advantages of GPU for Data Analytics

Advantage 1: **High computational power**

Advantage 2: **Higher Memory Bandwidth**

– GPU memory bandwidth is one-order-of-magnitude higher than CPU memory bandwidth
Challenges of GPU for Data Analytics

Challenge 1: **Limited memory capacity**

– Some data sets do not fit in GPU memory
Challenges of GPU for Data Analytics

Challenge 1: **Limited memory capacity**

Challenge 2: **Limited interconnect bandwidth**

– Inter-device data transfer is a performance bottleneck
GPU Database Roadmap

- Data size
- Performance
- CPU memory capacity (100GB-10TB)
- CPU DB
Project 1: **Saturate GPU memory** [1]

- Saturate GPU memory bandwidth when data fits in GPU memory

---

Project 1: **Saturate GPU memory** [1]

Project 2: **Data compression** [2]

– More data can fit in GPU memory

---


GPU Database Roadmap

Project 1: **Saturate GPU memory** \(^1\)
Project 2: **Data compression** \(^2\)
Project 3: **Hybrid CPU-GPU DB** \(^3\)
    - Leverage both CPU and GPU computation

---

\(^1\) Anil Shanbhag, Samuel Madden, Xiangyao Yu, *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, SIGMOD 2020

\(^2\) Anil Shanbhag*, Bobbi Yogatama*, Xiangyao Yu, Samuel Madden, *Tile-based Lightweight Integer Compression in GPU*, SIGMOD 2022

\(^3\) Bobbi Yogatama, Weiwei Gong, Xiangyao Yu, *Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS*, VLDB 2022
Outline

Project 1: Crystal library for in-GPU data analytics
Project 2: Data compression
Project 3: Hybrid CPU-GPU DB
Outline

Project 1: Crystal library for in-GPU data analytics
Project 2: Data compression
Project 3: Hybrid CPU-GPU DB
Past work reported wide variety of gains from $2 \times$ to $1000 \times$

One would expect the maximum gain to be roughly equal to the ratio of the memory bandwidth of GPU to that of CPU.

**Key contribution**: developed Crystal library that allows GPU data analytics to saturate GPU memory bandwidth (V100, 880GB/s)
Shared memory (~100 KB) is local to every Streaming Multiprocessors (SM). Shared memory access is 10x faster than global memory.
Store intermediate result between operators in the shared memory (~10x faster).

(a) Conventional execution model
Key Idea: Tile-Based Execution Model

(a) Conventional execution model
Key Idea: Tile-Based Execution Model

(a) Conventional execution model
Key Idea: Tile-Based Execution Model

(a) Conventional execution model
Key Idea: Tile-Based Execution Model

Store intermediate result between operators in the shared memory (~10x faster)
Data is partitioned and processed in tiles (each tile must fit in shared memory)

(a) Conventional execution model
(b) Tile-based execution model
Key Idea: Tile-Based Execution Model

Store intermediate result between operators in the shared memory (~10x faster)
Data is partitioned and processed in tiles (each tile must fit in shared memory)

(a) Conventional execution model
(b) Tile-based execution model
Key Idea: Tile-Based Execution Model

Store intermediate result between operators in the shared memory (~10x faster)
Data is partitioned and processed in tiles (each tile must fit in shared memory)
Experimental Results

With Crystal, GPU is on average **25X** faster than CPU running Star-Schema Benchmark (SSB)

GPU database with frequent PCIe data transfer can underperform a highly optimized CPU database
Outline

Project 1: Crystal library for in-GPU data analytics

**Project 2: Data compression**

Project 3: Hybrid CPU-GPU DB
GPU Data Compression

Supported compression schemes: (1) frame-of-reference + bit-packing, (2) delta encoding, (3) run-length encoding

**Key challenge:** decompression should not be a performance bottleneck
Idea 1: **Tile-based decompression**

- Multiple decompression steps can be encapsulated into a single device function
- Decompression can be done inline with query execution

---

(a) Conventional decompression model

(b) Tile-based decompression model
GPU Data Compression — Key Ideas

Idea 2: Efficient bit-packing compression
- Compact data format
- Low-level performance optimizations

Data:

- Block Size
- Miniblock Count
- Total count
- Bitwidth Word
- (total count/block size) blocks

Block Starts:

Values:

- 100 101 101 102 101 101 102 101 99 100 105 107 114 112 110 105

Reference:

- 99

Miniblock 1:

- 1 2 2 3 2 2 3 2 0 1 6 8 15 13 11 6

Maxbits = 2

Encoded Block:

- 99 2 4 0110101110101101 00000001011010001111110110110110

Miniblock 2:

- 0110101101000000101000111111110110110110

Maxbits = 4
Evaluation – Star Schema Benchmark

(a) Compressed data size

- Our compression rate (**GPU-⋆**) is comparable to the best-previous scheme (i.e. nvCOMP).
Our compression rate ($\text{GPU-\texttt{*}}$) is comparable to the best-previous scheme (i.e. $\text{nvCOMP}$).

$\text{GPU-\texttt{*}}$ is $2.2x$ faster in decompression time than the best-previous scheme.
### Evaluation – Star Schema Benchmark

- Our compression rate (GPU-*) is comparable to the best-previous scheme (i.e. nvCOMP).
- GPU-* is **2.2x** faster in decompression time than the best-previous scheme.
- GPU-* is **2.6x** faster in query running time than the best-previous scheme.
- GPU-* executes queries with minimal performance degradation compared to no encoding.
Outline

Project 1: Crystal library for in-GPU data analytics
Project 2: Data compression

Project 3: Hybrid CPU-GPU DB
Challenges in heterogeneous CPU-GPU data analytics

- **Data placement**: hot data should be cached in GPU memory
- **Heterogeneous query execution**: hybrid execution with minimal inter-device data transfer
GPU data caching should be:

1. **Fine-grained**: segment instead of column
2. **Semantic-aware**: priority of caching depends on the query pattern
Fine-grained Semantic-Aware Caching

Algorithm 1: Update the weighted frequency counter for segment $S$

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>UpdateWeightedFreqCounter(segment S)</code></td>
</tr>
<tr>
<td>2</td>
<td># estimate query runtime when S is not cached.</td>
</tr>
<tr>
<td>3</td>
<td>$RT_{\text{uncached}} = \text{estimateQueryRuntime}(cached_segments \setminus S)$</td>
</tr>
<tr>
<td>4</td>
<td># estimate query runtime when S and segments correlated with S are cached.</td>
</tr>
<tr>
<td>5</td>
<td>$RT_{\text{cached}} = \text{estimateQueryRuntime}(cached_segments \cup S \cup correlated_segments)$</td>
</tr>
<tr>
<td>6</td>
<td><code>weight = RT_{\text{uncached}} - RT_{\text{cached}}</code></td>
</tr>
<tr>
<td>7</td>
<td><code>S.weighted_freq_counter += weight</code></td>
</tr>
<tr>
<td></td>
<td><code>for C in correlated\_segments do</code></td>
</tr>
<tr>
<td></td>
<td># evenly distribute weight to all segments correlated with S</td>
</tr>
<tr>
<td></td>
<td>C.weighted_freq_counter += weight /</td>
</tr>
</tbody>
</table>

Weighted LFU replacement

- Each segment is assigned a different weight (higher weight $\rightarrow$ higher priority)

The weight of a segment reflects:

1. The relative speedup of caching a segment.
2. Correlation among segments from different columns

`estimateQueryRuntime()` uses a model to predict runtime, assuming bandwidth as the bottleneck
Heterogeneous Query Execution

Challenge:
- Extra complexity of query execution due to only subset of data cached in GPU
- Query executor should fully exploit the data in GPU and coordinate query execution across two devices

Solution:
- Segment-level query execution
Segment level execution
  – Group segments with the same execution plan into segment groups
Heterogeneous Query Execution

**Segment level execution**

- Group segments with the same execution plan into **segment groups**
- Execute each segment group and merge the results

**SELECT** S.D, SUM(R.C) **FROM** R,S
WHERE R.B = S.D AND R.A > 10 AND S.E > 20
GROUP BY S.E
Heterogeneous Query Execution

Other optimizations

– **Late materialization**: express intermediate relation in the form of row ID to reduce the total data transfer

– **Operator pipelining**: pipelining consecutive operators on the same device whenever possible

– **Segment skipping**: apply minmax pruning both for predicate evaluation and join operator
Mordred Architecture

Three components:
- Cache Manager
- Query Optimizer
- Query Execution Engine
Evaluation — Caching Policies

Semantic-aware fine-grained caching achieves the best performance
Evaluation — SSB Runtime

Mordred (our design) is 6× faster than the best-previous design
– 8GB cache size, ~32GB data size
Conclusion

Future Directions

GPU database has great performance potential

Key challenge: large data sets do not fit in GPU memory

- Project 1: Saturate GPU memory [1]
- Project 2: Data compression [2]
- Project 3: Hybrid CPU-GPU DB [3]

---

[1] Anil Shanbhag, Samuel Madden, Xiangyao Yu, A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics, SIGMOD 2020
[3] Bobbi Yogatama, Weiwei Gong, Xiangyao Yu, Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS, VLDB 2022
Special hardware designed for SQL analytics?
Transfer between CPU and GPU becomes a bottleneck?
How popular are GPUs used in industrial databases? What are the main barriers?
Next Lecture

DAWN workshop

– Reserve a presentation slot using the following google sheet
  https://docs.google.com/spreadsheets/d/1Re1M9FmJwI_YkidhNgeV0iKn-clssFrK_J1PMidaAuw/edit?usp=sharing
  – 8-min per group (presentation + QA)

Project report (DDL: Dec. 19)

– Submit to the hotcrp website (like the proposal)

Submit course evaluation on aefis.wisc.edu