GEANT-4 GPU Port:

User Manual

Stuart Douglas – dougls2
Matthew Pagnan – pagnanmm
Rob Gorrie – gorrierw
Victor Reginato – reginavp

Version 0
February 28, 2016

Contents

1 Revision History

All major edits to this document will be recorded in the table below.

Table 1: Revision History

Description of Changes	Author	Date
Initial draft of document	Matt	2016-02-26

2 List of Figures

Table #	Title

3 Definitions and Acronyms

4 Introduction

4.1 Purpose

4.2 Scope

4.3 Background

4.4 Document Overview

5 Legal Information

6 Installation

6.1 section about that they should know how to use a command line etc., couldn't think of the right word for the sectiontitle

6.2 Required Hardware

6.3 Required Software

6.4 Supported Operating Systems

6.5 Installation Instructions

7 Execution

Geant4 is not an executable program in the traditional sense. It is instead a large set of libraries, designed to work together and that give you, the user, a framework to develop simulations for your work. The development of such simulations is not within the scope of this manual, but it is well-documented with a thorough guide available at `http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/index.html`.

This section will instead outline how one would go about using (or not using) CUDA computations with an existing simulation. The `Hadr04` example that comes with Geant4 will be used as the simulation to demonstrate this.

7.1 Enabling/Disabling CUDA

The installation guide details how to build the Geant4 libraries that will be used by your simulation. The Cmake command from step **??** includes a flag that determines whether or not CUDA will be used – `-DGEANT4_ENABLE_CUDA=ON`. Changing this value to `OFF` will cause Geant4 to build without any CUDA features, and without requiring `nvcc`, the CUDA compiler. After step **??** finishes, users will have to rebuild Geant4 by executing the `make install` command from their `geant4.10.02-build` directory. To re-enable CUDA, simply change the value to `ON` and rebuild again.

7.2 Building the Simulation

After Geant4 has been built, building the simulation is very straightforward. First, navigate to your simulation's root folder (`geant4.10.02/examples/hadronic/Hadr04` in our case) in your terminal. Then, create a new directory `bin`, so your Hadr04 folder now has two folders, `bin` and `src`. Go into the `bin` folder and run `cmake ../src` followed by `make`. That's it!

7.3 Running the Simulation

There are two main methods of running a Geant4 simulation. The first is to simply run the executable, in our case by running `./Hadr04` in the Hadr04 directory. This launches an interactive Geant4 command prompt, and you can manipulate all aspects of the simulation from within it, including the types of particles, their number, and their energies. This also allows you to visualize the simulation using the `vis` commands.

The alternative is to create a macro file that contains a sequence of commands that would be manually given to the interactive prompt, making it much easier to run a given simulation more than once and without requiring user interaction. Both methods print the results of the simulation out to the terminal window, as well as the running time of the simulation. If you wish to save the results to a file, that can easily be done via redirection in Unix. For example, if you wish to save the results of your simulation using `myMacro.mac` to `results.txt`, simply run `./Hadr04 myMacro.mac > results.txt`.

8 Porting Other Geant4 Modules to CUDA

There is currently one class in Geant4-GPU that uses CUDA – G4ParticleHPVector. This class was originally chosen as it was thought to be well-suited to parallelism. Although certain functions within the class are, the majority of the time spent within the class is not in these functions, the main reason for the performance degradations when CUDA is enabled.

There is indeed the possibility that there are other classes within Geant4 that are better suited to parallelism. This section will outline the general methodology we used to port G4ParticleHPVector with the hopes that it may be of use to future developers or users.

8.1 Layout of Source Files

An important consideration during development was to keep the CUDA code separate from the main Geant4 codebase as much as possible. We created a folder named `cuda` in `geant4.10.02/source/externals` to contain all CUDA source code. The `cuda` folder contains two folders, `include` and `src`. All header files are in `include`, and the CUDA source files are in `src`.

8.2 CMake Changes

Geant4 uses CMake as its build system, and minor modifications need to be made to integrate with CUDA. This consists of adding a variable at the top level CMakeLists file to enable or disable CUDA, and creating a Geant4_Add_Library_Cuda macro to Geant4MacroLibraryTargets. This uses `cuda_add_library` function included in Cmake. More details can be found in the project's detailed design document.

8.3 Interfacing with Existing Code

We used compiler directives within the default G4ParticleHPVector.cc file to make use of the CUDA code or to use the original implementation based on the flag passed to Geant4 during the build phase. If the flag is true, an object of type G4ParticleHPVector_CUDA is initialized and all function calls become one line calls to the corresponding function on the G4ParticleHPVector_CUDA object. For example, the `GetY` function looks like the following:

```
G4double GetY(int i) {
        #if GEANT4_ENABLE_CUDA
                return cudaVector->GetY();
        #else
                < existing code from G4ParticleHPVector.cc >
        #endif
}
```

8.4 Naming Conventions

It is important to use naming conventions to ensure clarity within a project that contains many possible ambiguities. The two main conventions we used were adding a "_CUDA" suffix to all source files within the `cuda` directory as well as to the class name of any classes implemented in CUDA. In addition to this, within all CUDA files there

are two main types of pointer – those to GPU memory and those to CPU memory. To distinguish between these, all pointers to main memory are prepended with "h_" and pointers to GPU memory with a "d_".

9 Troubleshooting

9.1 Installation

9.2 Running the Simulation

9.3 FAQ

10 Appendix

10.1 Recommendations for Integration of Geant4 and CUDA

10.2 Future of Geant4-GPU