# Introduction

## What Is a GPU Database?

A GPU database uses GPU (Graphics Processing Unit) to perform database operations. It is different from traditional databases (such as MySQL, MS SQL Server, etc.) which use CPU for these tasks.

GPU databases use **standard drivers** and **SQL** to query data. Their deployments can be on-premise or run in the cloud.

## GPU Database vs CPU Database

First let's take a look at main differences between CPU and GPU:

|  |  |
| --- | --- |
| **CPU** | **GPU** |
|  |  |
| Sequential series processing with multiple cores | Parallel processing with thousands of small cores |
| Has large board instruction set, manages every input/output of a computer | Has specific instruction set, only work for its functions |
| CPU core is fast and smart | GPU core is slow and not full feature |
| Best suited for general-purpose serial applications | Best suited for repetitive and highly-parallel computing tasks |
| Consists of:   * Control unit block * ALU block * Cache memory | Consists of:   * PF unit * INT unit |

GPU databases use the **parallelism of GPUs** to perform massive data-processing acceleration. The GPU is ideally suited to accelerate processing SQL queries because SQL performs the same operation (usually a search) on every row in the set.

**Why GPU DB?**

The GPU DB offers various benefits:

* 10x-100x **faster** than CPUs when processing the same workloads. Thus, can deliver SQL queries across billions of records in milliseconds. Ideal for Big Data.
* Much **smaller** (6.5x – 20x smaller than a CPU). Just 16 GPU-accelerated servers could perform as well as a 1000 CPU cluster
* Easier to work with extremely large data sets or extremely fast data streams (from sources such as the Internet of Things, clickstreams and business transactions).
* Easier to **scale** because a GPU database requires adding more GPUs to a server rather than adding more servers.

**Why CPU DB?**

* Doing JOINs: [For GPU Databases of today, the big challenge is doing JOINS (kdnuggets.com)](https://www.kdnuggets.com/2018/03/brytlyt-gpu-databases-joins.html)

# GPU Database Architecture

### Models

#### GPU As an Accelerated Device (Both CPU-GPU)



In GPU-accelerated server model, the system combinates CPU and GPU. CPU handles all input and output, while GPU is used only as bulk-synchronous high-performance accelerators for data computing.

#### GPU As a Control Server (only GPU)



In GPU control server model, all data is sent directly to GPU without the presence of any CPU. It means the GPU is not only used for for data computing but also for I/O tasks.

# Common GPU Database Engines

[OmniSci vs. SQream DB Comparison (db-engines.com)](https://db-engines.com/en/system/OmniSci%3BSQream+DB)

GPU databases are wholly a startup phenomenon, with companies such as [Brytlyt](https://www.brytlyt.com/), [SQream Technologies](https://sqream.com/), [OmniSci](https://www.mapd.com/), [Kinetica](https://www.kinetica.com/), [PG-Strom](https://wiki.postgresql.org/wiki/PGStrom), and [Blazegraph](https://www.blazegraph.com/).

All vary slightly in how they work. For example, OmniSci does visualization of data, while SQream uses connectors to visualization tools like Tableau, so each needs to be individually evaluated to determine the best fit for your need.

This chart below should help you understand which of these GPU database is right for you:

* [SQream DB](https://laptrinhx.com/link/?l=http%3A%2F%2Fwww.sqream.com)
* [MapD](https://laptrinhx.com/link/?l=http%3A%2F%2Fwww.mapd.com)
* [Kinetica](https://laptrinhx.com/link/?l=http%3A%2F%2Fwww.kinetica.com)
* [PG-Strom](https://laptrinhx.com/link/?l=http%3A%2F%2Fstrom.kaigai.gr.jp%2F)
* [Blazegraph](https://laptrinhx.com/link/?l=https%3A%2F%2Fwww.blazegraph.com%2F)

# OmniSciDB

Formerly MapD

Official website: <https://www.omnisci.com/platform/omniscidb>

## Features

**Open-Source Code**

OmniSciDB is an open-source SQL engine and [available on GitHub](https://github.com/mapd/mapd-core) under the Apache 2.0 license.

**APIs**

CLI (via omnisql), Java (via JDBC), C/C++ (via ODBC), Thrift, Python (via pymapd), VGA, R (via RJDBC).

Refs: <https://docs-new.omnisci.com/apis-and-interfaces>

**Advanced Memory Management**

OmniSciDB keeps hot data in GPU memory for the fastest access possible. Other [GPU database](https://www.omnisci.com/technical-glossary/gpu-database) systems store the data in CPU memory, only moving it to GPU at query time, trading the gains they receive from GPU parallelism with transfer overheads over the PCIe bus.

OmniSciDB avoids this inefficiency by **caching recently touched data in High Bandwidth Memory on the GPU**, which offers up to 10x the bandwidth of CPU DRAM and far lower latency.

OmniSciDB is also designed to exploit efficient inter-GPU communication infrastructure such as NVIDIA NVLink when available.

**Native SQL Engine**

OmniSciDB **natively supports industry-standard SQL**. Thus, users can reuse their existing SQL querying data.

Besides, it can operate as a standalone SQL engine using the command line tool [mapdql](https://docs.omnisci.com/v4.1.0/3_mapdql.html), or the SQL editor (which is part of the OmniSci Immerse visual analytics interface).

**JIT Query Compilation**

OmniSciDB takes advantage of **the JIT (Just-In-Time) compilation framework** built on LLVM (Low-level Virtual Machine). By pre-generating compiled code for the query, it avoids many memory bandwidth and cache-space inefficiencies of traditional VMs or transpiler approaches.

Using LLVM, compilation times are much quicker – generally under 30 milliseconds for entirely new SQL queries. Furthermore, the system can cache templated versions of compiled query plans for reuse. This is important in situations where users are leveraging OmniSci Immerse to cross-filter billions of rows over multiple correlated visualizations.

**Hybrid Execution**

OmniSciDB can be run on **hybrid CPU/GPU systems**, as well as on **CPU-only systems** featuring X86, Power, and ARM (experimental support) architectures.

**Distributed Architecture**

**When a query is launched, each GPU processes a slice of data independently from other GPUs**. Even though multiple GPUs reside within a single machine, the data is fanned out from CPU to multiple GPUs and then gathered back together onto the CPU.

A distributed architecture also provides faster data load times. Import times speed up linearly with the number of nodes because loading can be done concurrently across multiple nodes. Reads from disk also benefit from similar acceleration in a scale-out configuration.

## Downloads and Installation

### Requirements

* OS: Linux (Ubuntu or CentOS)
* GPU:

### Installation Methods

One of following ways:

#### From Pre-Built Binaries

Guide: <https://docs.omnisci.com/installation-and-configuration/installation>

#### From Source Code

Guide: <https://omnisci.github.io/omniscidb/>

### Running

1. Start omnisci

Way 1: sudo systemctl start omnisci\_server

Way 2: sudo ./opt/omnisci/startomnisci

The unity omnisci's commands are documented [here](https://docs-new.omnisci.com/apis-and-interfaces/omnisql).

2. Connect to OmniSciDB

/opt/omnisci/bin/omnisql omnisci

password: ••••••••••••••••

Note: The default password is "HyperInteractive"

3. Test connection

Run any valid SQL query. If there is valid return, the connection is established successfully.

For example:

omnisql> SELECT origin\_city AS "Origin", dest\_city AS "Destination", AVG(airtime) AS "Average Airtime" FROM **flights\_2008\_10k** WHERE distance < 175 GROUP BY origin\_city, dest\_city;

The results should be similar to below:

Origin|Destination|Average Airtime

Austin|Houston|33.055556

Norfolk|Baltimore|36.071429

Ft. Myers|Orlando|28.666667

Orlando|Ft. Myers|32.583333

…

## APIs

[OmniSciDB Developer Documentation — OmniSciDB documentation](https://omnisci.github.io/omniscidb/)

### ODBC

ODBC (Open Database Connectivity), produced by Microsoft, is a standard API for accessing DBMS. OmniSciDB supports ODBC connections.

Ref: [ODBC - OmniSci Docs](https://docs-new.omnisci.com/apis-and-interfaces/odbc)

#### Installation

Installing ODBC on Linux: [here](https://docs-new.omnisci.com/apis-and-interfaces/odbc#installing-odbc-on-linux) (Note: username and password are "mapd" and "HyperInteractive" respectively)

Running

1. Configure /etc/odbc.ini as followings:

2. Configure /etc/odbcinst.ini as followings:

Note: Your odbcinst.ini file might be empty or might already contain other entries. If your file contains other entries, add the new entries to the end of the file and do not overwrite existing entries.

Ref: [ODBC - OmniSci Docs](https://docs-new.omnisci.com/apis-and-interfaces/odbc)

Error: [ruby on rails - How to fix the [unixODBC][Driver Manager]Data source name not found, and no default driver specified (ODBC::Error) - Stack Overflow](https://stackoverflow.com/questions/21237678/how-to-fix-the-unixodbcdriver-managerdata-source-name-not-found-and-no-defa)

# Others

CUDA: [Tutorial 01: Say Hello to CUDA - CUDA Tutorial (cuda-tutorial.readthedocs.io)](https://cuda-tutorial.readthedocs.io/en/latest/tutorials/tutorial01/)

OmniSci Big Data Analytics White Paper download: [Technical Analytics White Paper | OmniSci](https://www2.omnisci.com/resources/technical-whitepaper/lp)

How To Import A CSV File Into A MySQL Database? <https://phoenixnap.com/kb/import-csv-file-into-mysql>

Measure query time:

[mysql - Calculating query execution time - Stack Overflow](https://stackoverflow.com/questions/20300136/calculating-query-execution-time)

[How to Measure MySQL Query Time: A Detailed Look | Scalyr](https://www.scalyr.com/blog/how-to-measure-mysql-query-time/)

# Performance Testing

|  |  |  |
| --- | --- | --- |
|  | Omnisci | MySQL |
|  |  |  |