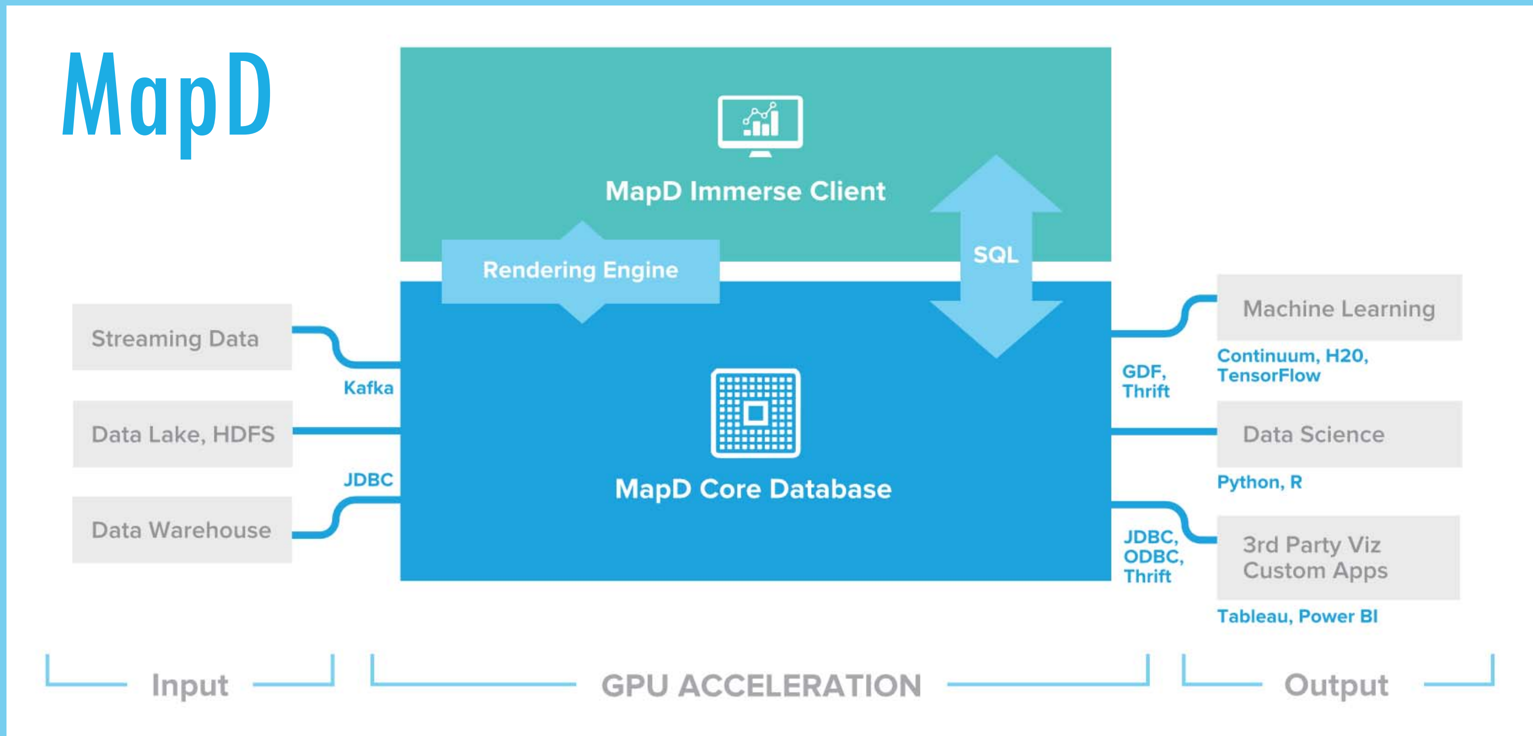
GROUP 1

Abeer Xiang David

Group DBA Level: Dora the Explorer



GPU OPTIMIZED, IN-MEMORY COLUMN-STORE DATABASE + VISUALIZATION TOOL

OBJECTIVES

- To understand the effectiveness of MapD in graphics processing units (GPU);
- To review the data storage, architecture, and access systems used in MapD;
- To understand the strengths and weaknesses of MapD.

The objectives of this project are manifold. First, it will attempt to understand the effectiveness of MapD in GPUs. Then it will review the data storage, architecture, and access systems used in MapD. Finally, it will compare MapD to traditional technologies and decipher its strengths and weaknesses.

OVERVIEW OF MAPD

- A visualization and database platform which uses the power of GPUs to process data faster;
- Used to instantly analyze large amounts of data;
- Improves performance of systems.

So now the question comes, what is MapD. MapD is a visualization and database platform which uses the power of GPUs and causes faster data processing. In doing so, it exponentially improves the performance of systems. It is primarily used to instantly analyze large amounts of data.

DATA MANAGEMENT FUNCTIONS OF MAPD

- Processes information in GPUs;
- Facilitates video rendering;
- Helps communication by making database management systems process data;
- Performs analytics of data.

MapD has many functions that are used in data management. It processes information in GPUs and performs data analysis. MapD is also used in video rendering. It helps communication as well by faster processing of data, and thereby facilitates learning and performance.

ARCHITECTURE, DATA STORAGE, AND ACCESS

- Data is passed on through GPU frames;
- SQL queries are divided up into small chunks and then combined at the end;
- Data is stored in datasets arranged in rows;
- Data is accessed using GPU frames, by being mapped on top of the frames.

This portion talks about the technology and architecture of data storage and access in MapD. First, data is passed through the GPU frames. After that, SQL queries are divided into small chunks and sorted in datasets, which are then arranged in rows. This data is combined together at the end. This data can be accessed by using GPU frames, as they are mapped on top of the frames.

STRENGTHS AND WEAKNESSES

- RDBMS is mostly used for storage of data. MapD can help with advanced tasks like processing and analytics;
- RDBMS can only present data in form of relations. MapD can presents the data in any format, as required;
- Data in RDBMS is manipulated in tables, which can be beneficial for someone looking to deal with exclusively that format. This is not the case for MapD.

As can be seen from the above discussion, in many cases, MapD is preferable over traditional technologies like RDBMS. For example, RDBMS can only help in the storage of data, but MapD can do other tasks like processing and analysis on top of that. RDBMS also presents data only in the form of relations, unlike MapD, which can provide the data in any format, as requested. Data in RDBMS is manipulated exclusively in table format, which can be beneficial for someone looking to work in simple formats. However, MapD uses better technology in data manipulation.

MAPD OPTIMIZATIONS

Apache Calcite

OTHER CONSIDERATIONS

Idea behind in-memory DB: increase performance by keeping data and code as local (to CPU) as possible

Other low level optimizations:

- Hardware – use GPU
- Compile queries (translate to machine code prior to running)

QUERY COMPILING IN MAPD

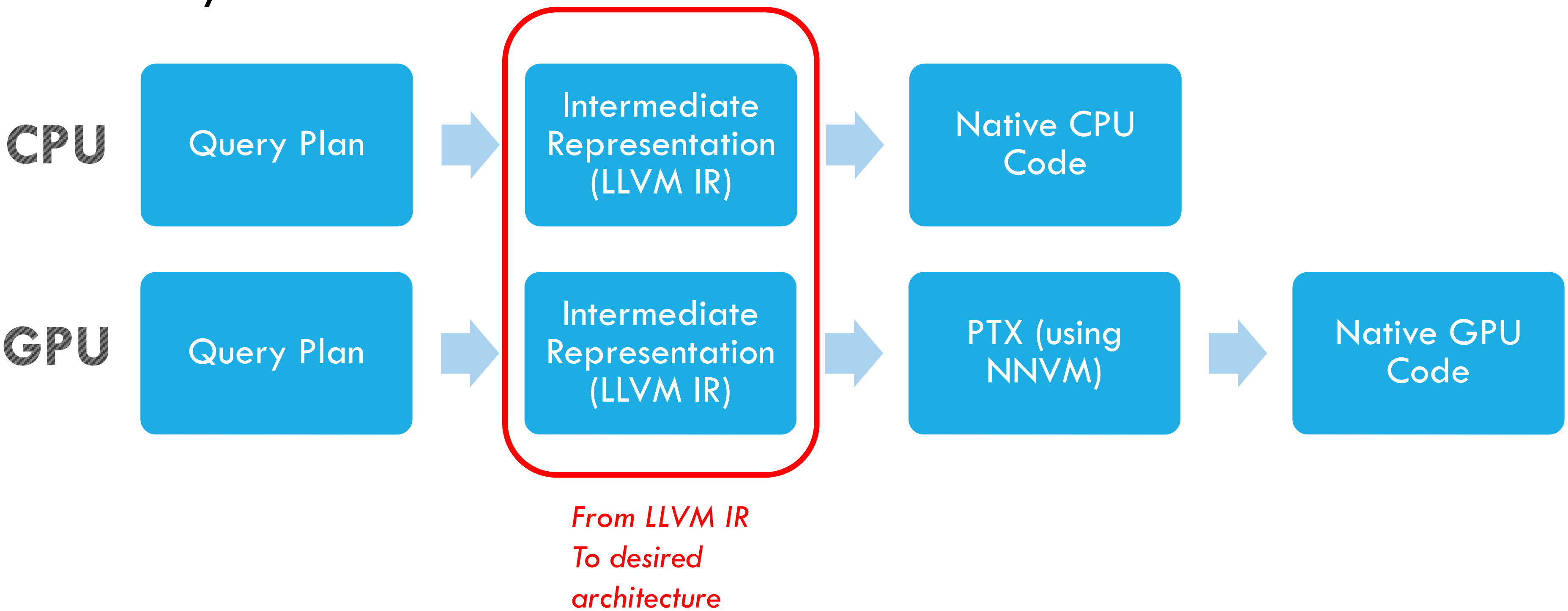
- Just in Time (JIT) compiling using LLVM (Low Level Virtual Machine) compiler infrastructure

LLVM

- Customizable compiler infrastructure
- Converts code to an intermediate representation (LLVM IR), and then into machine code using architecture-specific backends
- Built-in code validation APIs and tools (ensures well-formed queries)
- Optimizes code at runtime based on dynamic information

QUERY COMPILING IN MAPD

Query execution:



MAPD VS. TRADITIONAL DMBS COMPILER INFRASTRUCTURE

Key components of LLVM infrastructure:

- Compiled
- Low level
- JIT (just in time)

COMPILED VS. INTERPRETED

- Traditional DBMS Interpreted (translated to machine code at runtime, line by line)
- Interpreted vs Compiled: x*2+3

Evaluate x*2 for a number of rows → Store in buffer → Read intermediate result from buffer → Compute final result (add 3)

Evaluate x*2 → Store in register → Compute final result (add 3)

LLVM JIT compiling means that compilation only occurs if the benefits are greater than the costs

Compiled query avoids:

- Write to buffer
- Read from buffer
- Using memory for above ops
- Using cache

Compiled query adds:

- Compile overhead

Low Level

LOW LEVEL VS. HIGH LEVEL COMPILER

	LLVM	JVM
VM architecture	Register-based • low level • very fast	Stack-based • Higher level • garbage collection, objects, etc. • VM runs during program execution
Compiles to…	Native Code	Bytecode
Output processing	NA	Bytecode interpreted by VM (translated at runtime) or compiled to native code

Interprets

Compiles

Compiled ***Interpreted***

SPEED ⟷ FLEXIBILITY

JIT VS. NON-OPTIMIZED LOW LEVEL COMPILER

JIT

Efficiently Compiling Efficient Query Plans for Modern Hardware – Neumann, VLDB, 2011

- Used 5 different types of query to benchmark **LLVM-compiled query**, **C++ compiled query**, standard disk-based DBMS, and two others:

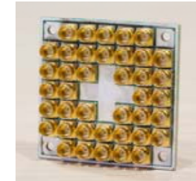
	Q1	Q2	Q3	Q4	Q5	
HyPer + C++ [ms]	142	374	141	203	1416	← Fast execution
compile time [ms]	1556	2367	1976	2214	2592	← Large compile time
HyPer + LLVM	35	125	80	117	1105	← Fast execution
compile time [ms]	16	41	30	16	34	← Small compile time
VectorWise [ms]	98	-	257	436	1107	
MonetDB [ms]	72	218	112	8168	12028	
DB X [ms]	4221	6555	16410	3830	15212	← Traditional DBMS

Table 2: OLAP Performance of Different Engines

COMPILED	*INTERPRETED*	
LLVM	JVM	Other Interpreted
• MapD • memSQL • Cloudera Impala • VitesseDB	• Oracle • Apache Spark	• PostgreSQL* • Many others

QUANTUM QUERY COMPILING?

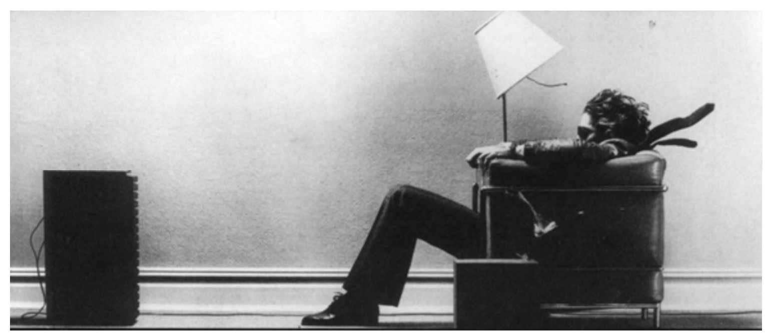
10/10/17 update:
Intel prototypes 17-qubit chip
100,000+ simultaneous states



Query Plan → Scaffold → QASM (using ScaffCC)

Query Plan → CPU / GPU / QPU

Scaffold: C-like quantum programming language, high-level
ScaffCC: LLVM-based compiler framework for Scaffold
QASM: Quantum assembly language (native code for quantum machines)

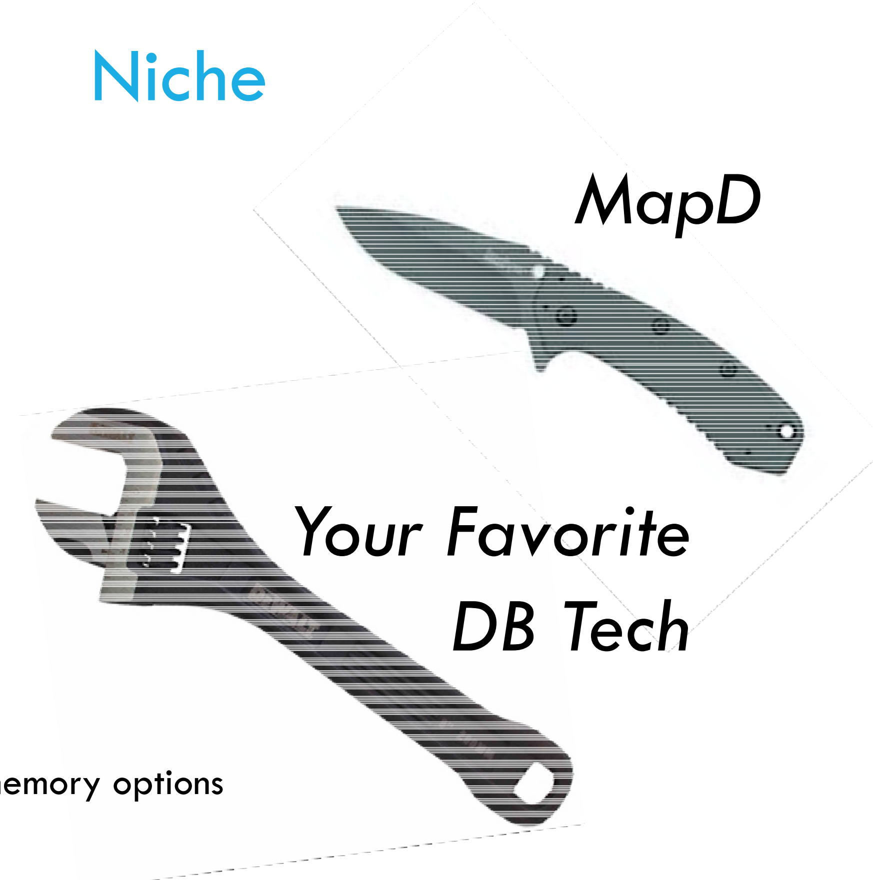


WHAT WE LEARNED, BLOB EDITION: IF YOU HAVE A GOOD IDEA, IT WILL BE INGESTED BY THE BIG 3…

Big 3 (Oracle, IBM, MS)

All your whitepapers belong to us.

Niche



In-memory: MS Hekaton, IBM DB2 and Oracle (since 11) have in-memory options
GPU optimization: IBM D2, Kinetica + MS Azure

WHAT WE LEARNED, NON-BLOB EDITION:

- Innovations often arise from solutions tailored to a specific problem domain

Mapd - PHD candidate wanted to analyze tweets during the Arab Spring

  - Needs: fast, handle very large real-time datasets, excel at visualization
  - Not needed/prioritized: persistence
  - Solutions: in-memory, GPU optimization

- No solution is strictly better than another, there are only tradeoffs
- Use of GPU allows optimization/ implementation of algorithms that benefit from the parallel nature of the processors (i.e. bucket select)
- In addition to cost-based query optimizations, there are hardware and other low-level optimizations available (kernel fusion, loop fusion, pipeline push, etc.)