CS 839: Cloud Native Databases

# Mordred:
# *Heterogeneous CPU-GPU DBMS*

Harshit Sharma

Dec 22, 2023

# INTRODUCTION

# GPUs for Data Analytics

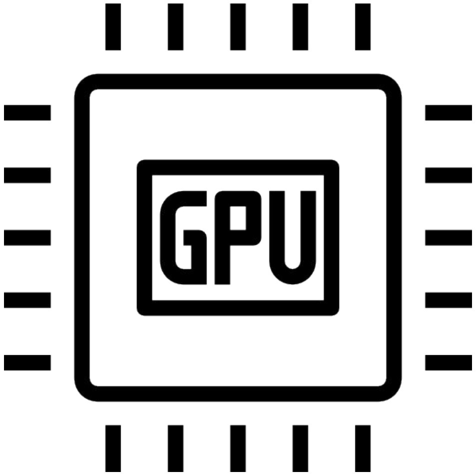
- Growing Interest in using GPUS for Accelerated Data Analytics
- Massive Parallelism and High Memory Bandwidth
- GPU databases have been studied in both academic research and developed as commercial products, demonstrating more than 10× speedup over the CPU counterparts.

+


= 10x

# What's the challenge then?

GPUs are expensive and offer limited capacity of GPU memory!!




Upto
~ 80 GBs

# Any Solutions?

**Solution I:**

1. Transfer data to GPU on demand through PCIe when a query accesses data that is not in GPU.
2. Pros: Straightforward Solution with commercial and academic implementations.
3. Cons: Potentially significant data traffic over PCIe, which can become a new performance bottleneck.

# Any Better Solutions?

**Solution II:**

1. Leverage both CPU and GPU for heterogeneous query processing.
2. Pros: Fully exploit the computational power of both devices and avoid excessive data transfer over PCIe by running certain sub-queries in CPU.
3. Cons: Trade-off with higher design complexity for data placement and heterogeneous query execution across devices.

# Solution II Focus:

**Data Placement:**

1

- How do we place data between GPU and CPU for a Heterogenous query executor.

2

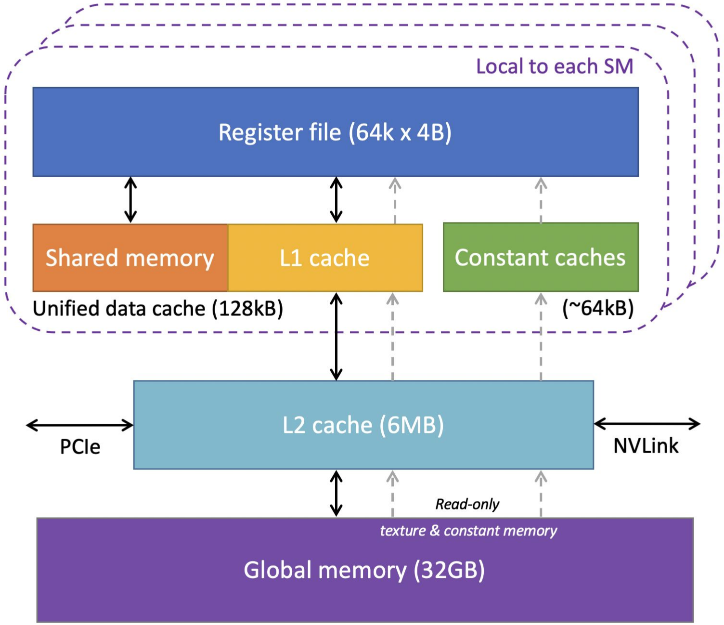
**Heterogeneous Query Executor:**

- How does the Heterogeneous query executor exploit data for a such devised data placement strategy?

# BACKGROUND

# GPU Architecture:

1. GPUs use streaming multiprocessors (SM) as the basic computing unit.
2. Each SM has fixed registers and shared memory (SMEM), with global memory accesses cached in L1 and L2 caches.
3. GPU programming organizes threads into blocks executed on an SM, with threads in a warp following the SIMT model for optimized memory access.



# GPU Data Analytics

GPU as the primary execution engine

Heterogeneous CPU-GPU query execution

GPU as a coprocessor

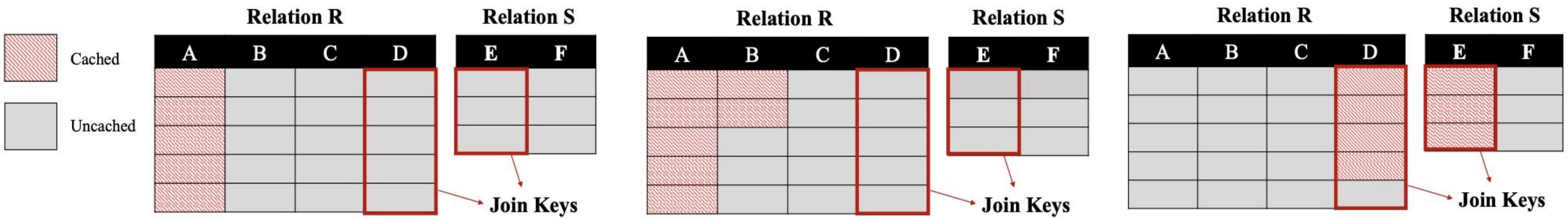
# Crystal Library

1. **Mordred utilizes Crystal, a CUDA library, employing a tile-based execution model for GPU analytic queries.**
2. **Crystal treats thread blocks as the basic units, processing 512-entry tiles in shared memory to minimize global memory round-trips.**
3. **Crystal enables pipelined execution for faster analytics query speeds.**
4. **Mordred adopts Crystal's cost model to estimate query runtimes for its replacement policy.**

# METHODOLOGY

# Data Placement:

1. Mordred treats data placement as a caching problem.
2. The complete data set resides in CPU memory and a mirrored subset of data is cached in GPU.
3. How to Cache: Follows a sub-column fine-grained policy.
4. What to Cache: Semantic-Aware Fine-Grained Caching.




(a) Coarse-grained caching (b) Fine-grained caching (c) Fine-grained semantic-aware caching



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

# Semantic-Aware Fine-Grained Caching:

1. **Extend conventional LFU with weighted frequency counters.**
2. **The weight reflects the potential benefits of caching the segment and is derived using a cost model.**
3. **The cost model captures:**
   a. **The relative speedup of caching a segment**
   b. **The correlation among segments from different columns.**

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

```
UpdateWeightedFreqCounter(segment S)
    # estimate query runtime when S is not cached.
    RT_uncached = estimateQueryRuntime(cached_segments \ S)
    # estimate query runtime when S and segments correlated with S
      are cached.
    RT_cached = estimateQueryRuntime(cached_segments ∪ S ∪
      correlated_segments)
    weight = RT_uncached − RT_cached
    S.weighted_freq_counter += weight
    for C in correlated_segments do
        # evenly distribute weight to all segments correlated with S
        C.weighted_freq_counter += weight / |correlated_segments|
```



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

# Heterogeneous Query Execution

1. Fine-grained caching policy == Extra complexity of query execution
2. Goals:
    a. Minimize inter-device data transfer
    b. Minimize CPU/GPU memory traffic
    c. Fully exploit parallelism in both CPU and GPU
3. Operator Placement:
    a. Data-driven operator placement heuristic applied at segment granularity
    b. A single operator can be split to run in both CPU and GPU depending on the location of input segments.
4. Resulting Plan as ***Segment-Level Query Plan***.

# Heterogeneous Query Execution

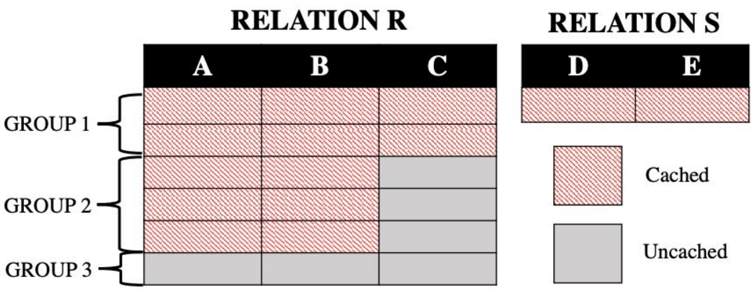



**Figure 2: Example of Segment Grouping.**

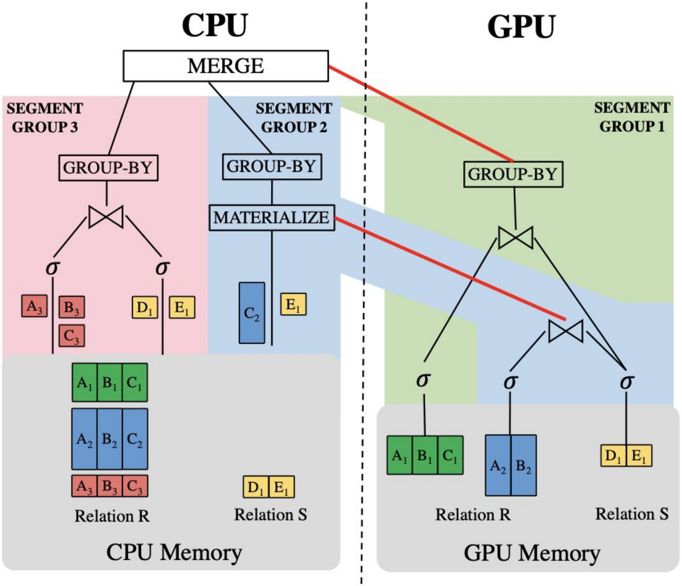



**Figure 3: Example of Segment-level Query Plan.**



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

# Other Optimizations

1. Late materialization
2. Operator Pipelining
3. Segment Skipping

# Overall System

1. The **Cache Manager** performs periodic data replacement in GPU memory based on the caching policy.
2. The **Query Optimizer** module converts a query plan into a segment-level query plan.
3. The **Query Execution Engine** executes segment-level query plan generated by the query optimizer.

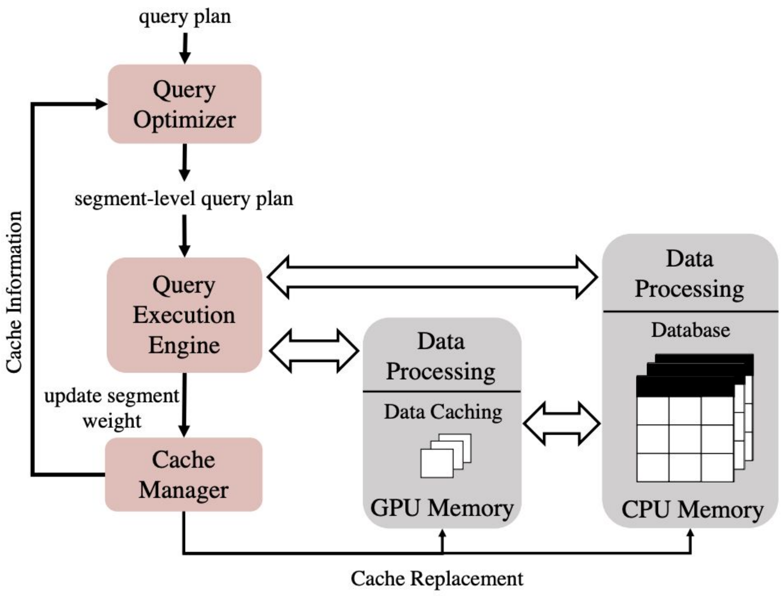



**Figure 4: Mordred System Overview**



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

# SYSTEM EVALUATION

# Semantic Caching Vs Others

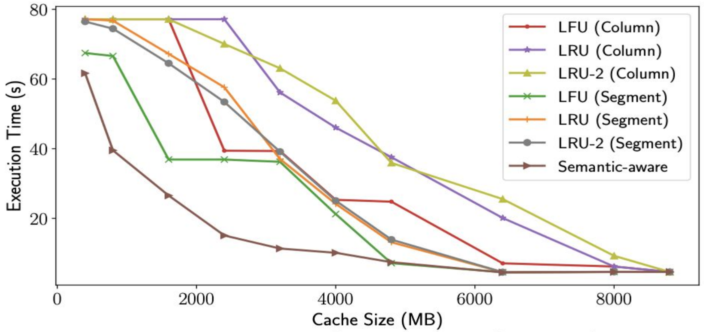



**Figure 5: Execution Time of Various Caching Policies with Different Cache Size (Uniform distribution with $\theta = 0$)**

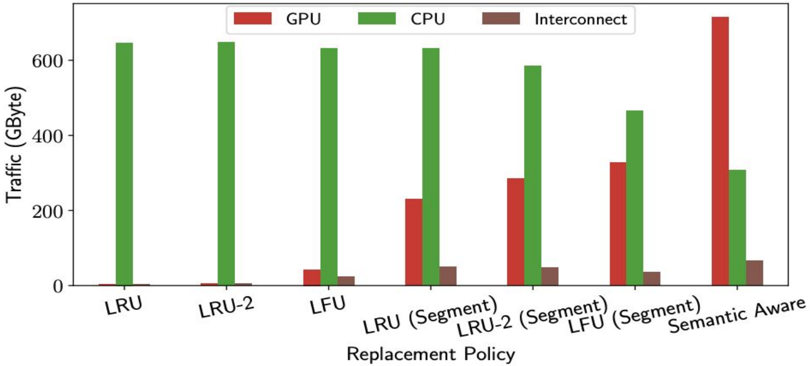



**Figure 6: Memory Traffic Breakdown for Each Caching Policy**



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

# Optimizations Evaluations

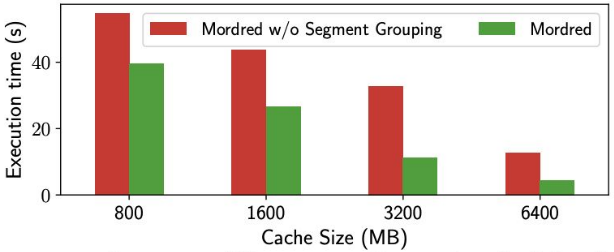



**Figure 10: Impact of Segment Grouping in Mordred**

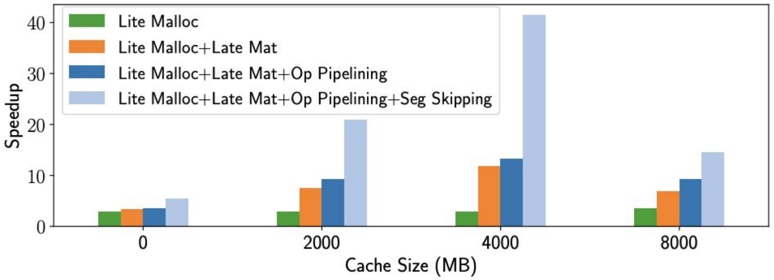



**Figure 11: Performance Speedup after Each Optimization**



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

# Mordred VS DBs

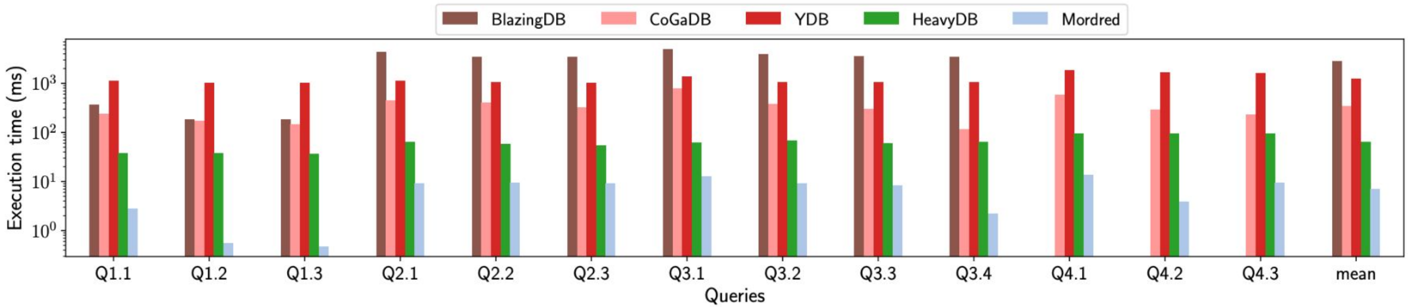



**Figure 13: SSB Query Performance of Different CPU/GPU DBMS (Data fits in GPU)**

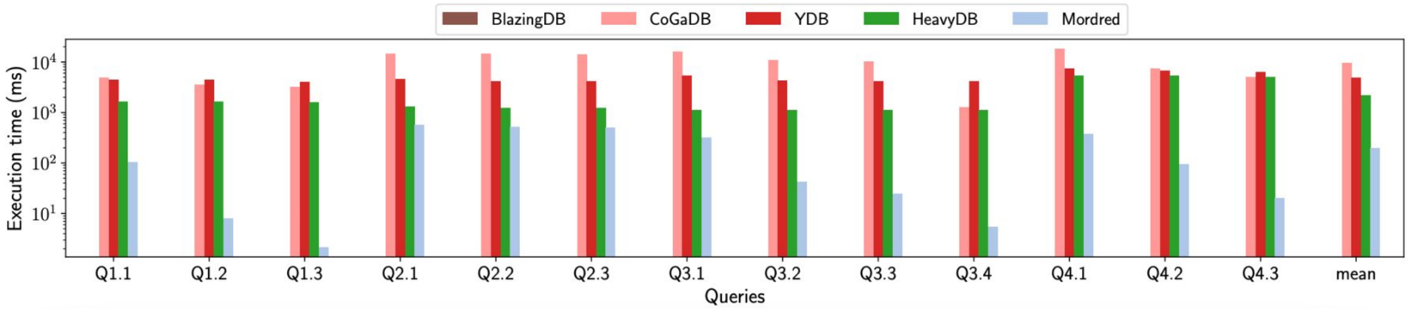



**Figure 14: SSB Query Performance of Different CPU/GPU DBMS (Data does not fit in GPU)**



Figures credit: Bobbi Yogatama, et al. Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS. VLDB 2022

CS 839: Cloud Native Databases

# THANK YOU