# 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 DB vs CPU DB

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 * ALU * Cache memory * DRAM memory | Consists of:   * PF unit * INT unit * DRAM memory |

**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

### CPU-GPU



*GPU as an accelerated device*

In this model (also called **hybrid computing model**), the system combinates CPU and GPU. While CPU handles all input and output, GPU is used only as bulk-synchronous high-performance accelerators for data computing.

In this model, applications offload these time-consuming routines and functions (also called hotspots) to run on GPUs and take advantage of massive parallelism. The rest of the application still runs on the CPU.

### GPU-only



*GPU as a control server*

In this 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.

## Challenges

Parallel programming with GPU is a profound way to accelerate applications. However, it has some common challenges.

* To simplify parallel programming to make it easy to program. Easy programming attracts more developers and motivates them to port many more applications on parallel processors.
* To develop application that transparently scales its parallelism to leverage the increasing number of processor cores with GPUs.

# 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)

[GitHub - BlazingDB/blazingsql: BlazingSQL is a lightweight, GPU accelerated, SQL engine for Python. Built on RAPIDS cuDF.](https://github.com/BlazingDB/blazingsql)

[GitHub - BenjaminTrapani/gpu-no-sql: A GPU-based NoSQL database that performs GPU-accelerated parallel lookups using CUDA and Thrust](https://github.com/BenjaminTrapani/gpu-no-sql)

# CUDA

## What Is CUDA?

CUDA is a **parallel computing platform and programming model** for general computing on GPUs. With CUDA, you can speed up applications by harnessing the power of GPUs.

NVIDIA released the first version of CUDA in Nov 2006. It's provided with a simple **interface based on C/C++**.

CUDA provides three key language extensions to programmers:

* **CUDA blocks**: A group of threads.
* **Shared memory**: Memory shared within a block among all threads.
* **Synchronization barriers**: Enable multiple threads to wait until all threads have reached a particular point of execution before any thread continues.

## Why Is CUDA?

CUDA is the **best** way for any developer who want to utilize GPU to accelerate their application.

We know that parallel programming is difficult. It has some common challenges.

- To make it easy to program. Easy programming attracts more developers and motivates them to port many more applications on parallel processors.

- To develop application that transparently scales its parallelism to leverage the increasing number of processor cores with GPUs.

With CUDA, any problem or application can be **divided into small independent problems** and solved independently **among CUDA blocks**. Each block solves a sub-problem into finer pieces with **parallel threads** executing and cooperating with each other. The CUDA runtime decides to schedule these blocks on multiprocessors in a GPU in any order. This allows the CUDA program to scale and run on any number of multiprocessors.

## CUDA Ecosystem

### Programming Languages and APIs

CUDA supports multiple programming languages, including C, C++, Fortran, Python, and so on.

With NVIDIA CUDA toolkit, you can develop applications using C/C++. With NVIDIA PGI toolkit, you can develop applications using Fortran. In addition, there are many third-party toolchains:

* [PyCUDA](https://developer.nvidia.com/pycuda): Use CUDA APIs from Python interface.
* [Altimesh Hybridizer](https://developer.nvidia.com/altimesh-hybridizer): Generate CUDA C code from .NET assemblies (MSIL) or Java archives (java bytecode).
* [OpenACC](https://developer.nvidia.com/openacc): Use directives to program GPUs and compiler-generated CUDA code before executing on CUDA GPUs.
* [OpenCL](https://developer.nvidia.com/opencl): Use low-level APIs to program CUDA GPUs.
* [Alea-GPU](https://developer.nvidia.com/alea-gpu): Program CUDA GPUs with the .NET framework.

### Libraries

NVIDIA provides a layer on top of the CUDA platform called [CUDA-X](https://developer.nvidia.com/gpu-accelerated-libraries), , which is a collection of libraries, tools, and technologies. GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning, and graph analytics. For developing custom algorithms, you can use available integrations with commonly used languages and numerical packages, as well as well-published development API operations.

The NVIDIA CUDA toolkit comes with a wide collection of commonly used libraries. Partners also contribute many libraries on the platform:

* **Mathematical libraries:** cuBLAS, cuRAND, cuFFT, cuSPARSE, cuTENSOR, cuSOLVER
* **Parallel algorithm libraries:** nvGRAPH, Thrust
* **Image and video libraries:** nvJPEG, NPP, Optical Flow SDK
* **Communication libraries:** NVSHMEM, NCCL
* **Deep learning libraries:** cuDNN, TensorRT, Jarvis, DALI
* **Partner libraries:** OpenCV, FFmpeg, ArrayFire, MAGMA

### Profiling and Debugging Tools

Here is a preview of CUDA profiling and debugging tools:

* [NVIDIA Nsight](https://developer.nvidia.com/nsight-visual-studio-edition): This is a low overhead profiling, tracing, and debugging tool. It provides a GUI-based environment to scale across a wide range of NVIDIA platforms, such as large multi-GPU x86 servers, Quadro workstations, and so on.
* [CUDA GDB](https://developer.nvidia.com/cuda-gdb): This is an extension of Linux GDB, which provides a console-based debugging interface that you can use from the command line. CUDA GDB can be used on your local system or any remote system. GUI-based plugins are also available, for example,  [DDD](http://www.gnu.org/software/ddd), [EMACS](http://www.gnu.org/software/emacs), or [Nsight Eclipse Edition](https://developer.nvidia.com/nsight-eclipse-edition" \t "_blank).
* [CUDA-Memcheck](https://developer.nvidia.com/cuda-memcheck): A must-have tool that provides insights into memory access issues by examining the thousands of threads running concurrently.

There are also many third-party solutions available, including the following:

* [ARM Forge](https://developer.nvidia.com/allinea-ddt)
* [TotalView Debugger](https://developer.nvidia.com/totalview-debugger)
* [PAPI CUDA Component](https://developer.nvidia.com/papi-cuda-component)
* [TAU Performance System](https://developer.nvidia.com/tau-performance-system)
* [VampirTrace](https://developer.nvidia.com/vampirtrace)

### Datacenter Tools and Cluster Management

NVIDIA GPUs provide massive acceleration to applications and these applications are scaled further to a large number of GPUs. Many scientific applications, like molecular dynamics and quantum chemistry, and also AI applications need a cluster of GPUs to scale out application performance to thousands of GPUs connected by a high-speed network. The modern data centers are built using NVIDIA GPUs and Mellanox high-speed interconnect to scale the applications to massively scale performance.

You need a sophisticated ecosystem to have ease of deployment in datacenters. Enterprises need tools to easily manage and run these dense datacenters. NVIDIA works closely with ecosystem partners to provide developers and DevOps with software tools for every step of the AI and HPC software lifecycle.

Here are some of the efforts that NVIDIA is working on to strengthen this ecosystem:

#### Containers

Containers are the modern way to easily deploy applications. NVIDIA provides all deep learning and HPC containers from [NVIDIA NGC](https://www.nvidia.com/en-us/gpu-cloud/). These containers are tested, maintained, and well-optimized by NVIDIA. NGC also provides a way to host third-party containers. Organizations can also choose to have private container repositories.

#### Scheduling and Orchestration

Scheduling and orchestration is another important aspect of datacenter management and operations. Kubernetes is the modern and popular container-orchestration system for automating application deployment, scaling, and management. Kubernetes on NVIDIA GPUs extends the industry-standard container orchestration platform with GPU acceleration capabilities. Kubernetes provides state-of-the-art support for NVIDIA GPU resource scheduling.

#### Cluster Management Tools

Major standard cluster management tools have support for NVIDIA GPUs. Some examples include Bright Cluster, Ganglia, StackIQ, and Altair PBS Works.

#### Monitoring Tools

NVIDIA also provides a suite of tools called [DCGM](https://developer.nvidia.com/dcgm)for the management and monitoring of GPUs in cluster environments. NVIDIA also exposes an API-based interface to monitor GPUs by NVML APIs.  With help from these tools, the datacenter ops team can continuously perform active health monitoring, comprehensive diagnostics, system alerts, and governance policies including power and clock management. These tools can be used either standalone or integrated with any industry-standard tool suites. You can also build your own tools using NVML API operations.

### GPUs Everywhere

Many years before, NVIDIA decided that every GPU designed at NVIDIA will support CUDA architecture:

* GeForce GPUs for gaming and notebooks
* Quadro GPUs for professional visualization
* Datacenter GPUs
* Tegra for embedded SoCs

A single compute architecture across all product lines with backward compatibility of CUDA makes this platform a developer’s choice. You can access GPUs in laptops and PCs, workstations, servers, and embedded devices—and run the same CUDA code everywhere. Every cloud service provider on the planet powers CUDA-supported GPUs.

## CUDA Applications

There are thousands of applications accelerated by CUDA. The most common is:

* Libraries and frameworks in machine learning and deep learning.
* GPU-accelerated database engines.

## CUDA Programming Model

### Steps

To execute any CUDA program, there are three main steps:

1. Copy the input data from host memory to device memory, also known as host-to-device transfer.
2. Load the GPU program and execute, caching data on-chip for performance.
3. Copy the results from device memory to host memory, also called device-to-host transfer.

*Keywords:*

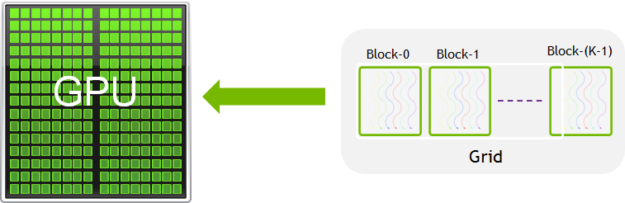
- **Host**: The CPU available in the system. **Host memory**: The memory associated with the CPU.

- **Device**: The GPU. **Device memory**: The memory associated with the GPU.

### Kernel and Thread Hierarchy

CUDA **kernel** is a function that gets executed on GPU. The parallel portion of your applications is executed *K* times in parallel by *K* different CUDA threads, as opposed to only one time like regular C/C++ functions. *.*

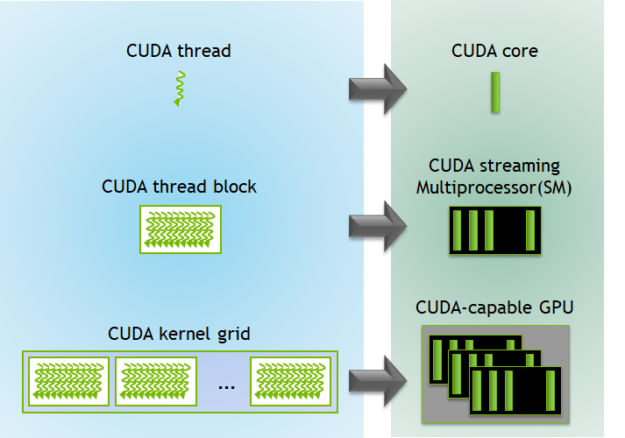
Every CUDA kernel starts with a \_\_global\_\_ declaration specifier. Programmers provide a unique global ID to each thread by using built-in variables.



*CUDA kernels are subdivided into blocks.*

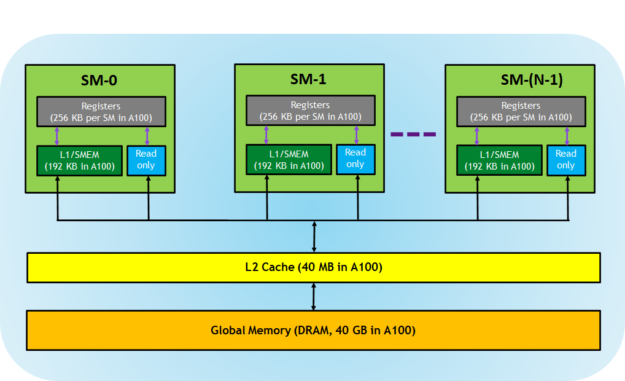
A group of **threads** is called a CUDA **block**. CUDA blocks are grouped into a **grid**. A kernel is executed as a grid of CUDA blocks.

Each CUDA block is executed by one **Streaming Multiprocessor** (SM) and cannot be migrated to other SMs in GPU (except during preemption, debugging, or CUDA dynamic parallelism). One SM can run several concurrent CUDA blocks depending on the resources needed by CUDA blocks. Each kernel is executed on one device and CUDA supports running multiple kernels on a device at one time.



*Kernel execution on GPU.*

### Memory Hierarchy



*Memory hierarchy in GPUs.*

The following memories are exposed by the GPU architecture:

* **Registers**: These are private to each thread, which means that registers assigned to a thread are not visible to other threads. The compiler makes decisions about register utilization.
* **L1/Shared memory (SMEM)**: Every SM has a fast, on-chip scratchpad memory that can be used as L1 cache and shared memory. All threads in a CUDA block can share shared memory, and all CUDA blocks running on a given SM can share the physical memory resource provided by the SM.
* **Read-only memory**: Each SM has an instruction cache, constant memory,  texture memory and RO cache, which is read-only to kernel code.
* **L2 cache**: The L2 cache is shared across all SMs, so every thread in every CUDA block can access this memory. The [NVIDIA A100 GPU](https://www.nvidia.com/en-us/data-center/a100/) has increased the L2 cache size to 40 MB as compared to 6 MB in V100 GPUs.
* **Global memory**: This is the framebuffer size of the GPU and DRAM sitting in the GPU.

## Downloads and Installation

<https://docs.nvidia.com/cuda/index.html#installation-guides>

# OmniSciDB

(Formerly named '[MapD](https://docs.omnisci.com/v3.2.0/)')

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 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 server**

$ sudo systemctl start omnisci\_server

# Another way:

# $ sudo ./opt/omnisci/startomnisci

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

**2. Connect to DB**

To connect to the default DB 'omnisci', run:

/opt/omnisci/bin/omnisql omnisci

password: ••••••••••••••••

# Another way:

/opt/omnisci/bin/omnisql omnisci -p HyperInteractive

Note:

* The default username is "admin" and password is "HyperInteractive". More [details](https://docs.omnisci.com/v5.1.1/5_usersanddatabases.html).
* The defaul TCP ports are 6274 and 6278.

**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

…

**4. Test CPU vs GPU**

Confirm that OmniSci is actually running on GPU or CPU:

omnisql> \cpu

omnisql> EXPLAIN SELECT origin\_city FROM flights\_2008\_10k;

omnisql> \gpu

omnisql> EXPLAIN SELECT origin\_city FROM flights\_2008\_10k;

On CPU mode, you should get something like "IR for the CPU". And on the GPU mode, you should get somethine like "IR for the GPU".

## Coding

Important code to refer:

* The 'omnisql' utility: omniscidb/SQLFrontend/omnisql.cpp
* Unit test and integration test

## 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)

1. ODBC Driver Manager

unixODBC

2. Omnisci ODBC Driver

#### Running

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

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

Note: Your odbcinst.ini file might be empty or already contain other entries. If it contains other entries, append the new entries to the end of the file.

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)

Packages:

unixODBC:

ODBC is produced by Microsoft, and it’s initially used for Windows only. unixODBC is an open-source ODBC driver manager which implements the ODBC APIs for Unix-like platforms (Linux, etc.). It provides the **odbcinst** and **isql** command line utilities used to install / uninstall, configure, and test the driver.

### Thrift

OmniSciDB uses Apache's Thrift framework for all internal communication between the processes within OmniSciDB and external client communication. The full list of thrift API methods can be found in the *omnisci.thrift* file in the root of the OmniSciDB source directory.

#### Installation (Ubuntu)

##### Install Boost library

Thrift needs Boost library to work with.

Run following command:

$ sudo apt-get install libboost-all-dev

# To make sure Boost is installed, run:

# whereis boost

# Typically it's located in the /usr/include/boost

##### Install Other Packages

Thrift needs these packages to work with.

Install runtime libraries for lex and yacc

$ sudo apt-get install -y bison flex

Install libtool:

$ sudo apt-get install -y libtool

Install libssl-dev:

$ sudo apt-get install -y libssl-dev

Install pkg-config autoconf macros (pkg.m4):

$ sudo apt-get install -y pkg-config

##### Thrift

Clone GitHub src: <https://github.com/apache/thrift>

Build and install Thrift: <https://thrift.apache.org/docs/BuildingFromSource.html>

Note: The whold building process can takes about 1-2 hours.

Common errors:

1. While building src with the "make" command, Thrift might not find Boost static libs and following issue occurs:

Cannot find libboost\_unit\_test\_framework.a, libboost\_system.a, libboost\_thread.a, libboost\_filesystem.a, libboost\_chrono.a, etc.

Solution: Create symbolic links to help Thrift finds above libs.

For example:

sudo ln -s /usr/lib/x86\_64-linux-gnu/libboost\_unit\_test\_framework.a /usr/local/lib/libboost\_unit\_test\_framework.a

sudo ln -s /usr/lib/x86\_64-linux-gnu/libboost\_system.a /usr/local/lib/libboost\_system.a

sudo ln -s /usr/lib/x86\_64-linux-gnu/libboost\_thread.a /usr/local/lib/libboost\_thread.a

sudo ln -s /usr/lib/x86\_64-linux-gnu/libboost\_filesystem.a /usr/local/lib/libboost\_filesystem.a

sudo ln -s /usr/lib/x86\_64-linux-gnu/libboost\_chrono.a /usr/local/lib/libboost\_chrono.a

<https://stackoverflow.com/a/42394281>

<https://stackoverflow.com/a/11803653>

#### Generate OmnisciDB C++ Code from Thrift

In OmnisciDB top dir, run:

$ thrift -r --gen cpp omnisci.thrift

This will create a folder named "gen-cpp" with following files:

common\_types.cpp

completion\_hints\_types.h

OmniSci.cpp

omnisci\_types.cpp

serialized\_result\_set\_types.h

common\_types.h

extension\_functions\_types.cpp

OmniSci.h

omnisci\_types.h

completion\_hints\_types.cpp

extension\_functions\_types.h

OmniSci\_server.skeleton.cpp

serialized\_result\_set\_types.cpp

Now, you can write and build C++ code for your OmnisciDB project with Thrift.

Ref: thrift\lib\cpp\README.md

#### C++ Code to Query Data in OmnisciDB

## Others

**Configuration file:**

[Configuration Parameters - OmniSci Docs](https://docs-new.omnisci.com/installation-and-configuration/config-parameters)

**QA:**

[FAQ - OmniSci Docs](https://docs-new.omnisci.com/troubleshooting/faq#confirm-gpus)

**GPU vs CPU performance comparion:**

<https://docs-new.omnisci.com/troubleshooting/faq#compare-performance>

# 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/)

Memory usage: The omnisci server will hold many memory usage, if the memory of the OS reach to limit, stop the server service aand restart.

# Performance Testing

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