GPU Cache Core (GcCore) – Manuel Développeur

Image ?

Image ?

Table des matières

[1- Bibliothèques et dépendances 3](#_Toc14360293)

[Liste des bibliothèques 3](#_Toc14360294)

[Liste des dépendances 3](#_Toc14360295)

[2- Compilation 6](#_Toc14360296)

[3- Implémentation simple 9](#_Toc14360297)

[4- Travaux futurs 10](#_Toc14360298)

# Bibliothèques et dépendances

## Liste des bibliothèques

GcCore est composé de plusieurs bibliothèques, celles-ci sont développées en C++ uniquement ou en CUDA.

Voici la liste des différentes bibliothèques :

* libCommon
  + Elle sert de « standard » à GcCore et elle contient tous les éléments partagés avec les autres bibliothèques. Exemple : le logger, tableaux, LRU, etc.
* libData
  + Elle contient les objets permettant d’utiliser les données. Exemple : lecteur de raw file, les briques, système de chargement des briques, etc.
* libGraphics
  + Elle contient les éléments permettant de faire du rendu / visualisation de données. Exemple : microscope virtuel, lancer de rayon, imgui, etc.
* libMath
  + Elle contient des opérateurs mathématiques basiques. Exemple : vecteurs, etc. Le but est de n’utiliser que ce qui est nécessaire et donc de ne pas recourir à bibliothèques tierces.
* libNetwork
  + Elle contient des éléments permettant des communications réseaux basiques. Exemple : socket tcp et upd, messages, etc. Elle n’existe que pour faire des POC.
* libPreprocessor
  + Elle contient les éléments permettant de faire du préprocessing sur les données. Exemple : mipmapping, bricking, etc.
* libPython
  + Elle permet de faire le lien entre le language python et c++. Elle ne doit jamais être compilé en utilisant NVCC.
* libTinyXML
  + Elle contient les éléments de la bibliothèque TinyXML.

Toutes ces bibliothèques sont compilées avec un compilateur c++ seulement. Certaines d’entre elle disposent aussi d’une bibliothèque paire CUDA. En voici la liste :

* cuda/libcudaCommon
* cuda/libcudaGPUCache
  + Elle contient le cœur de GcCore et tous les caches permettant la gestion out-of-core d’un volume.
* cuda/libcudaGraphics
* cuda/libcudaPreprocessor

## Liste des dépendances

GcCore à quatres dépendances externes, celles-ci sont :

* SDL2
* GLEW
* NvPipe
* LZ4

A ce jour voici la liste des dépendances entres les différentes bibliothèques. Ne pouvant certifier que ce document soit à jour, il est préférable de regarder les fichiers CMakeList.txt pour voir la liste des dépendances. Toutefois, voici comment sont agencées les bibliothèques au jour où ces lignes sont écrites :

* libMath
  + Aucune dépendance
* libTinyXML
  + Aucune dépendance
* libCommon
  + Aucune dépendance
* libcudaCommon
  + libCommon
  + libMath
* libData
  + libCommon
  + libMath
  + libTinyXML
  + LZ4
* libGraphics
  + libCommon
  + libMath
  + libData
  + SDL2
  + GLEW
* libcudaGraphics
  + libCommon
  + libCudaCommon
  + libMath
  + libData
  + libcudaGPUCache
  + libGraphics
  + libNetwork
  + SDL2
  + GLEW
* libNetwork
  + libCommon
  + libData
  + libMath
* libPreprocessor
  + libCommon
  + libData
  + libMath
* libcudaPreprocessor
  + libCommon
  + libData
  + libMath
* libPython
  + libCommon
  + libData
  + libMath
  + libPreprocessor
  + libcudaPreprocessor
* libcudaGPUCache
  + libCommon
  + libcudaCommon
  + libMath
  + libData

# Compilation

Voici la documentation présente dans le fichier « readme.md » :

***GcCore - Core C++/CUDA libraries for GPU Cache.***

Last version: [Link](https://git.neoxia.com/3DNS/gcCore/-/jobs/artifacts/master/raw/GcCore.tar.gz?job=generate\_package\_job)

**Getting started**

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

**Prerequisites**

First of all, the software you need to have to build the project.

**All**

\* A compute capabilty "compute\_30,sm30" at least (NVidia Kepler).

**Windows**

\* \*\*Visual studio 2015\*\* (at least)

\* \*\*Cuda 8.0\*\* (at least)

**Linux**

* \*\*CMake 3.2\*\* (at least)
* \*\*Cuda 8.0\*\* (at least)
* \*\*GCC 5.x\*\* (depending on the Cuda version)

**Mac OS not supported!**

***Building the project***

**Windows**

To build the project you need to go inside the folder **win\_project** and double-clic on the "open\_solution\_smXX.bat" script

It will open the visual studio solution file with all the environment variables set.

The value \*smXX\* depend on your NVidia graphics card.

Then select the configuration (Release / Debug / Static or not) and clic on build.

**Linux**

First of all download and compile the dependencies.

```

cd dependencies

mkdir build

cd build

cmake ../ -DBUILD\_NVPIPE=OFF

make

```

Then go back to the **root** and build the project.

```

mkdir build

cd build

cmake ../src -DCMAKE\_BUILD\_TYPE=Debug -DSHARED=ON -DCMAKE\_CUDA\_FLAGS="-arch=sm\_30"

make all

```

* \*\*Debug\*\* add the option -g
* \*\*Release\*\* add the option -O3.
* \*\*Shared\*\* if \*ON\* GcCore will be built as the shared library, if not set or \*OFF\*, it will be built as static library.
* CMAKE\_CUDA\_FLAGS the cuda flags you want to use.

**Compute capabilities**

Depending of you graphics card you may use differents compute capabilities. See [CUDA wikipedia](https://en.wikipedia.org/wiki/CUDA#GPUs\_supported) to termine the one you can use !

To change it on \*\*Windows\*\*, edit the "open\_solution\_smXX.bat" file and change the line

```

SET CUDA\_COMPUTE\_CAPABILITY=compute\_XX,sm\_XX

```

by the expected compute capabilities.

For Linux, change in the "src/CMakeLists.txt" file.

**Did it work ?**

A "root/bin" folder should have been created with the target build folder (debug / release) containing all the binaries built.

***Include in a project***

**C++ project**

To include GcCore in a C++ project you need to copy the folder "include/" to get all the header files and the folder "bin/" to get the binaries.

**Python project**

To include GcCore in a Python project you need first to build the project as shared library then you have to copy the folder "bin/" where all the binaries are in your python project. Finally, you can load them in your python script.

**Documentation**

You can find the doxygen documentation here : [Link](./doc/doxygen/html/index.html)

# Implémentation simple

En s’assurant que le volume à utiliser soit prêt, voici un exemple d’utilisation du système de cache.

tdns::data::VolumeConfiguration volumeconfig;

//remplir volumeconfig en spécifiant tous les LOD

tdns::data::CacheConfiguration cacheconfig;

//remplir cacheconfig en spéfiant la taille des caches et la tailles de blocks contenus dans les caches.

//ce sont des tableaux à remplir et l’élément 0 est toujours le dataCache.

//instanciation du cache

tdns::gpucache::CacheManager<TYPE> cache(volumeconfig, cacheconfig);

//..

//...

//..

//puis utilisation dans une boucle

while(true)

{

myfunction(cache);

cache.update();

}

//----------------------------------------------

void myfunction(tdns::gpucache::CacheManager<TYPE> &cache)

{

mykernel<1, 1>(cache.to\_kernel\_object());

}

//utilisation du cache

\_\_global\_\_ mykernel(tdns::gpucache::CacheManager<TYPE> cache)

{

float3 position = make\_float3(0.f, 0.f, 0.f);

uint32\_t level = 0;

tdns::gpucache::VoxelStatus status = manager.get(level, position, short\_voxel);

if(status == tdns::gpucache::VoxelStatus::Mapped)

{ //donnée présente }

}

# Travaux futurs

* Prendre en compte le cache flag de la structure « cacheConfiguration » dans le cache manager.
* Enlever la partie libGraphics du dépôt GcCore. En effet, cette partie est plus applicative que réellement liée à GcCore et à la gestion de données de manière out-of-core.