Revisiting Query Performance in GPU Database Systems

Dante Smith

Quick Aside on GPU Databases

Background on GPU as relates to Database Systems

- GPU = Graphics Processing Unit
- Initially designed to quickly render high resolution images and video
- However, GPUs are generally good for performing parallel operations on multiple sets of data
- Can be used for data analytics and machine learning

Info from: https://www.heavy.ai/learn/gpu-database

CPU vs GPU

In other words: if GPUs are so fast, why don't we just replace CPUs?

	CPU	GPU
Used for?	Complex decision making, sequential processing, multitasking, diverse workloads	Parallel processing of large amounts of data
Memory Hierarchy*	More sophisticated; uses caches optimized for variety of data types & access patterns	Simple, optimized for handling large blocks of similar data
Cost & Power Efficiency	Generally, more power efficient, better for balancing processing power and wide variety of tasks	Not as power-efficient, designed for max throughput of specific types of computation

CPU Databases vs GPU Databases

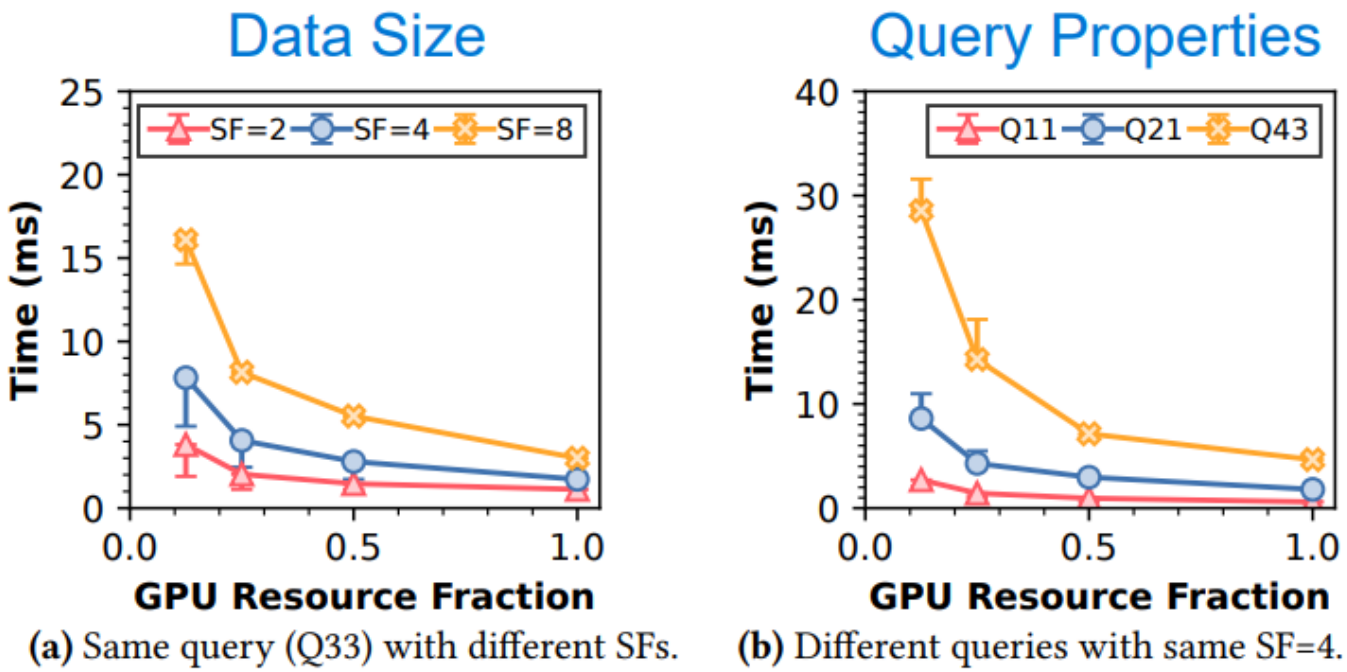
	CPU Databases	GPU Databases
Used for?	Transaction processing, complex queries, sophisticated data manipulation	Data analytics, machine learning, generally applying bulk calculations to large volumes of data in parallel
Examples	MySQL, PostgreSQL, Oracle	Crystal, BlazingSQL, HeavyDB, TQP, PG-Strom

Paper Review

Introduction

- **Paper Motivation**: study the resource utilization and bottlenecks in existing GPU database systems to propose improvements for better GPU database systems.
- Terminology:
  - SF (scaling factor): parameter to adjust size of dataset (larger SF = larger dataset)
  - GPU kernel: compute function
  - Kernel fusion: combine multiple independent kernels into one

Query Performance vs. GPU Resource Allocation



(a) Same query (Q33) with different SFs. **(b)** Different queries with same SF=4.

Figure 1: Query Execution Time (and Estimation Error) *vs.* Resource Allocation – Query execution time *vs.* GPU resource allocations for representative queries (§3.2) and scale factors (SFs), along with the error of the time estimates (shown with bars) using our predictive framework. X-axis denotes the fraction of GPU resources being used to run the queries.

- Query performance doesn't scale linearly with respect to allocated GPU
- Depends on data size and query properties
- Developers have many choices on how to schedule workloads over GPU, but it's not clear how best to utilize the resources

Key Contributions

1. Comparative Analysis of GPU DB systems
   - Finding traits that improve query performance
2. Performance Modeling
   - When data size changes
   - When resource allocation changes (roofline model)
3. Model Driven Resource Management & Scheduling
   - To predict workload performance with different resource allocation/degrees of concurrency

Databases Studied

Table 1: Qualitative comparison of GPU database systems – We consider the following characteristics. **Coverage**: whether the system is general purpose or only specific queries are supported. **Data Format**: formats for data input to the system. **Backend**: how data operators are compiled for execution. **Execution**: whether operators are executed on GPU or both on CPU and GPU. **Open Source**: if the system is publicly available.

System	Crystal	HeavyDB	BlazingSQL	TQP	PG-Strom
Coverage	SSB [53] only	General purpose	General purpose	General purpose	Join Aggr.
Data Format	Binary array	CSV Parquet	CSV, DF Parquet	CSV, DF Parquet	CSV
Backend	Hardcoded CUDA	LLVM to PTX	Thrust cuDF	PyTorch	CUDA
Execution	GPU	GPU	GPU	GPU	CPU+GPU
Open Source	Yes	Yes	Yes	No	Yes

Highlights from the Paper:

- **Crystal**: academic prototype, only supports queries from Star Schema Benchmark
- **Heavy DB**: LLVM to PTX
- **BlazingSQL**: sometimes generates intermediate results
- **TQP**: from Microsoft, PyTorch backend allows it to work with GPU kernels with built in PyTorch support
- **PG-Strom:** extension to PostgreSQL, allows offloading some operators (ex. join, aggregate) to GPU. Supports CPU + GPU execution

All have built-in query optimization and query compilation, except for Crystal.

Experimental Setup

- Hardware
  - NVIDIA A100 GPU (high end GPU)
- Workload
  - SSB benchmark, since Crystal only supports these queries
- Profiling Toolchain
  - NVIDIA Nsight System (time breakdown: data transfer, mem alloc, kernel execute)
  - Nsight Compute (kernel execution metrics)
  - nvidia-smi (power monitoring)

Performance Analysis

Observed runtime metrics of queries with fixed input size and GPU resource allocation

Performance Analysis: End-to-End Cold Query Execution

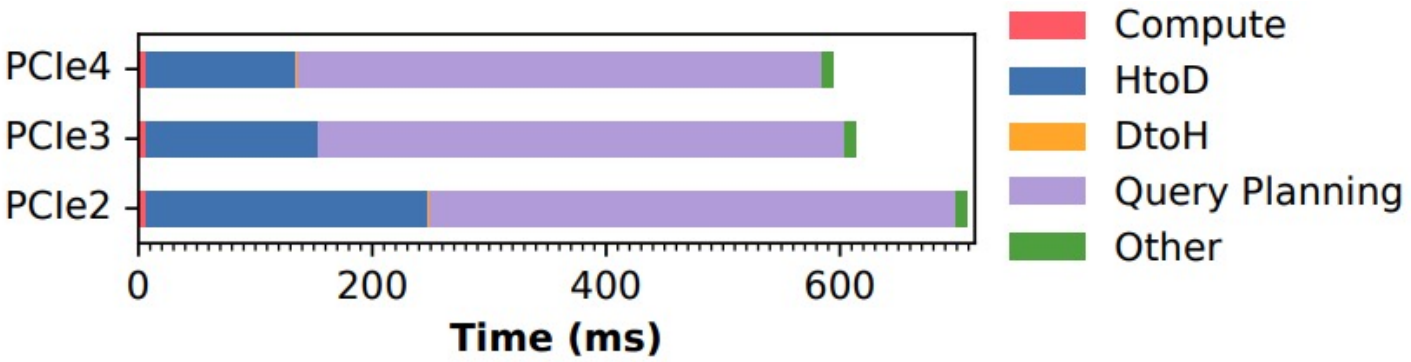


Figure 2: Cold execution characterization over Q21 at SF=16 in HeavyDB – We split the end-to-end performance of cold query execution into six components: *Compute* time, *Host to Device* data transfer (HtoD), *Device to Host* data transfer (DtoH), *Query Planning* time, and *Other* CUDA context setup and memory management time. Similar results apply for other queries and database systems.

2 main phases dominate run time:

- **Data Transfer** (Host to Device)
- **Query Planning** (unavoidable first time running query)

Performance Analysis: End to End Warm Query Execution

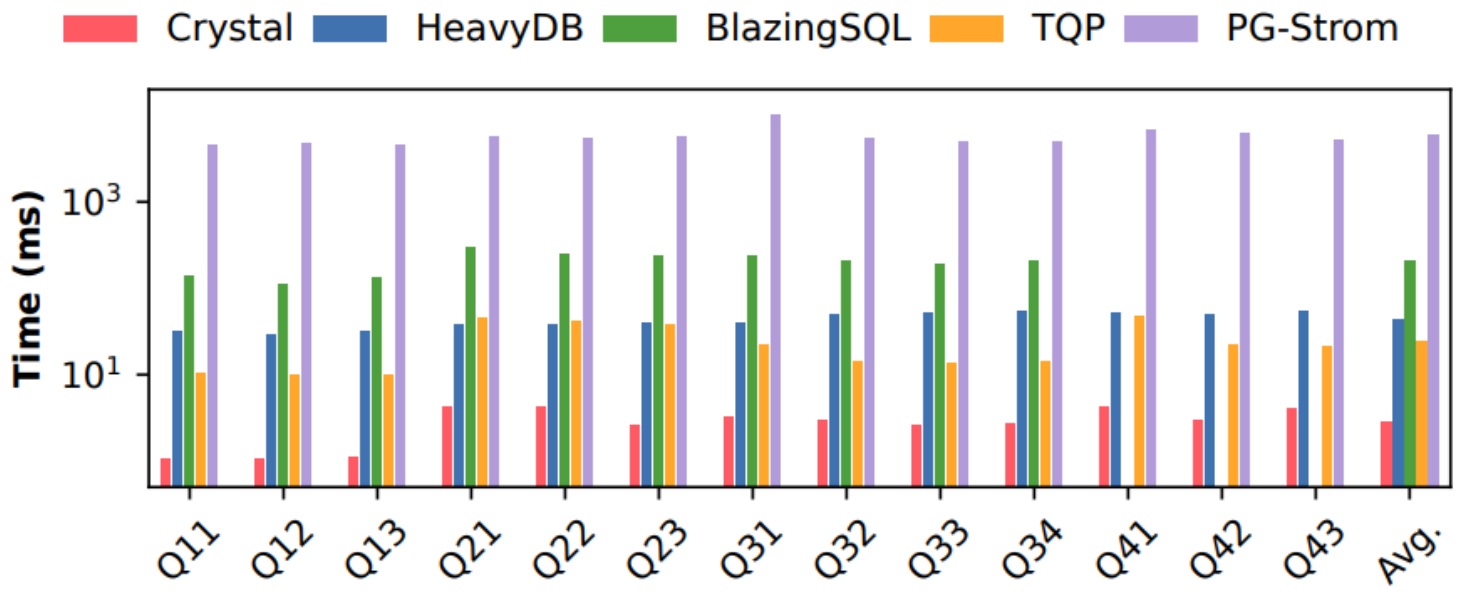


Figure 3: Warm execution characterization – End-to-end performance for all queries (BLAZINGSQL does not support queries in group 4).

Crystal & TQP: E2E is only the GPU compute time (can avoid compilation overhead)

HeavyDB & BlazingSQL: overhead from query plan optimization & compilation (unexpected)

PG-Strom: transfer of intermediate results between CPU and GPU is expensive

Performance Analysis: GPU Compute Time

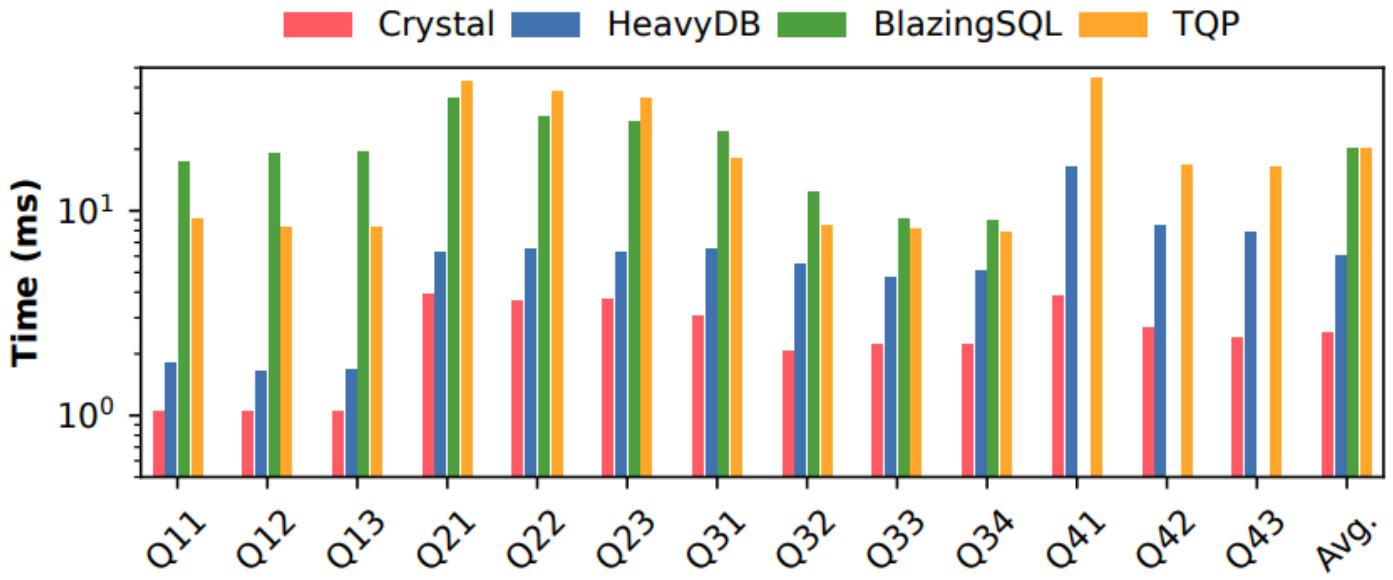


Figure 4: GPU compute time characterization – compared to the end-to-end warm execution of Figure 3, HeavyDB and BlazingSQL performance improve compared to Crystal, while TQP performance decreases.

Crystal is still the fastest, but advantage is less.

E2E execution time doesn't necessarily correlate with GPU compute efficiency

What influences GPU Compute Efficiency?

- Arithmetic Intensity (AI): num operations / num bytes read
- Stalls from loading data from device memory
  - Packing complex operations into memory requests doesn't necessarily help
- Amount of kernels used for a query
  - Using large amount of kernels → materializing intermediate data (want to avoid)

Performance Modeling

Predicting query performance with variable input size and resource allocation

Performance Modeling: “White Box”

- “White Box”: dependent on implementation of query operators, can estimate query performance for different input sizes

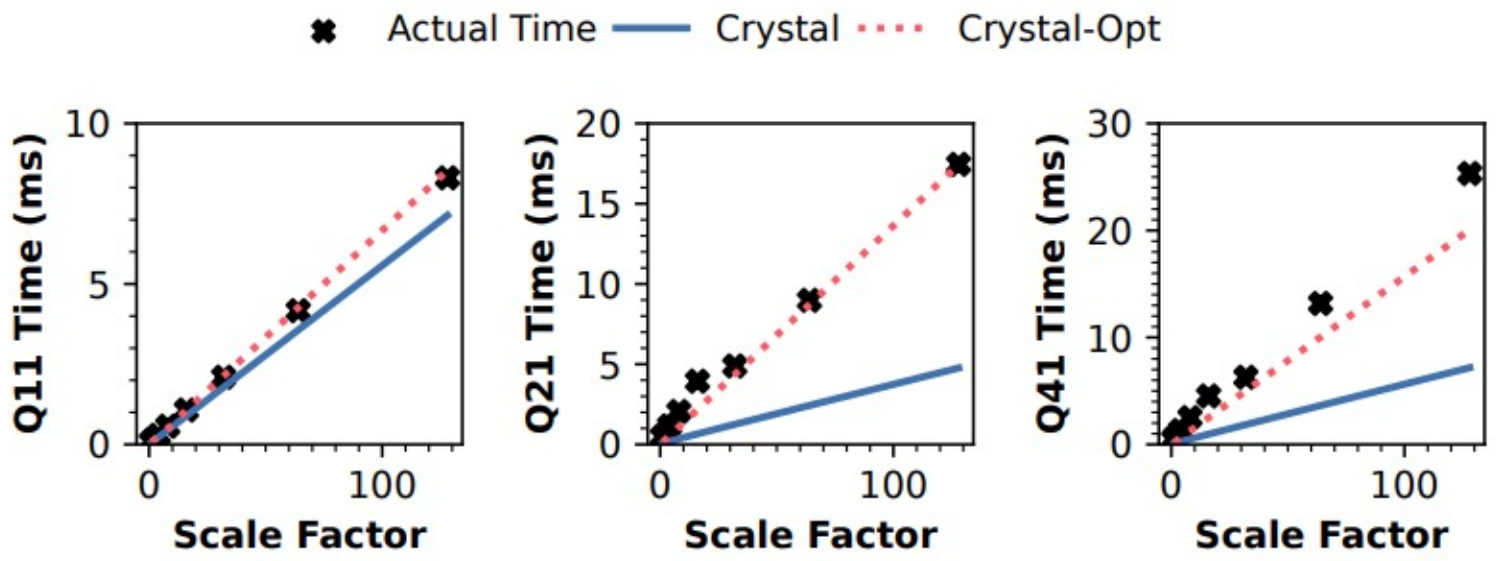


Figure 7: Estimation Time Comparison – Actual query execution time and estimated query execution time of CRYSTAL and CRYSTAL-Opt models.

Crystal: doesn’t account for less than full bandwidth utilization of DRAM and cache

“Crystal-Opt”: profiles previous run to get estimate of bandwidth utilization; model becomes more accurate

Memory bandwidth is often under-utilized

Performance Modeling: “Black Box”

- “Black Box”: agnostic of query implementation, can predict query performance as GPU resource allocation changes

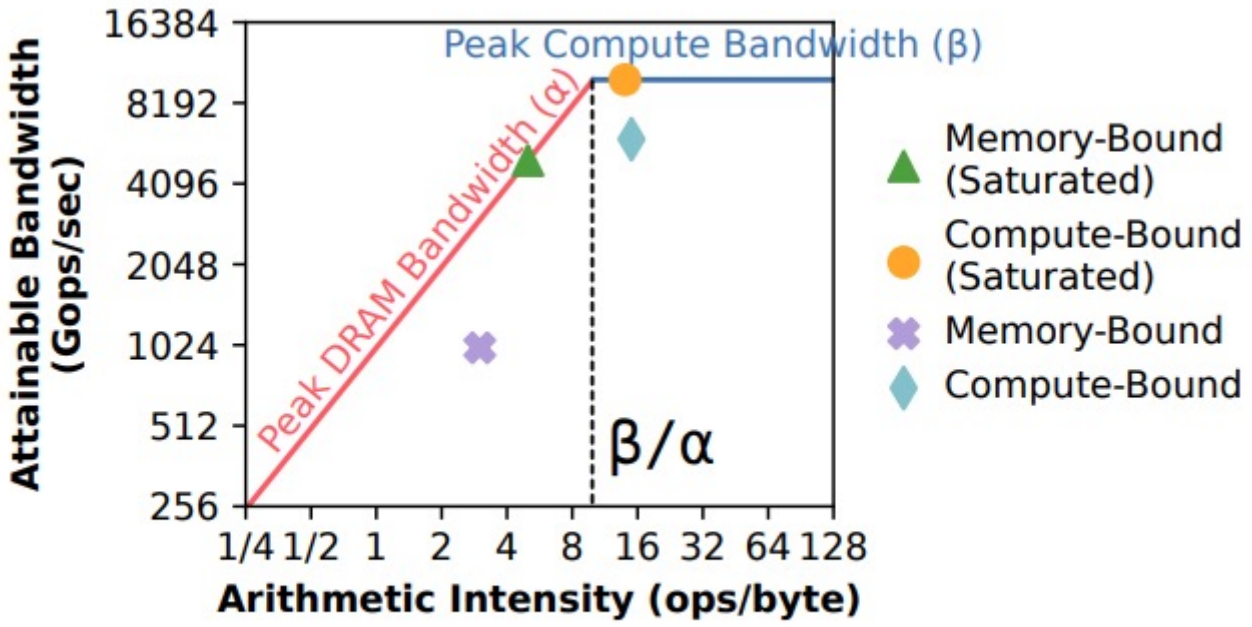


Figure 8: Roofline Model with different resource bounds.

Roofline: assumes that any execution on a specific hardware is bounded by either memory resources or compute resources

Slope = limited by memory resources

Flat = limited by compute resources

DRAM Roofline Model

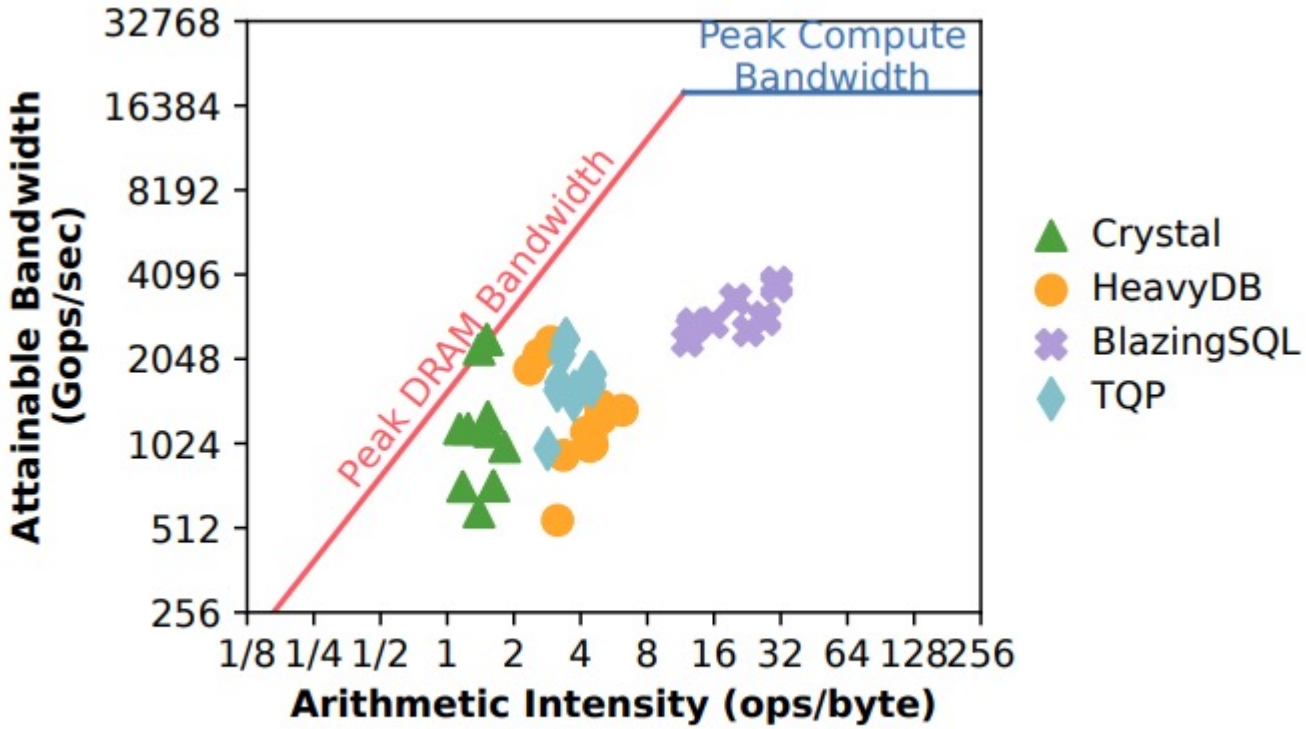


Figure 9: DRAM Roofline Model – for SSB queries (SF=16).

Arithmetic Intensity (AI):
operations / bytes read during execution

Most queries don't saturate compute bandwidth or GPU DRAM bandwidth
(not on the lines in the roofline model)
→ maybe another memory constraint?

L2-Cache Roofline Model

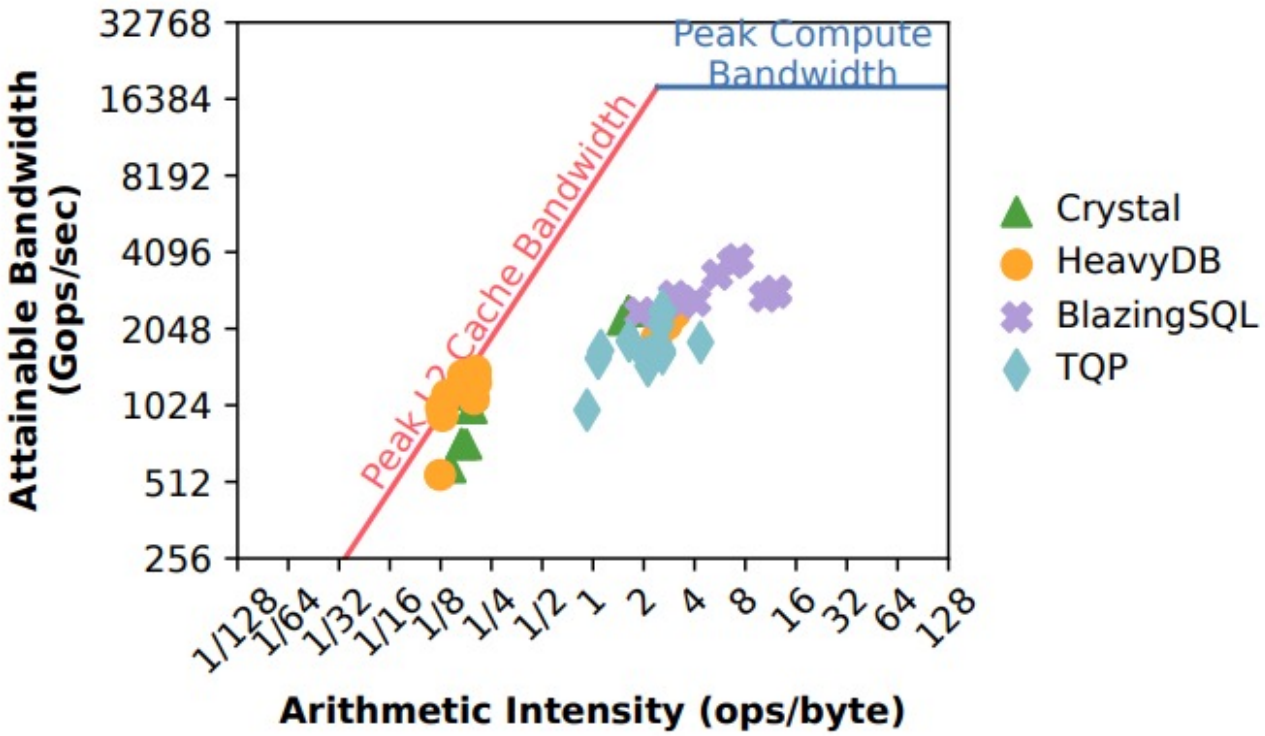


Figure 10: L2-Cache Roofline Model – for SSB queries (SF=16).

AI of a query with respect to the GPU **DRAM** is much different than AI of the same query with respect to GPU **L2 Cache** (tends to be less)!

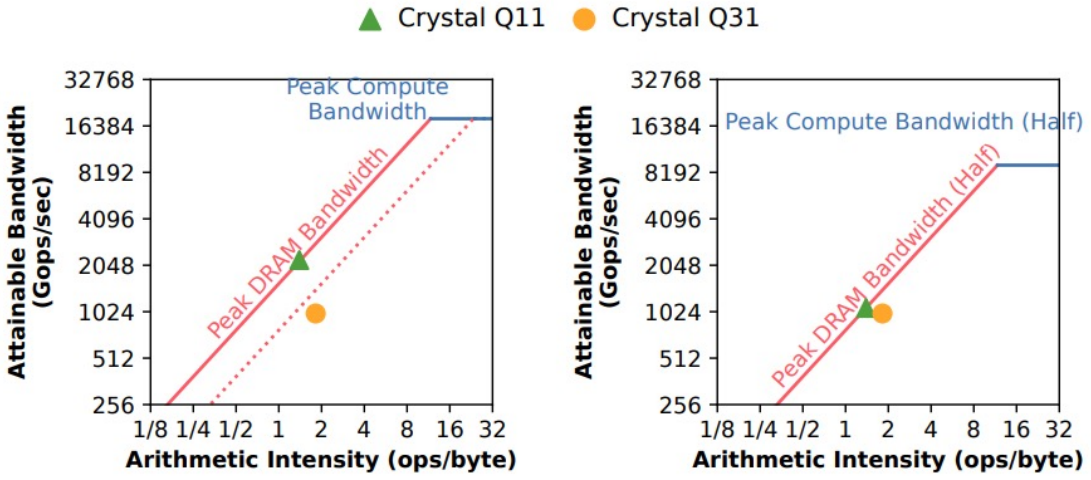
Good L2 Cache saturation leads to lower AI with respect to GPU DRAM

Crystal & HeavyDB are optimized to saturate the L2 Cache bandwidth

L2 cache utilization also depends on properties of the query

Resource Allocation Effects (i.e. Concurrent Queries sharing GPU)

Peak compute goes down, memory bandwidth same slope

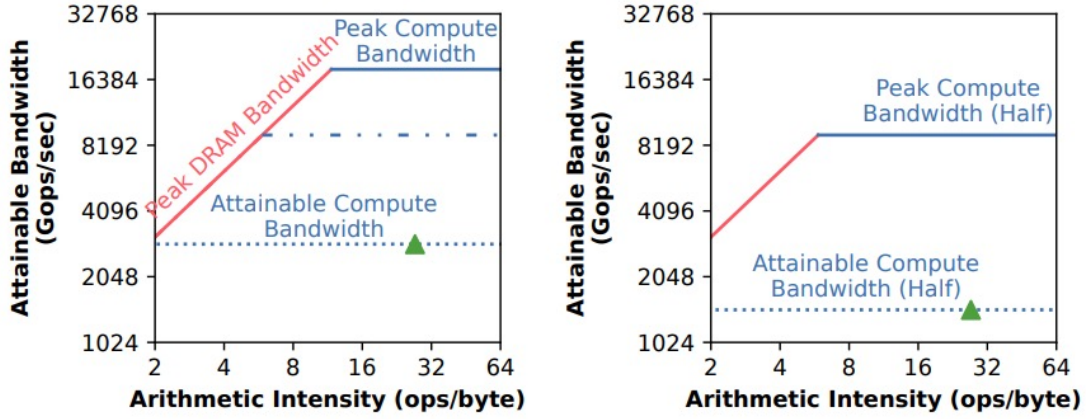


(a) Query performance, memory bandwidth, and compute bandwidth of *full* GPU resource (*dashed line is projected memory bandwidth of half GPU resource*).

(b) Query performance, memory bandwidth, and compute bandwidth of *half* GPU resource.

Figure 12: Memory-Bound Queries – Performance impact on memory-bound queries (Q11 and Q31) for CRYSTAL.

Peak compute goes down, so does attainable compute BW



(a) Query performance, memory bandwidth, and compute bandwidth of *full* GPU resource (*dashed-dotted line projects compute bandwidth of half GPU resource*).

(b) Query performance, memory bandwidth, and compute bandwidth of *half* GPU resource.

Figure 13: Compute-Bound Queries – Performance impact on compute-bound queries (Q34 from BLAZINGSQL).

Modeling Query Time

$$Slowdown\ (observed) = \frac{Query\ Exec\ Time\ (new\ configuration)}{Query\ Exec\ Time\ (old\ configuration)}$$

$$Slowdown\ (single\ process) = \begin{cases} \frac{1}{ComputeAllocationRatio}, if\ AI_{DRAM} > \frac{Bandwidth_{Compute}}{Bandwidth_{DRAM}}\ or\ AI_{L2Cache} > \frac{Bandwidth_{Compute}}{Bandwidth_{L2Cache}}, \\ \max(Slowdown_{DRAM}, Slowdown_{L2Cache}), otherwise \end{cases}$$

Compute Bound

Memory Bound

$$End\ to\ End\ Execution\ Time\ (concurrent\ processes) = \max(ExecTime_{p_1}, \ldots, ExecTime_{p_n})$$

How accurate is the roofline model vs. naïve linear scaling?

Linear: $t' = \frac{t}{ComputeAllocationRatio}$

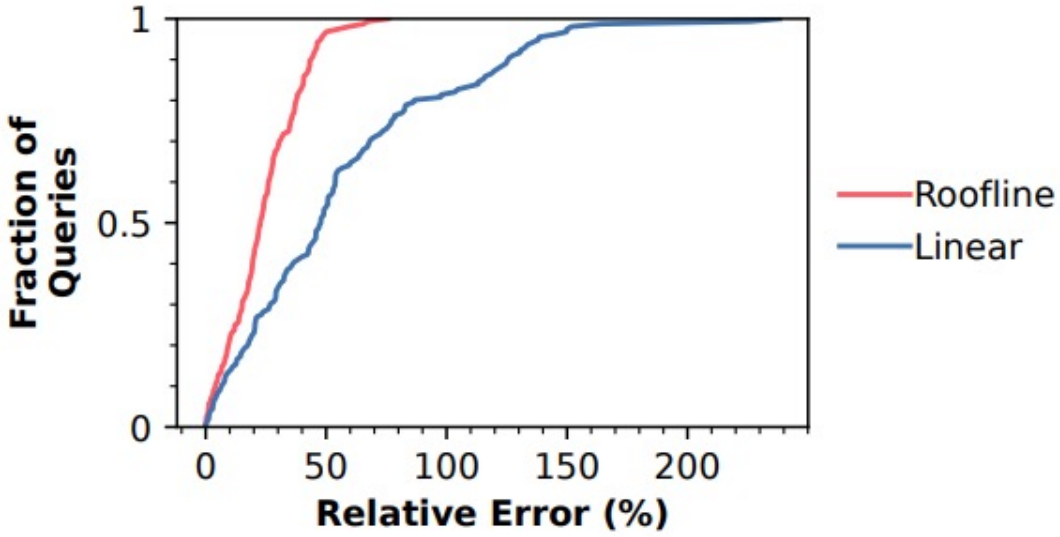


Figure 14: Query Time Estimation Error – Comparison of CDF of query time estimation error between roofline and linear model.

Model is substantially more accurate than the linear scaling:

- p50 error: 22% vs 48%
- p95 error: 48% vs 140%

Linear is good with large SF or sequential scan operators

Model is better for more complex queries

How beneficial is concurrent query execution?

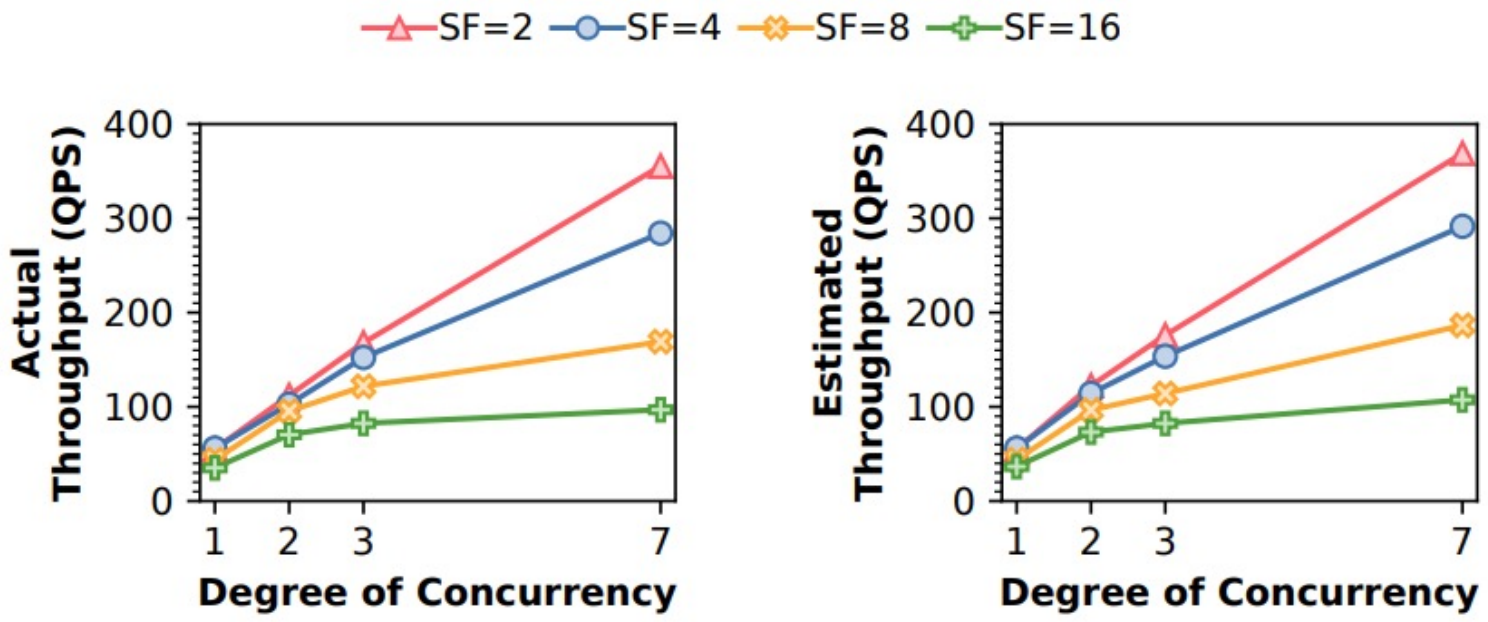


Figure 15: Throughput *vs.* Degree of Concurrency – Comparison between actual and estimated throughput of running queries concurrently.

At **small SF**, concurrent execution provides **near-linear** performance **improvement**.

At **larger SF**, queries are more easily affected by resource reduction; **increase but not linear**.

Estimated Throughput (right) very close to Actual Throughput (left); **model good estimator** for E2E query execution performance vs. degree of concurrency.

Research Directions

- Efficient and Automatic Kernel Fusion
  - Avoid materializing intermediate data
  - Apply CPU research to make fusion more efficient into GPUs
- Flexible Resource Allocation
  - GPUs allowing decoupled compute & memory allocation
- GPU Resource-Aware Query Optimization
  - Exploiting L1 Cache: Has higher capacity and lower latency now, can help with skewed workloads with high data reuse
- Time Prediction for Different GPUs
  - Paper only tested NVIDIA A100

Thank you!