Heterogeneous Parallel Programming COMP4901D

Relational Query Processing on the GPU

`

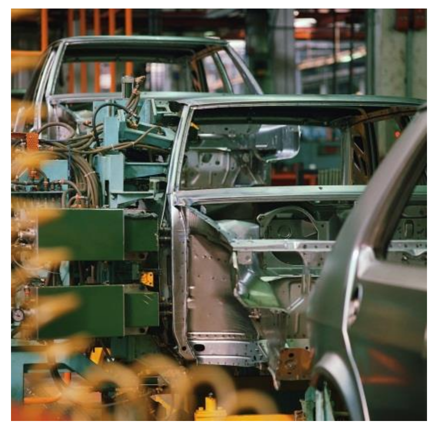
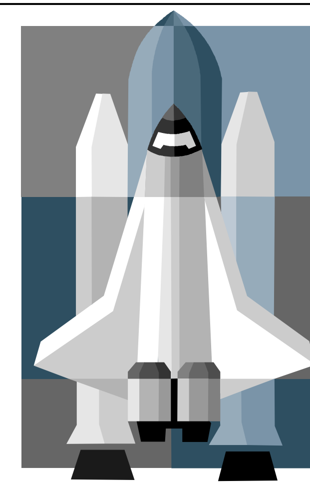
Overview

- Relational query processing
  - Relational operators: select, project, join, aggr.
  - Access methods: table scan, B+-tree, hashing
  - Physical query operators
- Using the GPU for query co-processing
  - What can be done on the GPU and what not
  - On the GPU, how operators are implemented
  - Is co-processing worthwhile? If so, how?

Database Management Systems

IBM
Microsoft SQL Server
ORACLE
SYBASE
hp invent
Teradata a division of NCR
MySQL
PostgreSQL



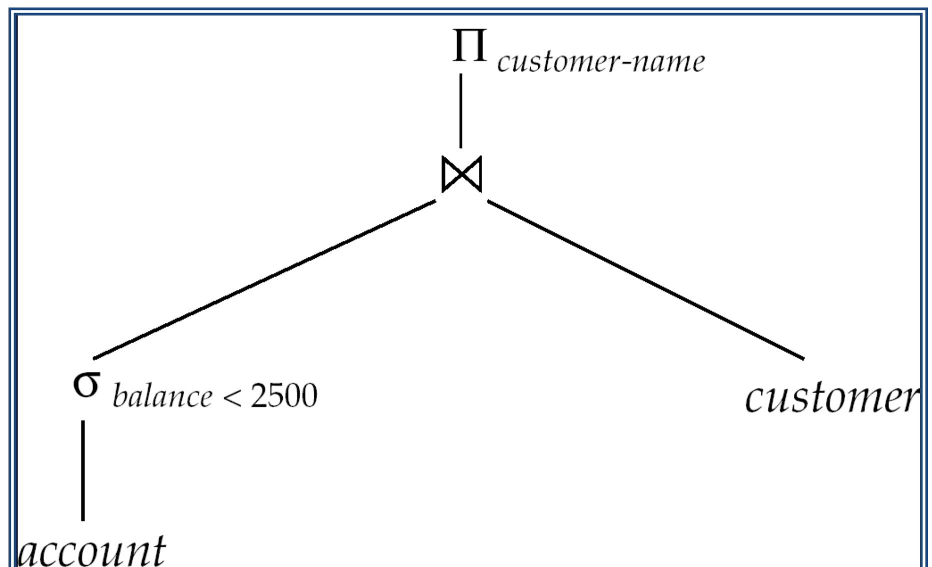



DBMSs have been a 40-year success in various applications.

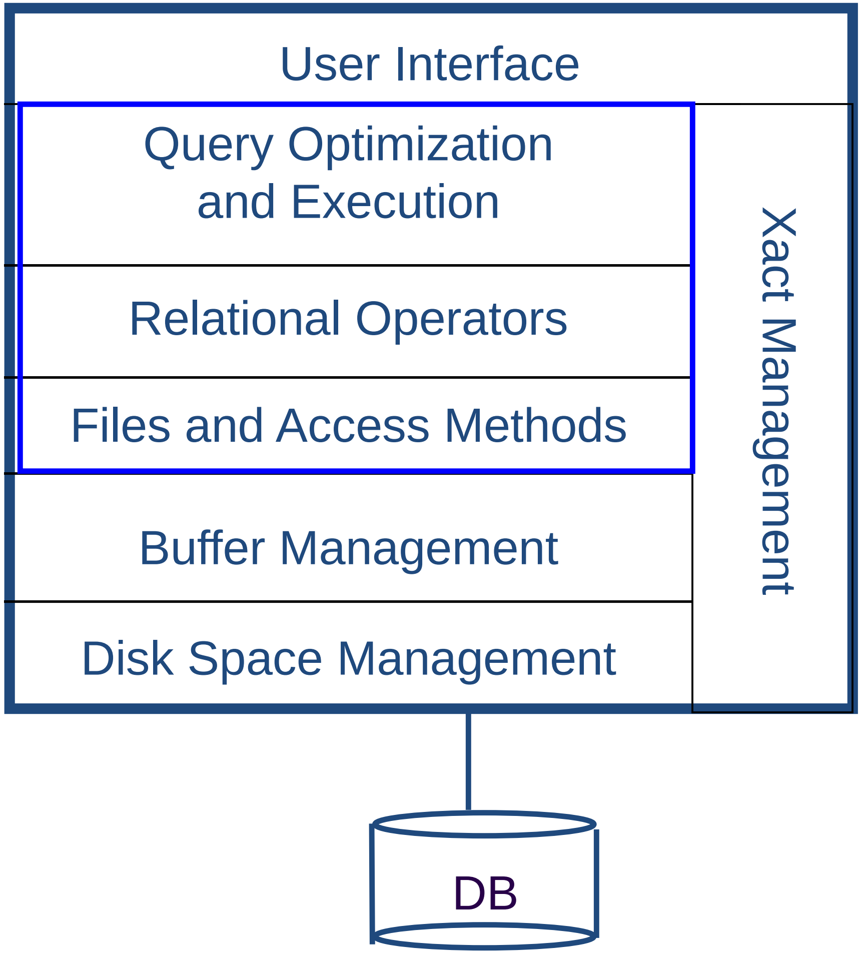
SQL and Query Operators

SELECT select-clause	- Projection, aggregation
FROM from-clause	- Join
[WHERE where-clause]	- Selection and/or Join condition
[ORDER BY order-by-expression]	- Sort
[GROUP BY group-by-attributes	- Partitioning
[HAVING condition-for-each-group]]	- Selection on partitions

```
SELECT customer-name
FROM account, customer
WHERE account.balance < 2500 AND
        account.customer-ID =
        customer.customer-ID
```



DBMS Architecture



Relational Query Processing

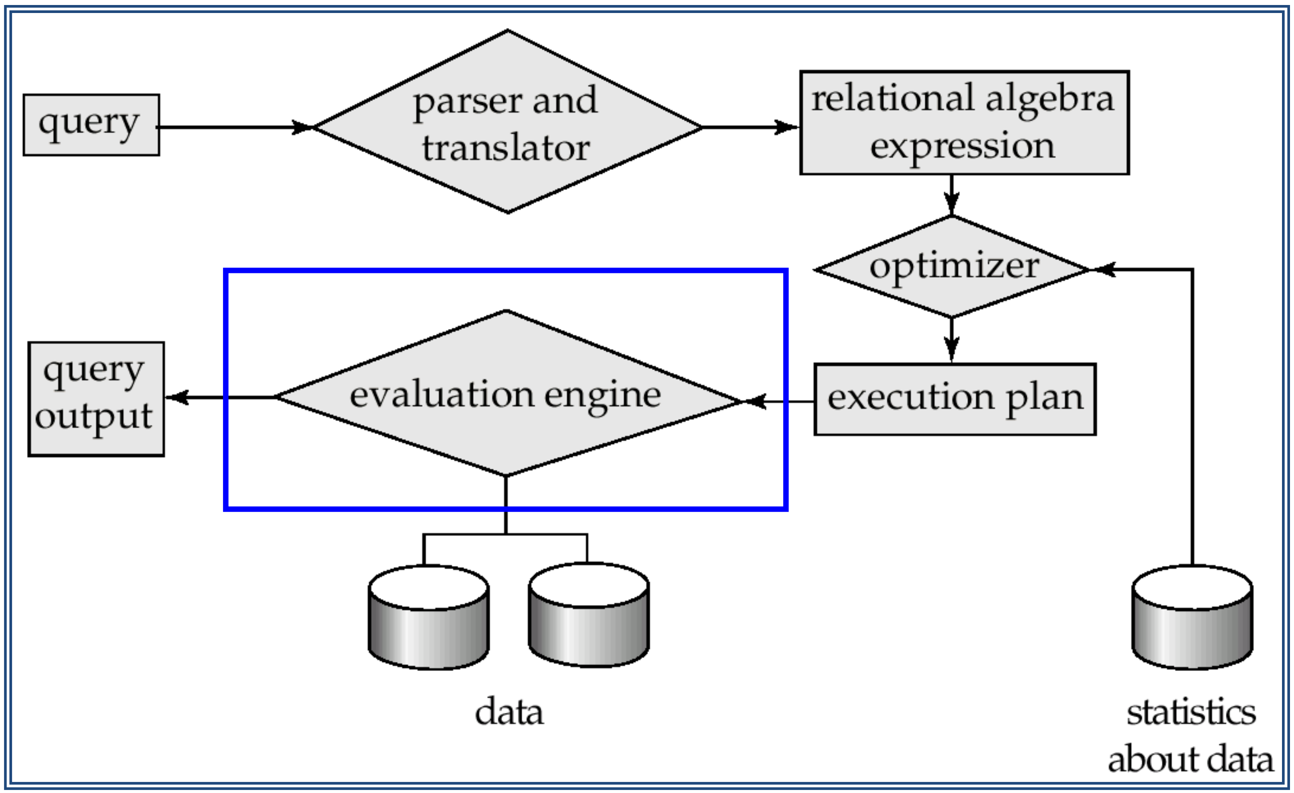
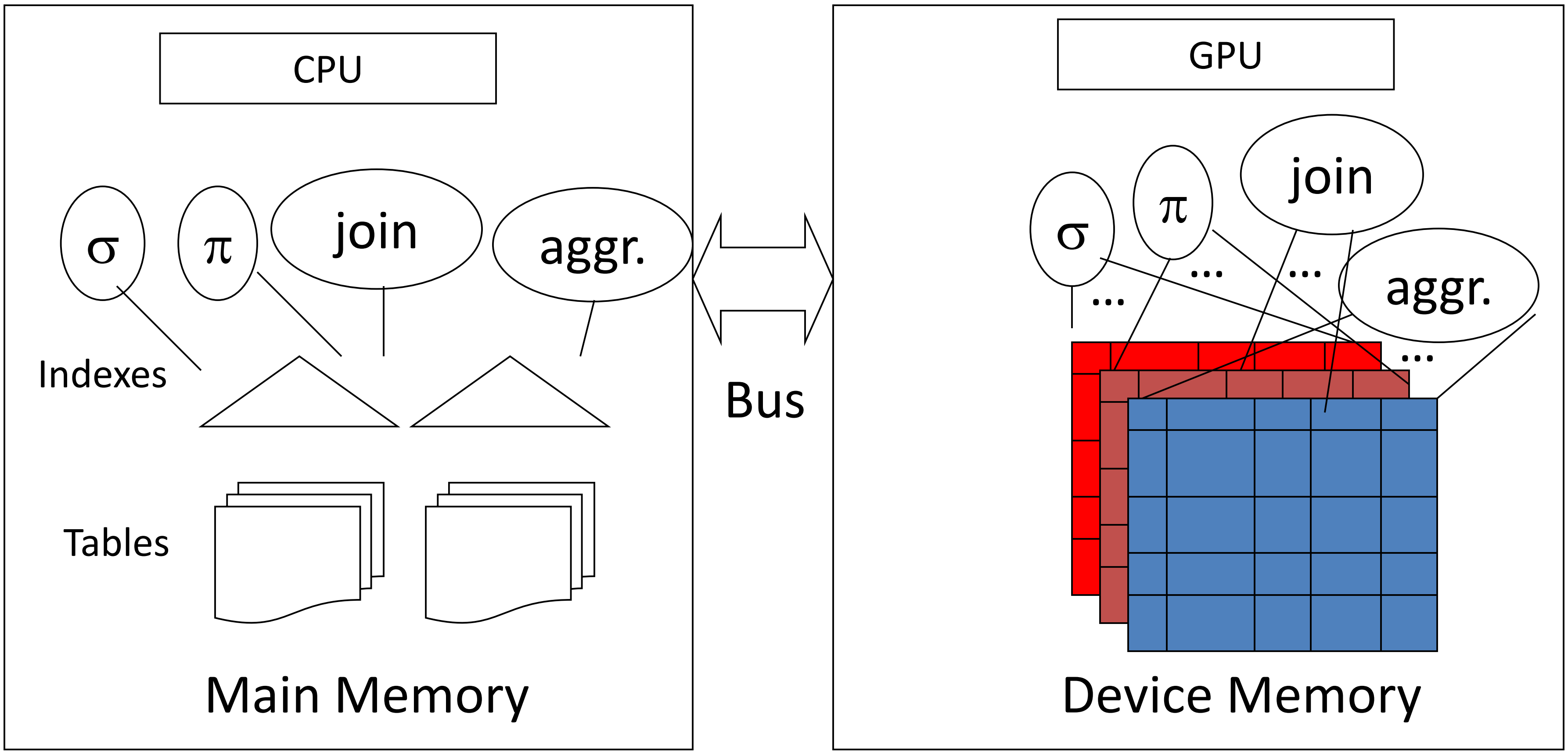


Figure source: Silberschatz et al.

Query Evaluation on the GPU

- Bring data into the GPU memory
- Construct GPU-suitable index structures
- Implement operators using primitives
- Determine result size lock-free
- Handle data skew for better parallelism

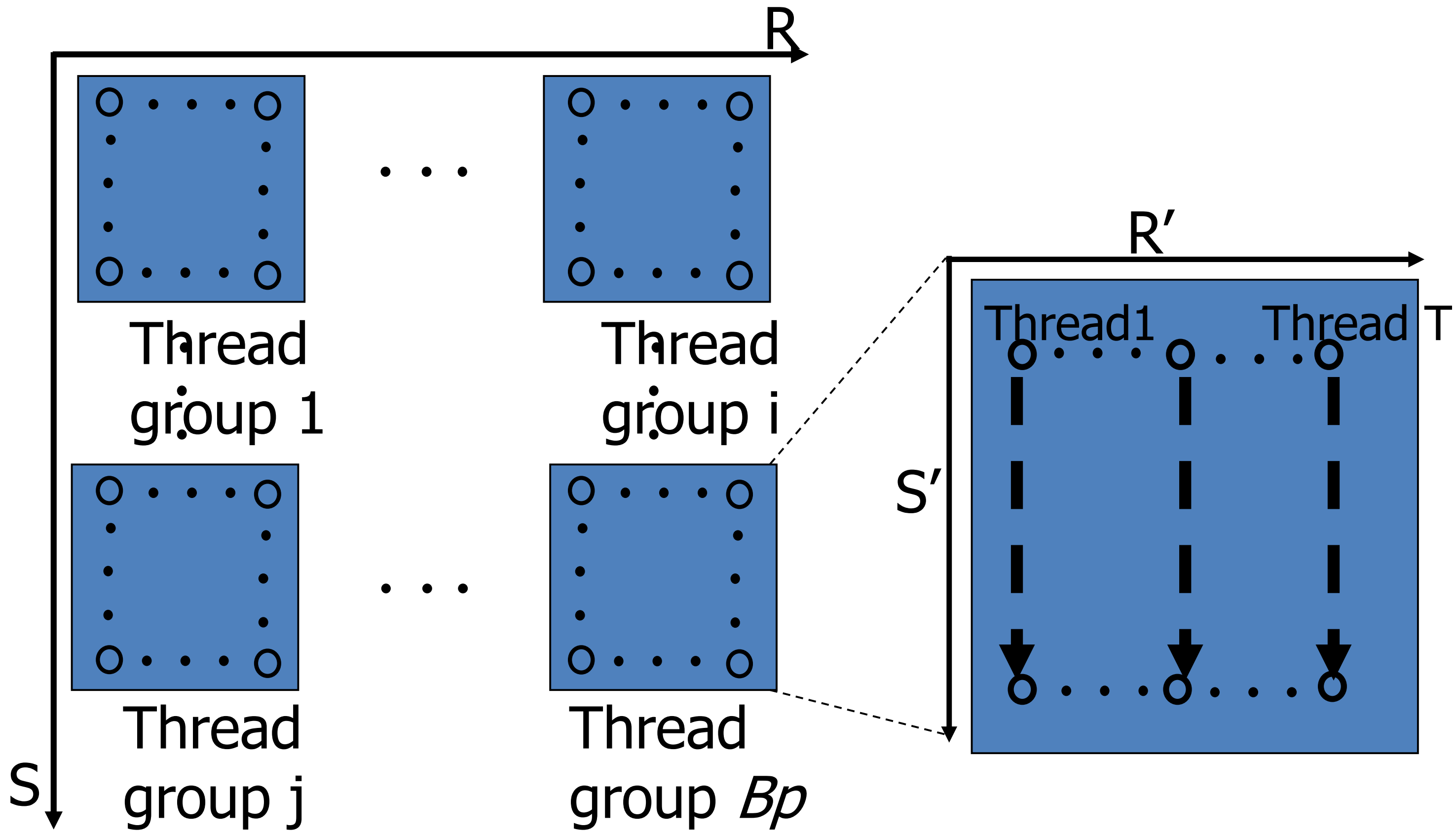
GPUQP Architecture



GPU-Based Join Algorithms

- Non-indexed nested-loop join (NINLJ)
  - Use the map primitive on both tables
- Indexed nested-loop join (INLJ)
  - Use the map primitive on the outer table
  - Adopt CSS-Tree [Rao99] to index the inner table
- Sort-merge join (SMJ)
  - Use sort on both tables and map for merging
- Hash join (HJ)
  - Adopt radix join [Boncz99]
  - Use split on both tables for partitioning

Nested-Loops Join on the GPU



Result Output Lock-free

- Problem: Join result size unknown
- Solution: Count result size before output
  - Each thread **counts** the number of join results for the partitioned join.
  - **Prefix sum** for write locations for each thread and the total number of join results.
  - Each thread **outputs** the join results in parallel.

Skew Handling in SMJ & HJ

- Identify the partitions that do not fit into the local memory.
  - Given an array storing partition sizes, we **split** it into two groups.
    - Partitions larger than the local memory
    - Partitions not larger than the local memory
- Decompose each of the large partitions into multiple small chunks.

Experimental Setup

Implementations

- CPU: OpenMP
- GPU: **CUDA** and DirectX

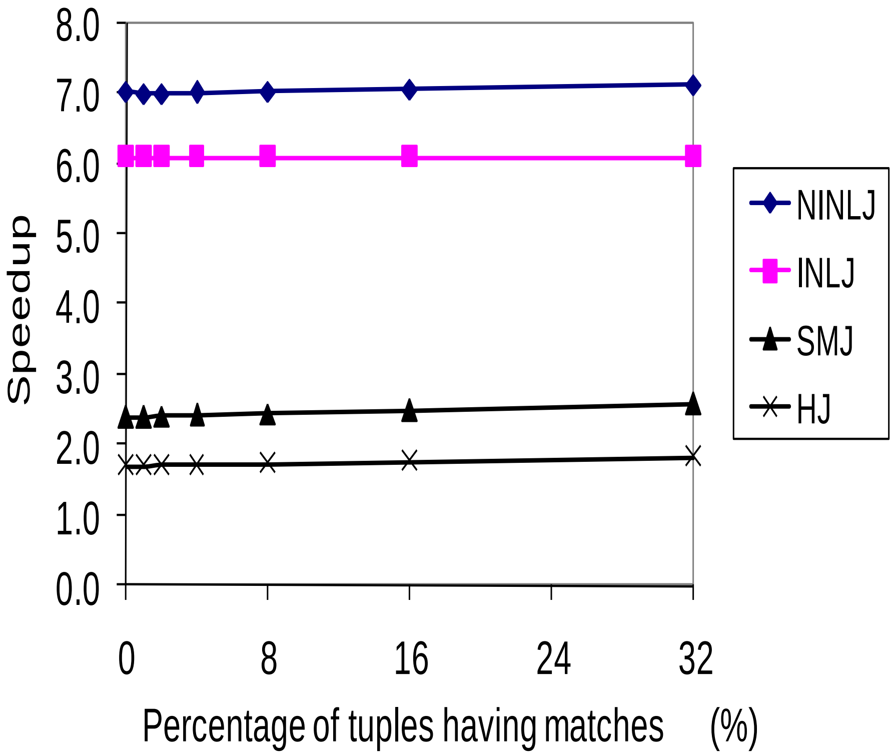
	CMP (P4 Quad)	GPU (NV G80)
Processors (HZ)	2.66G*4	**1.35G*128**
Cache size	**8MB**	256KB
Bandwidth (GB/sec)	10.4	**86.4**

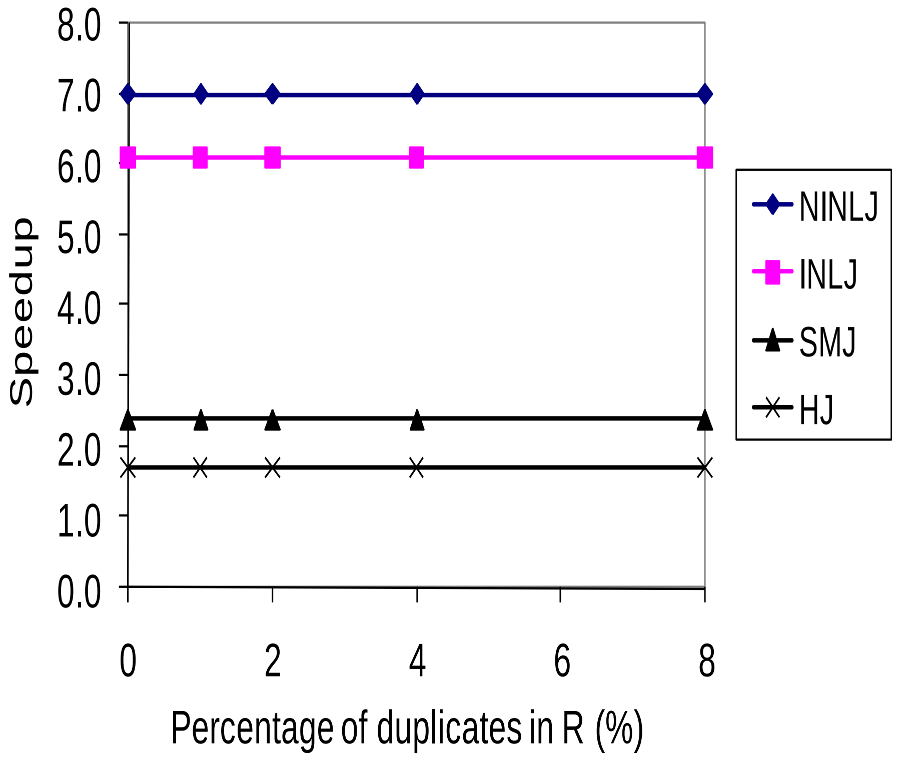
Performance on Uniform Data

Joins	CPU (sec)	GPU (sec)	**Speedup**
NINLJ	528.0	75.0	**7.0**
INLJ	4.2	0.7	**6.1**
SMJ	5.0	2.0	**2.4**
HJ	2.5	1.3	**1.9**

The GPU measurements **include** the time for data transfer between the GPU memory and the main memory.

Performance on Skewed Data





GDB: Beyond GPUQP

- Co-processing between the CPU and the GPU
  - The CPU handles disk IO, cost estimation, workload partitioning, and runs as a worker for a query when selected.
  - The GPU runs as a worker for a query when selected.
  - The cost model estimates the execution time of a query including memory stalls and computation.
  - Each operator can be (1) on the CPU only, (2) on the GPU only, and (3) on both processors.

Results from GDB

- The query cost model for the GPU was accurate.
- For TPC-H queries on disk-resident data, GDB's performance was similar to a commercial DBMS.
- For queries on in-memory data,
  - With data transfer time excluded, the GPU was 2-27 times faster than the CPU worker.
  - With data transfer time included, the GPU was 2-7 times faster on complex queries but 2-4 times slower on simple selections.
- GDB's co-processing achieved the best performance by making the right decision on where to execute a query.

Summary

- Relational query processing is a data-parallel task, suitable for GPU processing.
- Using data-parallel primitives as building blocks in GPU-based query processing simplifies programming and improves performance.
- Utilizing both the CPU and the GPU for query co-processing gets the best of both worlds.

http://www.cse.ust.hk/gpuqp