GEANT-4 GPU Port:

Design Document: System Architecture

Team 8
Stuart Douglas – dougls2
Matthew Pagnan – pagnanmm
Rob Gorrie – gorrierw
Victor Reginato – reginavp

System Architecture: Version 0
February 8, 2016

Table of Contents

1 Introduction

1.1 Revision History

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

Table 1: Revision History

Description of Changes	Author	Date
Set up sections and filled out Introduction section	Matthew	2015-12-15
Document structure adjustments	Stuart	2016-1-8
Filled out Likely/Unlikely Changes section	Rob	2016-1-9

1.2 Terms Used

Table 2: Glossary

Term	Description
Geant4	open-source software toolkit for simulating particle interactions
G4-STORK	fork of Geant4 developed by McMaster's Engineering Physics department to simulate McMaster's nuclear reactor
GPU	graphics processing unit, well-suited to parallel computing tasks
CPU	computer processing unit, well-suited to serial tasks
CUDA	parallel computing architecture for GPU programming, developed by NVIDIA
CUDA file	text file with .cu extension that includes host code (which runs on the CPU) and device code (which runs on the GPU)
NVIDIA	computer hardware and software company specializing in GPU's
Host	the CPU and its memory, managed from within a CUDA file, sends and receives data to the GPU
Device	the GPU and its memory, managed from the host, performs computations requested by the host

1.3 List of Tables

1.4 List of Figures

Table 3: List of Figures

2 Overview

2.1 Purpose of Project

The purpose of the project is to reduce the computation times of particle simulations in Geant4 by parallelizing and running certain procedures on the GPU. The main stakeholders are members of McMaster's department of Engineering Physics that use Geant4 to simulate particle interactions in McMaster's nuclear reactor.

2.2 Description

The project aims to improve the computation times of Geant4 particle simulations by running certain parallel operations on a GPU. GEANT4-GPU will be a fork of the existing Geant4 system with the additional option for users with compatible hardware to run operations on the GPU for improved performance. This functionality will be available on Mac, Linux and Windows operating systems running on computers with NVIDIA graphics cards that support CUDA.

The design strategy for the project will be based on taking a specific, computationally heavy class from Geant4 and creating a class that fulfills the same interface but that runs on the GPU. This will be repeated for many classes until the project's deadline has been reached. The user will have the option of using the existing classes (running on the CPU) or the new ones (running on the GPU).

2.3 Document Structure & Template

The design documentation for the project is broken into two main documents.

This system architecture document details the system architecture, including an overview of the modules that make up the system, analysis of aspects that are likely and unlikely to change, reasoning behind the high-level decisions, and a table showing how each requirement is addressed in the proposed design.

A separate detailed design document covers the specifics of several key modules in the project. This includes the interface specification and implementation decisions.

3 Important Design Decisions & Reasoning

3.1 GPU Computing for Geant4

Geant4 simulations typically involve simulating the movement of many particles (up to millions). Each particle has its new state (position, energy, etc.) calculated at each time-step independently of the other particles. Each state update requires serially traversing an array of particles and performing calculations on each.

This problem is well-suited to parallelization, as each particle has the same, relatively simple functions applied to it independently. Although not faster in many cases, when massive parallelization is possible, running on the GPU can lead to significant performance improvements. This led the stakeholders to propose using the GPU to do some calculations in Geant4 with the goal of reducing the runtime of the simulations.

3.2 CUDA

The two most popular software libraries for performing general computing tasks on the CPU are CUDA and OpenCL. The decision was made to use CUDA for a variety of reasons. Developed by NVIDIA, CUDA has more extensive documentation than OpenCL, a larger online community, a more consistent interface, integration with CMake (the Geant4 build system), and several researchers have successfully integrated small parts of Geant4 with CUDA.

There are two main disadvantages of CUDA however that are worth discussing. First and foremost, CUDA programs can only run on NVIDIA GPUs. If the user has a

GPU made by a different manufacturer, they will not be able to receive any of the performance benefits of the new product. This was determined to be acceptable as the stakeholders explicitly stated plans to purchase a NVIDIA GPU, and McMaster's GPU servers all use NVIDIA graphics cards. Secondly, while OpenCL is open-source, CUDA is proprietary and owned by NVIDIA (although distributed as freeware). This is not optimal, however it was decided that the advantages of the larger community and consistent interface outweighed the disadvantages of a non-open-source library.

3.3 GPU Integration Approach

When considering how to integrate the GPU computations with the non-ported procedures that run on the CPU there were five main options (option 5 was chosen):

1. Port entire Geant4 toolkit to fully run on the GPU.
2. Port all functions of some classes to run on the GPU, those classes will only be instantiated in GPU memory.
3. Port some functions of some classes to run on the GPU, those classes will be stored only in main memory. Data required by the GPU functions will be passed from the host.
4. Port some functions of some classes to run on the GPU, those classes will be stored in main memory and a copy will be stored in GPU memory. Updates to the state will be applied to both copies.
5. Port some functions of some classes to run on the GPU, those classes will have some data stored in main memory and some (different) data stored in GPU memory. If data is required by a function running on the CPU that is stored on the GPU (or vice versa), a copy of the data will be received through a getter function.

Table 4: Comparison of GPU Integration Approaches

Option	Pros	Cons
1	• largest speedup as all aspects of system would be parallelized	• far beyond scope of project and resources available
2	• no memory usage penalty as class only instantiated once • no performance penalty for passing data from main memory to GPU memory	• difficult to use existing code if GPU computation disabled • requires porting all functions of class to GPU even if functions won't give large performance benefits • smaller memory available for GPU may mean that less particles can be simulated at once
3	• no memory usage penalty as class only instantiated once • easily interfaces with existing codebase • easy to use existing code if GPU computation disabled • doesn't require porting functions to GPU that will not have large performance benefits	• all data must be passed from main memory to GPU memory every function call
4	• no performance penalty for passing data between main memory and GPU memory • easily interfaces with existing codebase • easy to use existing code if GPU computation disabled • doesn't require porting functions to GPU that will not have large performance benefits	• double the memory usage • smaller memory available for GPU may mean that less particles can be simulated at once • every time state updated it must be updated twice
5	• reduced performance penalty for passing data between main memory and GPU memory • easy to use existing code if GPU computation disabled • easily interfaces with existing codebase • doesn't require porting functions to GPU that will not have large performance benefits	• performance penalty if data required by CPU from GPU memory (or vice versa)

In all options except 1 and 2, a function call to one of the functions of a ported class will be received by the existing C++ class, which will then either execute the existing code if that function has not been ported or if GPU computation is disabled, or will call the corresponding function from the CUDA file.

Option 5 was chosen due to its ability to divide work easily, its compatibility with the current codebase, and the memory advantages over option 4.

3.4 G4ParticleVector

As was explained in the preceding subsection, the problem has been decomposed to that of only integrating specific functions from given modules with the CUDA technology. With that said, a decision needed to be made as for which class and which functions within that class to integrate.

G4HPNeutronVector was chosen to port to the GPU for two main reasons. The first is that this decision was made to satisfy system and technical requirements, and the second is a matter of satisfying stakeholder requirements.

Geant4 organizes the processes for particles that it must manipulate into separate classes. These classes house functions appropriate to the mechanics surrounding those particles. These classes are the target for parallelization because they are where large arrays of particles and data are stored and manipulated (note: these are not the classes that represent the particles themselves, but the classes that store the functions germane to the given class of particles). These classes are referred to as models in the Geant4 codebase. There are over 100 of these ‘models’ under the hadronic category alone (hadronic particles are those composed of quarks). The problem now is reduced to deciding which model(s) to integrate with the CUDA technology.

The decision as to which model to integrate was made to satisfy stakeholder requirements. The problem at hand was proposed by Dr. Buijs and his graduate students from the Engineering Physics department at McMaster. Dr. Buijs’ team uses Geant4 to model the university’s local nuclear reactor. Their simulations very heavily depend on manipulating neutrons through the G4ParticleVector class. Therefore, this model was chosen for CUDA integration in an attempt to offer the most significant performance increases to the project stakeholders specifically.

4 Likely and Unlikely Changes

Different aspects of the system are presented below in two lists, those that are likely to change in the future and those that are unlikely to change. Relative likelihood of changes in the same category may vary.

Following the lists of likely and unlikely changes, the relationship between likely changes and design components is presented.

4.1 Likely Changes

1. Parallelizing other modules in addition to G4ParticleVector
2. The serial, CPU-bound algorithms used by Geant4 will likely not be suited to running on the GPU
3. Specifics of CMake implementation to compile and link CUDA code

4.2 Unlikely Changes

1. Using CUDA as GPU programming library
2. Using the GPU to run certain computations in parallel
3. The command-line interface of Geant4
4. The use of CMake as the build system
5. Additions of any new process or model modules to Geant4
6. Enabling/disabling CUDA using CMake flag

4.3 Traceability of Likely Changes to Design Components

The following table addresses all likely changes discussed above and their relationship to the design of the system. All numbers in the "Change" column refer to the enumeration from the "Likely Changes" list (section 4.1)

Table 5: Likely Changes and Design Relationship

Change	**Design Component**
1	Deciding which process model to parallelize (see 3.4, paragraph 3)
2	Specific algorithms used will be parallelized versions of the CPU-bound existing ones. See *Algorithms* section of detailed design: G4ParticleVector.
3	How to link and compile the CUDA code is described in more detail in the detailed design document "CMake" section.

5 Module Hierarchy

5.1 Decomposition of Components

The project is based off of Geant4, an existing software system, and modifying specific modules of that system. As such, the decomposition used by the existing system will match the modules of the new system in the project development.

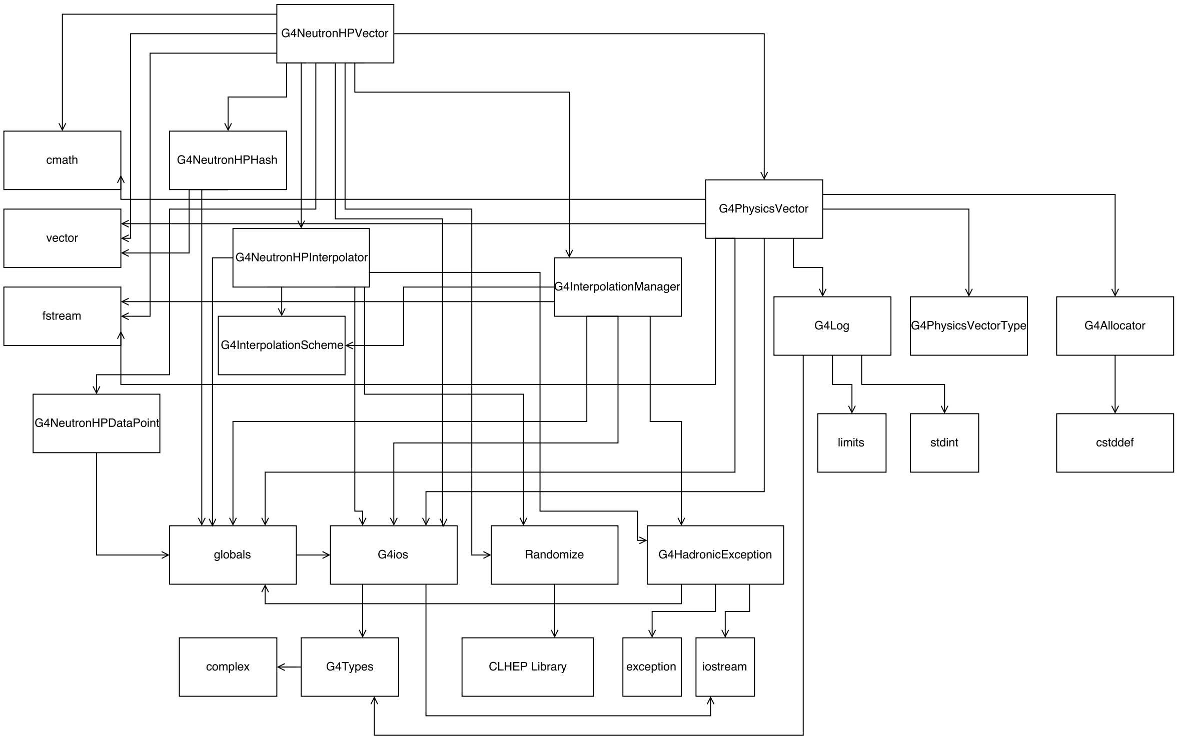
Each module in Geant4 is represented by a C++ class, and the project will port a subset of said classes to the GPU. The decision of which classes to port is derived from performance analysis of the program, with the most computationally time-consuming classes being the targets, such as G4ParticleVector (see section 3.4). Furthermore, it is possible that not all functions of the class will be ported. The focus will be on porting "clean" (having a limited number of side effects), computationally heavy functions. This allows the work to be broken down on a class-by-class and function-by-function basis, and completing the port of one class does not depend on any other classes being ported. If a function running on the GPU requires an external module, the C++ host code for the CUDA that interfaces with the GPU will extract and pass any required information to the GPU and perform any state updates.

5.2 Uses Hierarchy

Since each modified module of the project will have a direct 1-to-1 relationship with an existing module in Geant4, the uses hierarchy with the new, GPU-enabled classes will be identical to the existing uses hierarchy of Geant4 with their corresponding existing classes.

Geant4 is an extremely large system with many modules, so its entire uses hierarchy will not be documented here. Instead, the following hierarchy shows the dependencies of the G4ParticleVector module, which is currently the focus of the project. Note that modules used by system libraries are not included, but the system libraries themselves are included.

Figure 1: Uses Hierarchy for G4ParticleVector



[Hard to read. —DS] [sorry —MP]

6 Traceability of Requirements and Design Components

The following table outlines each requirement and its relationship to the design.

For requirements that do not directly map to sections in the design document (such as providing a repository), the requirement is discussed at a high level in relationship to the design within the context of the entire project.

Table 6: Requirements and Design Relationship

Req.	Description	Design Component
1	computations run on GPU	reasoning behind using GPU explained in section 3.1. Refer to detailed design document for implementation details.

2	existing projects not affected	by retaining existing code and only calling GPU functions if enabled requirement is met. See section 3.3.
3	by default simulation will run on CPU	CMake module description in detailed design document explains how this is implemented.
4	enabling GPU functionality on existing projects easy	A single boolean flag will be set during CMake phase to enable GPU functionality, see CMake module in detailed design document.
5	runtime of simulation decreased with same output	This is the purpose of the project, and is addressed by porting some computationally-heavy classes to the GPU. Although discussed throughout the design document, particularly relevant sections are sections 3.1, 3.3, and the G4ParticleVector module in the detailed design document.
6	accuracy of results same as when run on CPU	Getting the same result whether computation is run on the CPU or GPU is extremely important, and is covered by the internal module design of G4ParticleVector in the detailed design document.
7	at least as stable as existing system	Stability again relies on the implementation of the ported modules, specifically the G4ParticleVector module from the detailed design document
8	errors will throw exceptions	For each module in the detailed design document a section details how errors will be handled, including the throwing of exceptions
9	will support Geant4 10.00.p02 and later	Currently development is based off of Geant4 10.00.p02. Before the project is made public testing will be done with all major newer versions of the software to ensure compatibility.
10	available on public repo with installation instructions	The project is currently available on a private repo, the installation instructions will be added before the repo is set to public.

11	new versions of product will be available on repo, won’t break previous features	All current development is being done with respect to the repository, and changes are all being pushed there. Retaining existing features will be a priority once initial development is complete.
12	all users have access to entire product	This is addressed by the public repository and an upcoming open source license that will be added to the project’s documentation.

[Removed hardware detection requirement – see SRS comment for reasoning —MP]