A Comprehensive Empirical Study of Query Performance Across GPU DBMSes

Young-Kyoon Suh (Kyungpook National University, Korea)
Junyoung An (Kyungpook National University, Korea)
Byungchul Tak (Kyungpook National University, Korea)
Gap-Joo Na (ETRI, Korea)

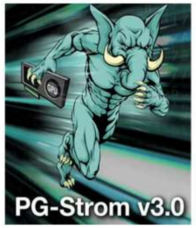
Contents

- Introduction
- The Structural Causal Model
- Empirical Evaluation
- Testing the Model
- Implications on Research and Engineering
- Conclusion and Future Work

Introduction: Background

- General Purpose computing on Graphics Processing Units (GPU)
  - Has received heightened attention from the data management community, thanks to its ***massive parallelism*** and ***high bandwidth***.

- The community has successfully yielded many ***GPU-accelerated database management systems*** (DBMSes).
  - Key idea: enabling query executions under co-processing with GPU
    - Market players:
    - Academic prototype engines:
      - HyPE(2013), CoGaDB (2014), GPL (2016), Voodoo(2016), HAPE (2019), HTAP(2020)

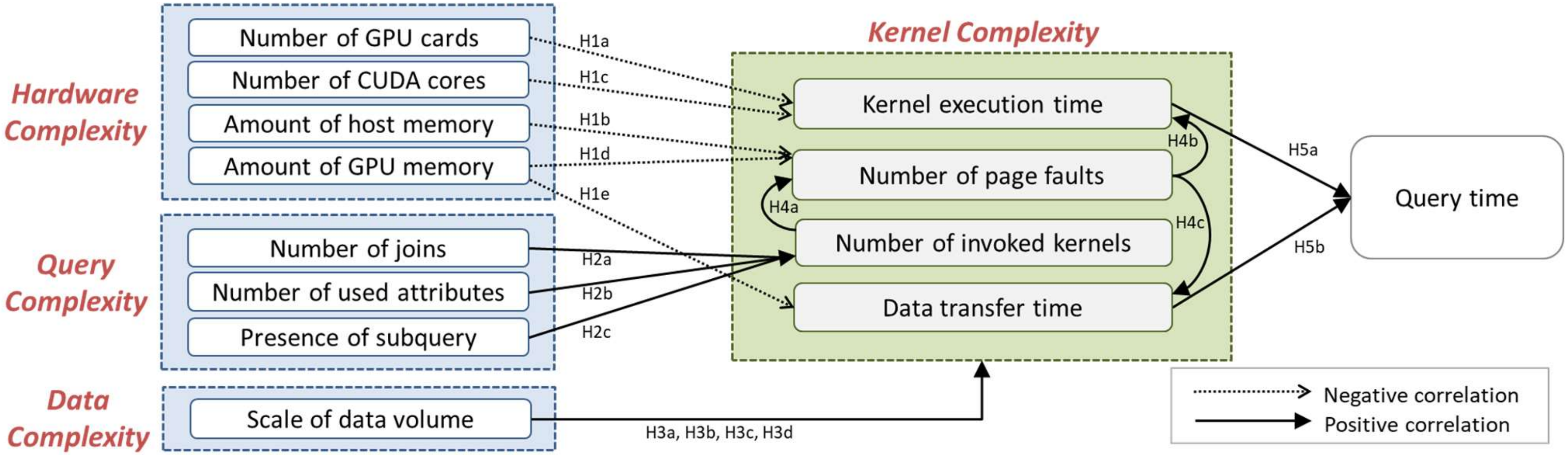


Introduction (Cont'd): Motivation

- Research questions regarding performance of GPU DBMS
  - *What factors affect the query evaluation and how do they interact?*
  - *Newer GPU model present an opportunity for better performance?*
  - *GPU DBMSes support a wide spectrum of queries well?*
  - *Is the use of multiple GPUs effective?*
  - *Is it scalable with the growing volume of data?*
  - *Does the under-utilization of GPU still exist?*

- We conduct a **comprehensive empirical study** to better understand the query performance of multiple, modern GPU DBMSes as a general class.

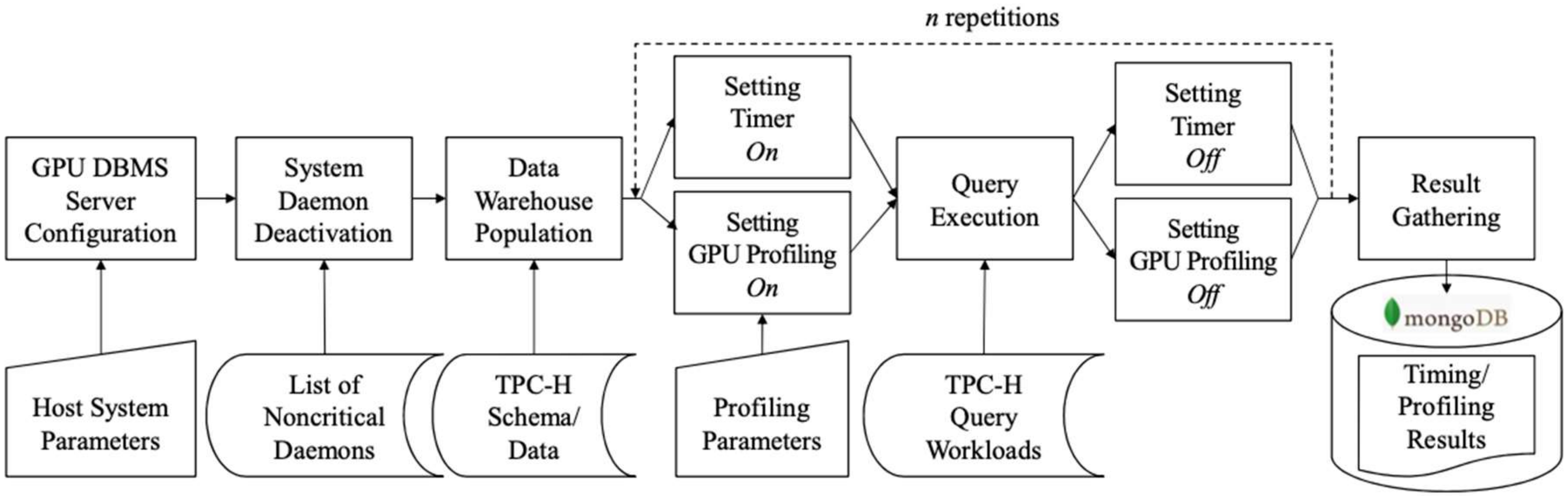
The Proposed Structural Causal Model



- Indicates *how* query time on a GPU DBMS may vary.
- Includes four constructs with 13 variables: eight independent variables (IVs) and five dependent variables (DVs).
  - Note that for some variables, they correlate with each other.
- Presents five major hypotheses with 17 detailed correlations:
  - They are ‘directed’ associations: 12 positive and 5 negative.

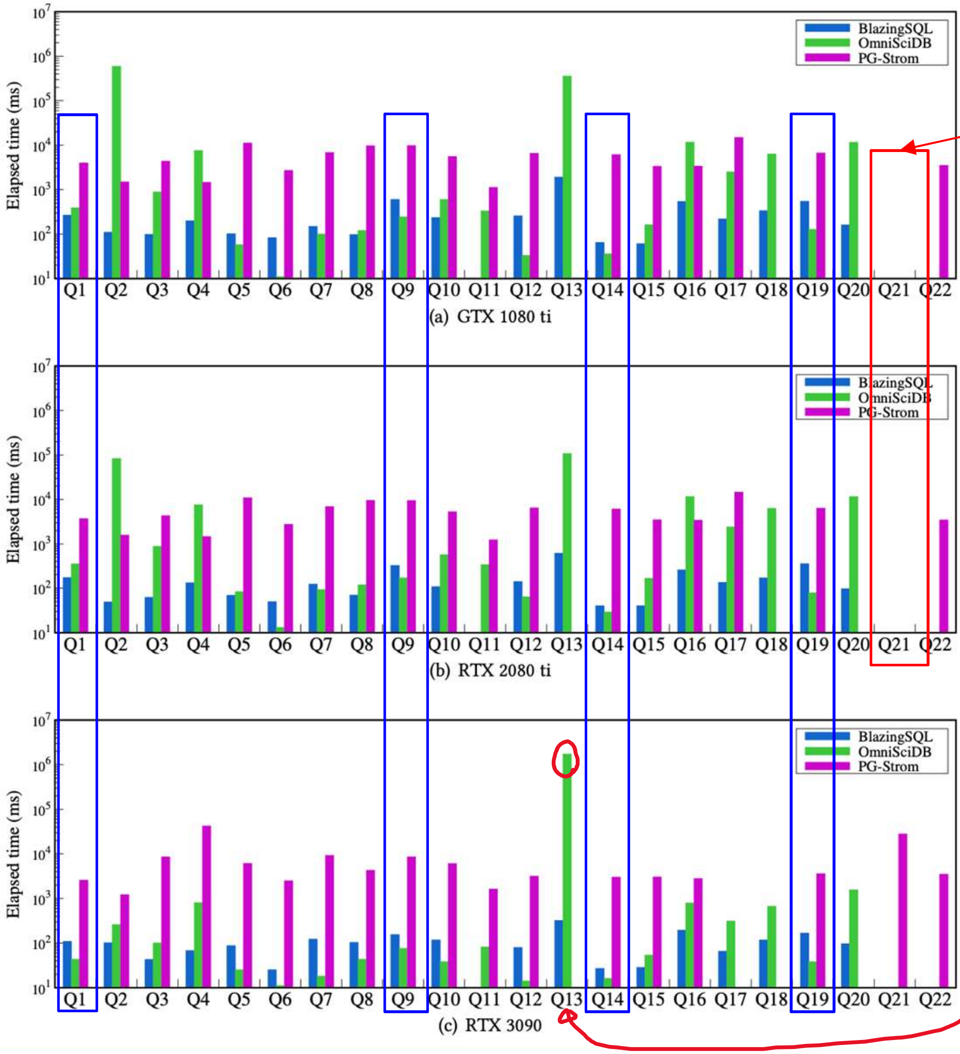
Empirical Evaluation (1 / 6)

- Three GPU DBMSes under test: stock configurations
  - BlazingSQL
  - OmniSciDB (previously known as MapD and now as HeavyDB)
  - PG-Strom (an extension of PostgreSQL)
    - Several other candidates (Alenka, CoGaDB, Kinetica, and SQreamDB) were excluded as they were deprecated, incompatible, or commercial.
- Applying a devised sophisticated timing methodology inspired by our prior timing techniques [SIGMOD'13, TODS'17, SPE'17, VLDBJ'22]:



Empirical Evaluation (2 / 6): Results

- Impact of Advancing GPU Models on Query Performance



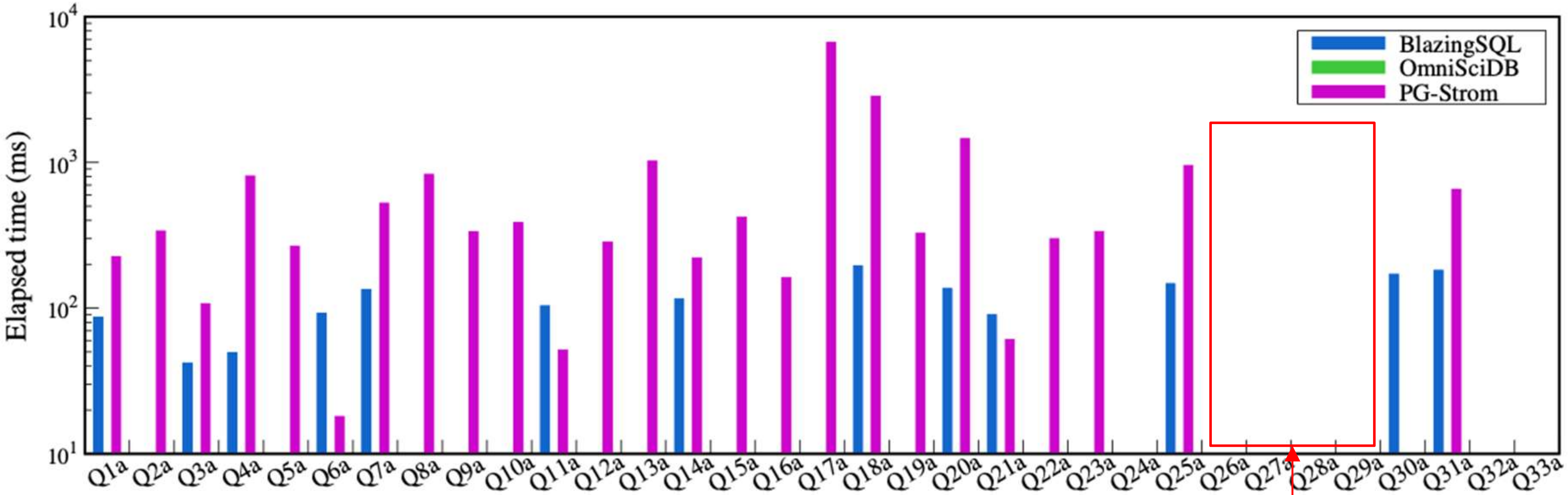
None of the studied GPU DBMSes could execute Q21 on GTX 1080 ti and RTX 2080 ti.

Q13: Slower at RTX 3090 compared to the earlier models

[**TPC-H** query performance on GTX 1080 ti, RTX 2080 ti, and RTX 3090]

Empirical Evaluation (3 / 6): Results

- Impact of Advancing GPU Models on Query Performance (Cont'd)

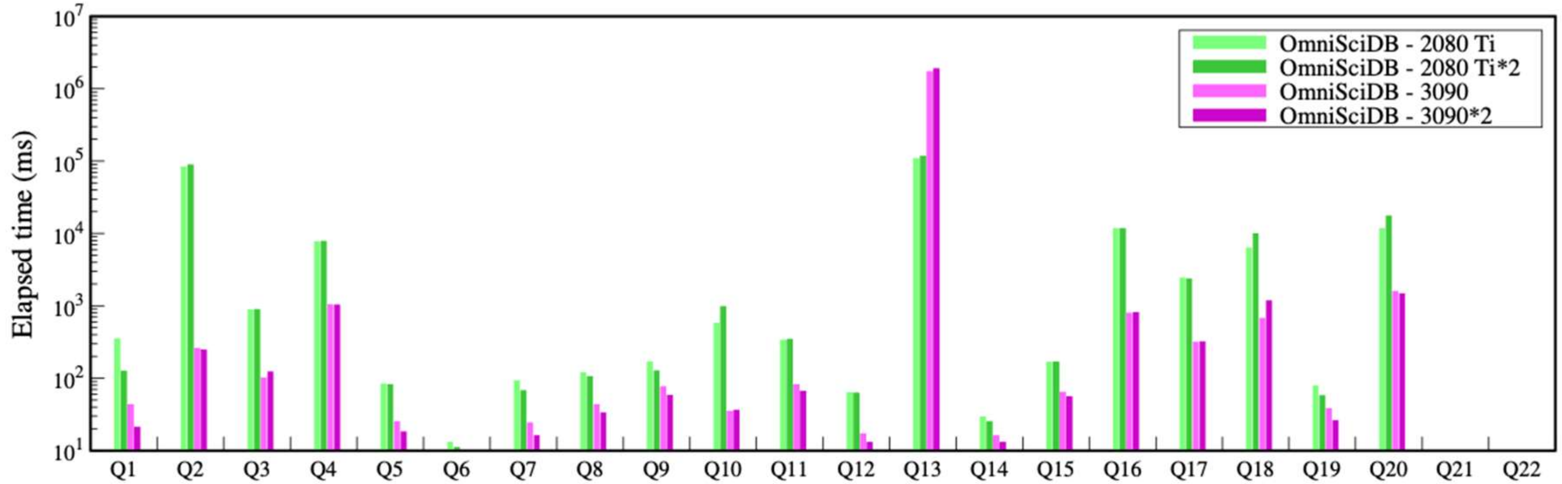


[**Join Order Benchmark (JOB)** query performance on RTX 3090]

- OmniSciDB could *not* run any JOB query because it didn't support an aggregate on text type.
- Some queries (Q26a ~ Q29a) could not be executed at all due to various reasons: *parsing error related to brackets*, *unsupported aggregation on text data*, *and inability to handle large text data.*

Empirical Evaluation (4 / 6): Results

- Impact of Multiple GPUs on TPC-H

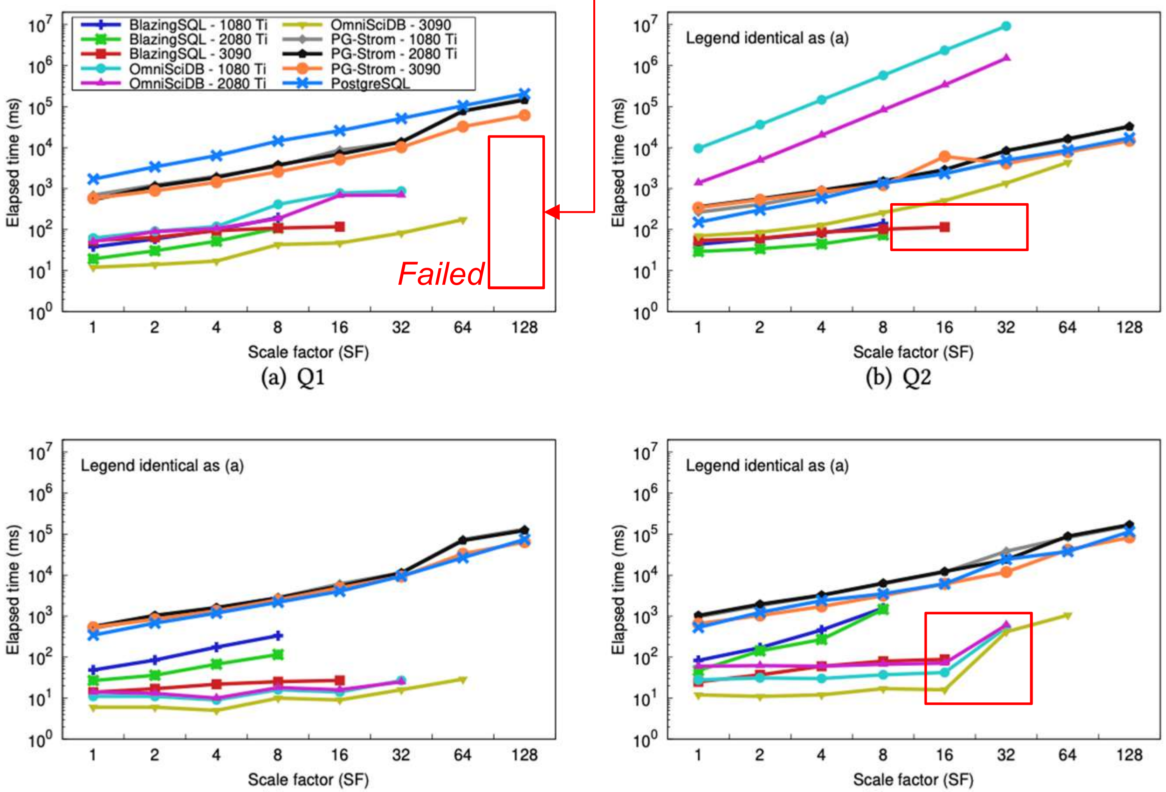


  - OmniSciDB: the only DBMS running normally with multiple GPUs.
    - That said, the dual-mode was unsuccessful in running Q21 and Q22.
  - BlazingSQL: faced a run-time error, ‘`CNMEM_STATUS_CUDA_ERROR`.’
    - Occurred due to a lack of GPU memory; revealed *inefficient intermediate result management*.
  - PG-Strom: required a commercial license to enable the multi-GPU feature.

More engineering efforts needed for exploiting multi-GPUs

Empirical Evaluation (5 / 6): Results

- Scalability test – *How scalable are they over growing DB?*
  - Exposed serious concerns observed at BlazingSQL and OmniSciDB.



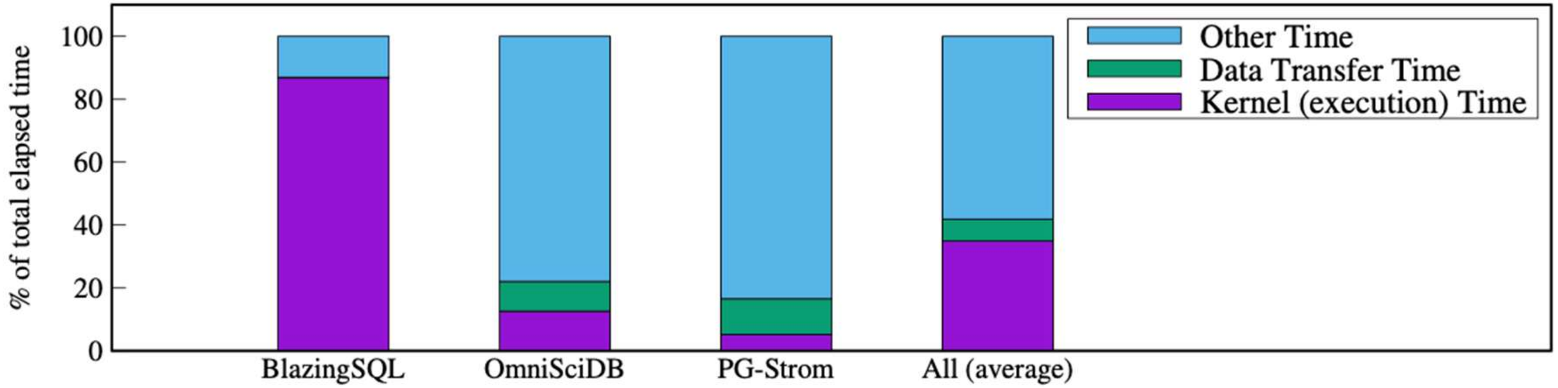
(a) Q1 (b) Q2 (c) Q6 (d) Q12

[Query performance over growing databases]

- BlazingSQL: overall outperformed the others under SF = 16; but *not beyond SF = 16*.
- OmniSciDB: overall best on RTX 3090 but revealed the bottleneck between SF = 16 and 32 on Q12; *unstable performance*
- PG-Strom: overall scaled better compared to but *underperformed* the others.

Empirical Evaluation (6 / 6): Results

- GPU Utilization - *Where is the query time spent?*



[Breakdown of average query time (of TPC-H) on RTX 2080 ti]

- Overall: 35% of query time spent on GPU
  - BlazingSQL: most utilizing GPU; about 87% of a query time
  - OmniSciDB and PG-Strom: suffered from GPU under-utilization
- Several implications for performance enhancement:
  - A need to achieve better GPU utilization
  - A need to offload on GPU a larger portion of query evaluation pipeline

Testing the Model

- Descriptive statistics of our dataset: refer to the paper
- Correlational analysis results
  - Obtained by `cor.test()` in R
  - All hypotheses in the model are *significant*.
    - Among 17 cells: 6 **green** cells, 11 **yellow** cells.
  - Provide strong empirical support for the validity of our model.

Variables	*Kernel execution time*	*Number of page faults*	*Number of invoked kernels*	*Data transfer time*	***Query time***
Number of GPU cards	**H1a: -0.156**	–	–	–	–
Amount of host memory	–	**H1b: -0.063**	–	–	–
Number of CUDA cores	**H1c: -0.110**	–	–	–	–
Amount of GPU memory	–	**H1d: -0.146**	–	**H1e: -0.278**	–
Number of joins	–	–	**H2a: 0.186**	–	–
Number of used attributes	–	–	**H2b: 0.231**	–	–
Presence of subquery	–	–	**H2c: 0.074**	–	–
Scale of data volume	**H3a: 0.377**	**H3b: 0.556**	**H3c: 0.289**	**H3d: 0.117**	–
Kernel execution time	–	**H4b: 0.535**	–	–	**H5a: 0.745**
Number of page faults	**H4b: 0.535**	–	**H4a: 0.161**	**H4c: 0.454**	–
Number of invoked kernels	–	**H4a: 0.161**	–	–	–
Data transfer time	–	**H4c: 0.454**	–	–	**H5b: 0.709**

Testing the Model (Cont'd)

- Regression analysis results

Dependent Variables	Amount of Variance Explained
Query time	**77.16%**
Kernel execution time	47.7%
Number of page faults	31.08%
Number of invoked kernels	13.89%
Data transfer time	21.84%

This implies that all other potential origins influencing Query time in the GPU DBMS will have *less* explanatory power than the variables of the proposed model.

Implications on Research and Engineering

- Criticality of Reducing the Kernel Execution Time
  - Our model indicates that *Kernel execution time* is as essential a factor as Data transfer time in reducing the Query time in a GPU DBMS.
    - *Number of GPU cards*, *Amount of GPU memory*, *Number of page faults*, and *Number of Invoked Kernels* are considered significant as well.
- Performance Dependency of the Type of GPU Devices
  - Overall, the more advanced model delivers the faster query time.
  - But the newer model is *not always* effective; a user's query rewriting needed.
- Signs of Poor Scalability
  - A lack of device memory significantly limits the scalability.
    - It would be helpful to exploit (i) a group of GPUs that can construct a shared device memory pool or (ii) a higher-end card with a larger memory for caching more data (including intermediate results) during QE.
- Low Utilization of GPU Resources
- The Need of Richer Query Operators

Conclusion and Future Work

- We have conducted a comprehensive empirical study across several modern GPU-based DBMSes to understand the performance characteristics better.
  - TPC-H/JOB analytical workloads were used in our study.
- We have explored and identified the key factors and proposed the *first* structural causal model of the Query time.
  - Our model explained a substantial portion (77%) of the variance.
  - Our analysis using the model presented several exciting insights: <u>significance of *Kernel execution time*</u>, <u>employment of an up-to-date GPU device with large memory</u>, and <u>reduction of data volume for querying</u>.
  - Our experiments revealed several performance concerns: *limited scalability*, *imperfect multi-GPU support*, *low GPU utilization*, and *lack of query operators*.
- Future work can take into account:
  - More complex factors (e.g., cache size, indexes, ...), a server-class GPU, other profiling measures, and any other GPU-based system.


thank you
danke
謝謝
gracias
merci
obrigado
bedankt
dziękuję
teşekkür ederim
ngiyabonga
go raibh maith agat
mochchakkeram
sukriya
kop khun krap
terima kasih
arigatô
dakujem
grazie
takk
tapadh leat
спасибо
dank je
감사합니다
ありがとう
мерси
рахмат
Баярлалаа
hvala
enkosi
faafetai lava
kiitos
dankie
dhanyavad
maururu
sagolun
tack
welalin
barka
mersi
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
djiere dieuf
chnorakaloutioun
gratias ago
gràcies
sulpáy
tanemirt
rahmet
najis tuke
kam sah hamnida
diolch
dhanyavadagalu
shukriya
trugarez
xièxie
ευχαριστώ


Q&A