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
January 9, 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

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

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.

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 performane. 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 based off of WHAT TEMPLATES?????? and 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 moves independently of the other particles. Currently, the particles are all stored in an array that the CPU has to go through moving each particle one by one. Considering how many particles this array holds this is a rather time consuming operation, especially since this movement of the particles is done often. Due to the nature of the particles being able to move independently of each other this is a problem that can be easily parallelized. Since the act of moving a particle is so simple GPU cores are able to execute these types of functions. So instead of parallelizing Geant4 on multiple CPU's we can instead parallelize the code to run on all the GPU cores since the GPU has many more cores than the CPU it is able to run the parallelized code much faster.

3.2 CUDA

NVIDA recently created a toolkit to make creating parallel code on the GPU much easier. Due to how accessible and easy to use CUDA is to parallelize existing code CUDA was chosen to be as the architecture to be used for this project as it would require the least amount of understanding of the existing Geant4 functions to produce speedups.

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.

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 • everytime 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 G4NeutronHPVector

4 Likely and Unlikely Changes

4.1 Likely Changes

1.

4.2 Unlikely Changes

1.

4.3 Traceability of Likely Changes to Design Components

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.

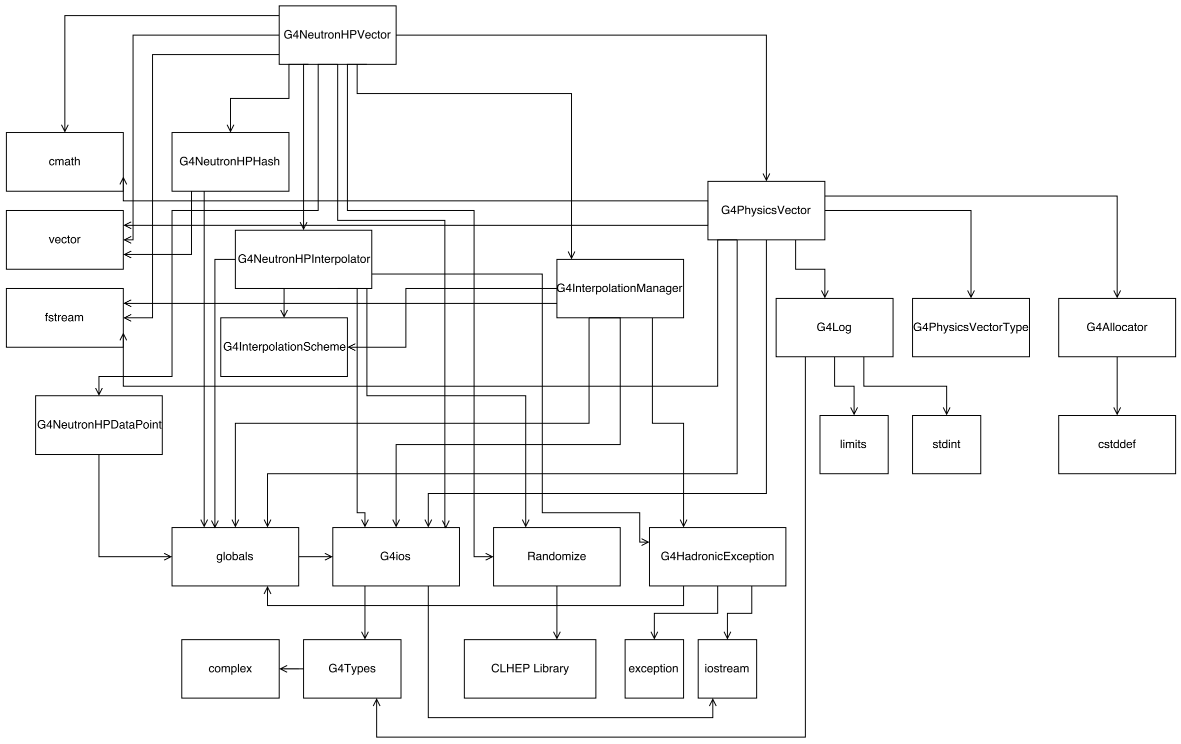
Each module in Geant4 is represented by a C++ class, and the project will port a subset of said classes to the GPU. The decision of which classes to port is derived from performance analysis of the program, with the most computationally time-consuming classes being the targets, such as G4NeutronHPVector (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 G4NeutronHPVector 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 G4NeutronHPVector



6 Traceability of Requirements and Design Components

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

Table 3: Requirements and Design Relationship

Req.	**Brief Description**	**Design Component**
1	computations run on GPU	entire document
2	existing projects not affected	
3	by default simulation will run on CPU	
4	should detect if computer has compatible GPU	
5	enabling GPU computation on incompatible hardware not allowed	
6	enabling GPU functionality on existing projects easy	
7	runtime of simulation decreased with same output	
8	accuracy of results same as when run on CPU	
9	at least as stable as existing system	
10	errors will throw exceptions	
11	will support Geant4 10.00.p02 and later	
12	available on public repo with installation instructions	
13	new versions of product will be available on repo, won't break previous features	
14	all users have access to entire product	