GEANT4 GPU Port:

Test Report

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

Version 0
March 21, 2016

Contents

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, Rob, Victor, Stuart	2016-03-18
Template of document	Matt	2016-03-15

List of Tables

Tables for specific unit and system tests have been omitted in order to keep this document readable.

List of Figures

Table 2: Definitions and Acronyms

Term	Description
GEANT4	Open-source software toolkit used to simulate the passage of particles through matter
GEANT4-GPU	GEANT4 with some computations running on the GPU
GPU	Graphics processing unit, well-suited to parallel computing tasks
CPU	Computer processing unit, general computer processor well-suited to serial tasks
CUDA	Parallel computing architecture for general purpose programming on GPU, developed by NVIDIA
Hadr04	Example simulation installed by default with Geant4, used for system tests.

Definitions and Acronyms

1 Introduction

1.1 Purpose of the Document

This document summarizes the testing and test conclusions of GEANT4-GPU. This document uses the implementation outlined in the test plan.

1.2 Scope of the Testing

The implemented tests are designed to give a general yet rigorous assessment of the components involved.

The tests are segregated into two categories: unit tests and system tests. The unit tests test function components of the G4ParticleVector module, and the system tests compare total system differences between CPU (original GEANT4) and GPU implementations. For both categories, performance and correctness are the key concerns.

Neither the unit tests nor the system tests are concerned with the correctness of original GEANT4 runs, as these runs are used as the baseline for the correctness of GEANT4-GPU modules.

A basic knowledge of programming concepts and command-line tools is assumed, as well as familiarity with GEANT4.

1.3 Organization

In Section 4 we provide an introduction to this report. Section 5 describes the test cases which are carried out on each function. Section 6 describes system test cases that were carried out by our team. In section 7 traceability matrices to requirements and modules are documented. Section 8 provides a summary of changes made in response to the testing results.

1.4 Usability Testing

GEANT4-GPU is a back end implementation of already existing GEANT4 modules. Therefore users will not be interacting with is directly. Since there is no direct user interaction with GEANT4-GPU. There are no usability test.

1.5 Robustness Testing

The GEANT4-GPU functions are meant to mimic the already existing GEANT4 functions. Therefore the GEANT4-GPU functions must also mimic the the robustness of the GEANT4 functions. The accuracy section for unit tests has several unit tests designed to test the robustness of the functions.

2 Unit Testing

2.1 Use of Automated Testing

2.1.1 Overview

Our unit testing system is semi-automated. The user runs a program to generate a test results text file, inputting whether or not Geant4 was compiled with CUDA enabled or disabled. Then, they recompile Geant4 in the opposite configuration (i.e. with CUDA enabled if previously disabled, and vice versa) and run the test program again. At this point there will be two test results text files, one for CUDA enabled, and one for CUDA disabled. In addition, two text files containing runtimes of all computationally-intensive functions are produced. After generating the files, a program to analyze the results is run outputting whether each test case passed or failed, and creating an Excel document (.csv) with the running times.

2.1.2 Generating Test Results

`GenerateTestResults` first initializes several G4ParticleHPVector objects from data files included with Geant4 of varying numbers of entries, including the creation of one G4ParticleHPVector with 0 entries. After the vectors have been initialized, the unit-tested methods are tested with a variety of input values. These cover edge cases (i.e. negative index for array, index greater than number of elements etc.) as well as more "normal" cases. The result of each function is then written to the results text file. This can be a single value in the case of "clean" functions that simply return a value, or it could be the state of the G4ParticleHPVector object, that is the array of points stored by that object. For performance reasons, instead of writing out the entire array of points, a hash value is generated from the array and is outputted. The value of the input variable for each function call is also outputted, so the results for specific inputs can be analyzed.

2.1.3 Analyzing Test Results

After the above files are generated, the `AnalyzeTestResults` utility runs through both documents and for each unit test outputted its status. If it failed, then the result from the CPU and from the GPU are both printed out. After the analysis completes, the total number of tests passed is outputted. In addition, `AnalyzeTestResults` will read the files containing runtimes for each function and output them in .csv format to simplify performance analysis.

2.1.4 Note About Random Results

Some of the tests run in `GenerateTestResults` are based off of random numbers, which differ between the CPU and GPU implementations. To counteract this, each of those tests is run multiple times and the result is averaged. When analyzing results for those functions, they are only marked as failed if the difference in the values of the GPU and CPU results are more than a specified tolerance. There are some functions that depend on random numbers that modify the data array. Since a hash is outputted and will differ no matter how small the difference in the values of the array are, before hashing the values are all rounded to a lower precision.

2.2 Definition of Variables Used for Unit Testing

The following are variables that are used for multiple unit tests. Instead of defining them again for each unit test they are defined here only once. Other variables used for specific unit tests will be defined in their respective unit test sections
For all unit tests:

Table 3: General Unit Test Variables

Name	Type	Description
n	G4double	number of entries in the G4ParticleHPVector
r1	G4double	-1.0
r2	G4double	0.0
r3	G4double	0.00051234
r4	G4double	1.5892317
r5	G4double	513.18
vec0	G4ParticleHPVector	0 entries
vec1	G4ParticleHPVector	80 entries
vec2	G4ParticleHPVector	1509 entries
vec3	G4ParticleHPVector	8045 entries
vec4	G4ParticleHPVector	41854 entries
vec5	G4ParticleHPVector	98995 entries
vec6	G4ParticleHPVector	242594 entries

2.3 void Init(istream & aDataFile, G4int total, G4double ux, G4double uy)

Initializes the data in the current vector with `total` data points from `aDataFile`. Each data point is multiplied by factor `ux` for the x-value and `uy` for the y-value.

2.3.1 Test Inputs

Each vector *vec1*, *vec2* ... *vec6* is associated with a data file, which is the input `aDataFile` to the `Init` function. These data files are bundled with Geant4, and include measured data points for a given isotope of an element. `vec0` is not initialized with a data file, as it is meant to be an empty vector to test edge cases. As such, `vec0` is not tested with this function.

Table 4: Unit Tests - `Init`

Test #	Inputs			
	`aDataFile`	`G4int`	`ux`	`uy`
1	Current data file	n	1	1

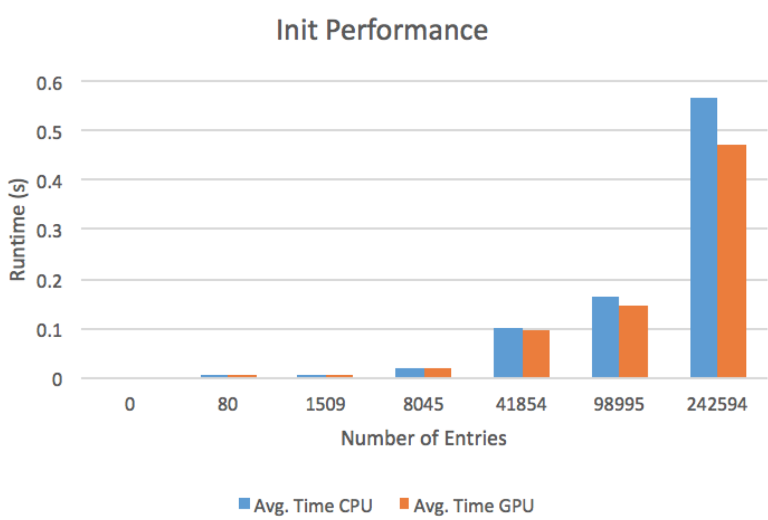
2.3.2 Test Results

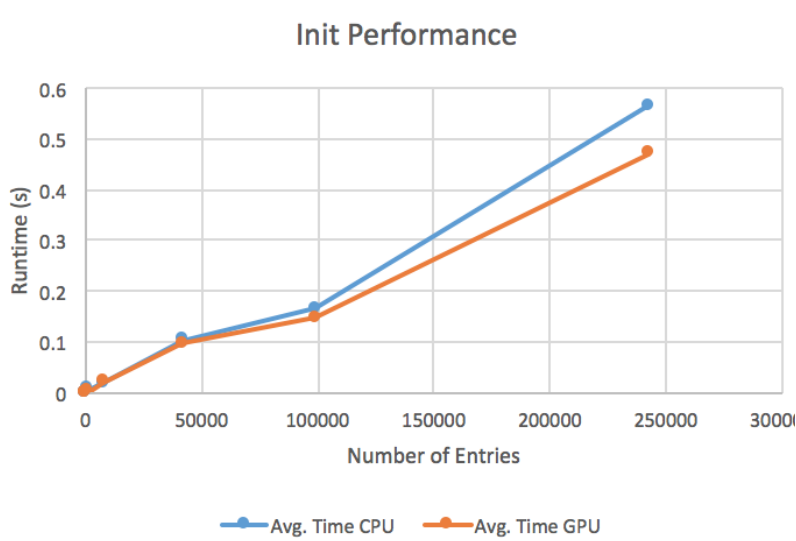
Table 5: Test Results – Init

Test #	Test Result					
	vec1	vec2	vec3	vec4	vec5	vec6
1	Pass	Pass	Pass	Pass	Pass	Pass

2.3.3 Performance

Figure 1: Performance results for `Init` function





2.4 G4ParticleHPVector & operator = (const G4ParticleHPVector & right)

Create a new, temporary G4ParticleHPVector object and assign the current vector to it. Outputs the data and the integral from the new vector.

2.4.1 Test Inputs

Table 6: Unit Tests - `=` (overloaded assignment operator)

Test #	Inputs `right`
2	Current vector

2.4.2 Results

Table 7: Test results - `=` (overloaded assignment operator)

Test #	Test Result vec0	vec1	vec2	vec3	vec4	vec5	vec6
2	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.4.3 Performance

This method is not computationally heavy, so performance data was not included.

2.5 const G4ParticleHPDataPoint GetPoint(G4int i)

Returns the G4ParticleHPDataPoint at index `i` in the current vector. The `x` and `y` values of the point are outputted.

2.5.1 Test Inputs

Table 8: Unit Tests - `GetPoint`

Test #	Inputs `i`
3	-1
4	0
5	n/2
6	n-1
7	n

2.5.2 Test Results

Table 9: Test Results – GetPoint

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
3	Pass	Pass	Pass	Pass	Pass	Pass	Pass
4	Pass	Pass	Pass	Pass	Pass	Pass	Pass
5	Pass	Pass	Pass	Pass	Pass	Pass	Pass
6	Pass	Pass	Pass	Pass	Pass	Pass	Pass
7	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.5.3 Performance

This method is not computationally heavy, so performance data was not included.

2.6 G4double GetX(G4int i)

Returns the energy at index `i` in the current vector. The `x` value of the point are outputted.

2.6.1 Test Inputs

Table 10: Unit Tests - `GetX`

Test #	Inputs `i`
8	-1
9	0
10	n/2
11	n-1
12	n

2.6.2 Test Results

Table 11: Test Results – GetX

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
8	Pass	Pass	Pass	Pass	Pass	Pass	Pass
9	Pass	Pass	Pass	Pass	Pass	Pass	Pass
10	Pass	Pass	Pass	Pass	Pass	Pass	Pass
11	Pass	Pass	Pass	Pass	Pass	Pass	Pass
12	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.6.3 Performance

This method is not computationally heavy, so performance data was not included.

2.7 G4double GetEnergy(G4int i)

Returns the energy at index `i` in the current vector. The `x` value of the point are outputted.

2.7.1 Test Inputs

Table 12: Unit Tests - `GetEnergy`

Test #	Inputs `i`
13	-1
14	0
15	n/2
16	n-1
17	n

2.7.2 Test Results

Table 13: Test Results – GetEnergy

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
13	Pass	Pass	Pass	Pass	Pass	Pass	Pass
14	Pass	Pass	Pass	Pass	Pass	Pass	Pass
15	Pass	Pass	Pass	Pass	Pass	Pass	Pass
16	Pass	Pass	Pass	Pass	Pass	Pass	Pass
17	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.7.3 Performance

This method is not computationally heavy, so performance data was not included.

2.8 G4double GetY(G4int i)

Returns the xSec at index `i` in the current vector. The `y` value of the point are outputted.

2.8.1 Test Inputs

Table 14: Unit Tests - `GetY`

Test #	Inputs `i`
18	-1
19	0
20	n/2
21	n-1
22	n

2.8.2 Test Results

Table 15: Test Results – GetY

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
18	Pass	Pass	Pass	Pass	Pass	Pass	Pass
19	Pass	Pass	Pass	Pass	Pass	Pass	Pass
20	Pass	Pass	Pass	Pass	Pass	Pass	Pass
21	Pass	Pass	Pass	Pass	Pass	Pass	Pass
22	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.8.3 Performance

This method is not computationally heavy, so performance data was not included.

2.9 G4double GetXsec(G4int i)

Returns the xSec at index `i` in the current vector. The `y` value of the point are outputted.

2.9.1 Test Inputs

Table 16: Unit Tests - `GetXsec`

Test #	Inputs `i`
23	-1
24	0
25	n/2
26	n-1
27	n

2.9.2 Test Results

Table 17: Test Results – GetXsec

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
23	Pass	Pass	Pass	Pass	Pass	Pass	Pass
24	Pass	Pass	Pass	Pass	Pass	Pass	Pass
25	Pass	Pass	Pass	Pass	Pass	Pass	Pass
26	Pass	Pass	Pass	Pass	Pass	Pass	Pass
27	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.9.3 Performance

This method is not computationally heavy, so performance data was not included.

2.10 void SetData(G4int i, G4double x, G4double y)

Sets the energy and xSec at index `i` in the current vector.

2.10.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 18: Unit Tests - `SetData`

Test #	Inputs		
	`i`	`x`	`y`
28	-1	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
29	0	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
30	n/2	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
31	n-1	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
32	n	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5

2.10.2 Test Results

Table 19: Test Results – SetData

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
28	Pass	Pass	Pass	Pass	Pass	Pass	Pass
29	Pass	Pass	Pass	Pass	Pass	Pass	Pass
30	Pass	Pass	Pass	Pass	Pass	Pass	Pass
31	Pass	Pass	Pass	Pass	Pass	Pass	Pass
32	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.10.3 Performance

This method is not computationally heavy, so performance data was not included.

2.11 void SetPoint(G4int i, const G4ParticleHPDataPoint & it)

Sets the data point to `it` at index `i` in the current vector.

2.11.1 Test Inputs

- "rPoint" is a G4ParticleHPDataPoint with random values
- "nPoint" is a G4ParticleHPDataPoint with negative values
- "zPoint" is a G4ParticleHPDataPoint with zero values

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 20: Unit Tests

Test #	Inputs	
	`i`	`it`
33	-1	rPoint, nPoint, zPoint
34	0	rPoint, nPoint, zPoint
35	n/2	rPoint, nPoint, zPoint
36	n-1	rPoint, nPoint, zPoint
37	n	rPoint, nPoint, zPoint

2.11.2 Test Results

Table 21: Test Results – SetPoint

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
33	Pass	Pass	Pass	Pass	Pass	Pass	Pass
34	Pass	Pass	Pass	Pass	Pass	Pass	Pass
35	Pass	Pass	Pass	Pass	Pass	Pass	Pass
36	Pass	Pass	Pass	Pass	Pass	Pass	Pass
37	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.11.3 Performance

This method is not computationally heavy, so performance data was not included.

2.12 void SetX(G4int i, G4double e)

Sets the energy at index `i` in the current vector.

2.12.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 22: Unit Tests - `SetX`

Test #	Inputs	
	`i`	`e`
38	-1	r1, r2, r3, r4, r5
39	0	r1, r2, r3, r4, r5
40	n/2	r1, r2, r3, r4, r5
41	n-1	r1, r2, r3, r4, r5
42	n	r1, r2, r3, r4, r5

2.12.2 Test Results

Table 23: Test Results – SetX

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
38	Pass	Pass	Pass	Pass	Pass	Pass	Pass
39	Pass	Pass	Pass	Pass	Pass	Pass	Pass
40	Pass	Pass	Pass	Pass	Pass	Pass	Pass
41	Pass	Pass	Pass	Pass	Pass	Pass	Pass
42	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.12.3 Performance

This function is not computationally heavy, so performance data was not included.

2.13 void SetEnergy(G4int i, G4double e)

Sets the energy at index `i` in the current vector.

2.13.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 24: Unit Tests - `SetEnergy`

Test #	Inputs	
	`i`	`e`
43	-1	r1, r2, r3, r4, r5
44	0	r1, r2, r3, r4, r5
45	n/2	r1, r2, r3, r4, r5
46	n-1	r1, r2, r3, r4, r5
47	n	r1, r2, r3, r4, r5

2.13.2 Test Results

Table 25: Test Results – SetEnergy

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
43	Pass	Pass	Pass	Pass	Pass	Pass	Pass
44	Pass	Pass	Pass	Pass	Pass	Pass	Pass
45	Pass	Pass	Pass	Pass	Pass	Pass	Pass
46	Pass	Pass	Pass	Pass	Pass	Pass	Pass
47	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.13.3 Performance

This method is not computationally heavy, so performance data was not included.

2.14 void SetY(G4int i, G4double e)

Sets the xSec at index `i` in the current vector.

2.14.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 26: Unit Tests - `SetY`

Test #	Inputs	
	`i`	`e`
48	-1	r1, r2, r3, r4, r5
49	0	r1, r2, r3, r4, r5
50	n/2	r1, r2, r3, r4, r5
51	n-1	r1, r2, r3, r4, r5
52	n	r1, r2, r3, r4, r5

2.14.2 Test Results

Table 27: Test Results – SetY

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
48	Pass	Pass	Pass	Pass	Pass	Pass	Pass
49	Pass	Pass	Pass	Pass	Pass	Pass	Pass
50	Pass	Pass	Pass	Pass	Pass	Pass	Pass
51	Pass	Pass	Pass	Pass	Pass	Pass	Pass
52	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.14.3 Performance

This function is not computationally heavy, so performance data was not included.

2.15 void SetXsec(G4int i, G4double e)

Sets the xSec at index `i` in the current vector.

2.15.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 28: Unit Tests - `SetXsec`

Test #	Inputs	
	`i`	`e`
53	-1	r1, r2, r3, r4, r5
54	0	r1, r2, r3, r4, r5
55	n/2	r1, r2, r3, r4, r5
56	n-1	r1, r2, r3, r4, r5
57	n	r1, r2, r3, r4, r5

2.15.2 Test Results

Table 29: Test Results – SetXsec

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
53	Pass	Pass	Pass	Pass	Pass	Pass	Pass
54	Pass	Pass	Pass	Pass	Pass	Pass	Pass
55	Pass	Pass	Pass	Pass	Pass	Pass	Pass
56	Pass	Pass	Pass	Pass	Pass	Pass	Pass
57	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.15.3 Performance

This method is not computationally heavy, so performance data was not included.

2.16 G4double Sample()

Performs samples of the vector according to interpolation its interpolation scheme.

2.16.1 Test Inputs

Table 30: Unit Tests - `Sample`

Test #	Inputs
	`N/A`
58	N/A

2.16.2 Test Results

Table 31: Test Results – Sample

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
58	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.16.3 Performance

This method is not computationally heavy, so performance data was not included.

2.17 G4double SampleLin()

Performs samples of the vector with a linear interpolation scheme.

2.17.1 Test Inputs

Table 32: Unit Tests - `SampleLin`

Test #	Inputs `N/A`
59	N/A

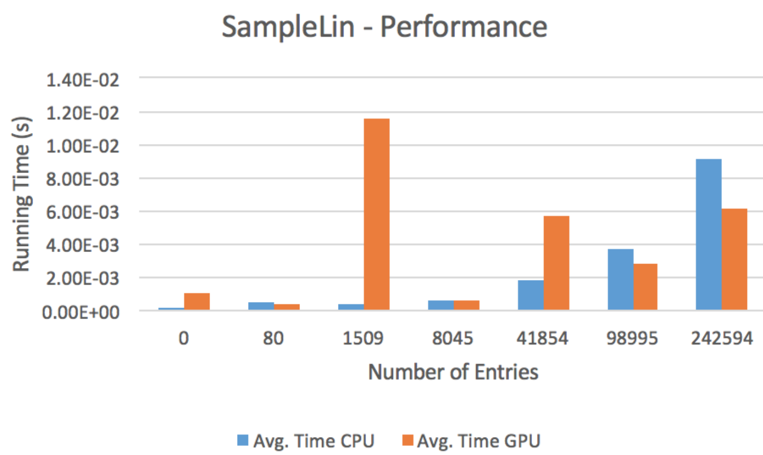
2.17.2 Test Results

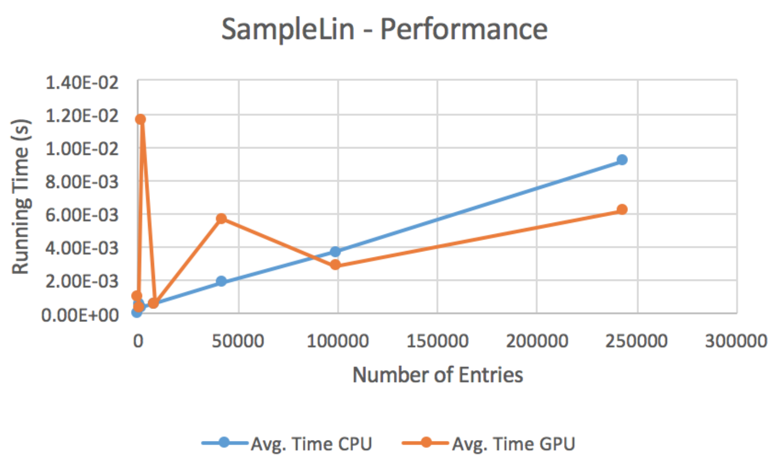
Table 33: Test Results – SampleLin

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
59	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.17.3 Performance

Figure 2: Performance results for `SampleLin`





The extraneous point for 1509 entries where the GPU function is significantly slower can be attributed to the need to copy the data from the GPU memory back to CPU memory in this case.

2.18 void Times(G4double factor)

Multiplies every element in the vector by `factor`.

2.18.1 Test Inputs

Table 34: Unit Tests - `Times`

Test #	Inputs `factor`
60	r1
61	r2
62	r3
63	r4
64	r5

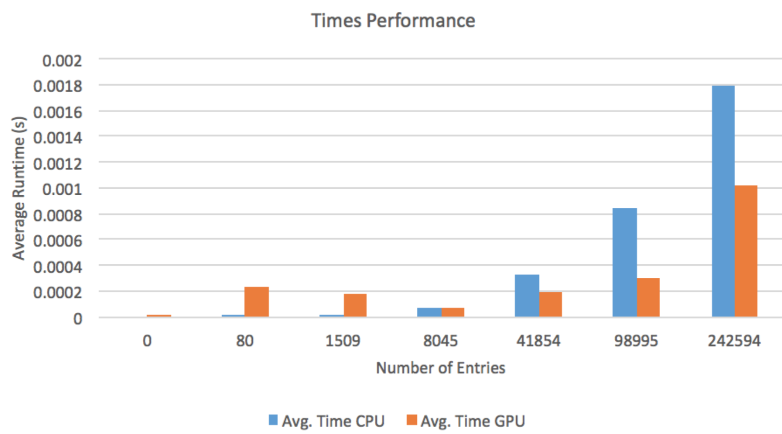
2.18.2 Test Results

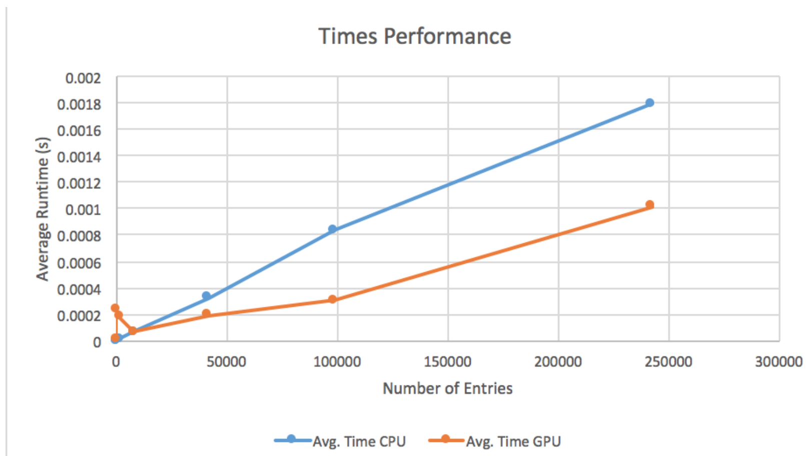
Table 35: Test Results – Times

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
60	Pass	Pass	Pass	Pass	Pass	Pass	Pass
61	Pass	Pass	Pass	Pass	Pass	Pass	Pass
62	Pass	Pass	Pass	Pass	Pass	Pass	Pass
63	Pass	Pass	Pass	Pass	Pass	Pass	Pass
64	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.18.3 Performance

Figure 3: Performance results for `Times` function





2.19 void ThinOut(G4double precision)

Removes any element from the vector whose neighbor is closer than `precision`.

2.19.1 Test Inputs

Table 36: Unit Tests - `ThinOut`

Test #	**Inputs** `factor`
65	r1
66	r2
67	r3
68	r4
69	r5

2.19.2 Test Results

Table 37: Test Results – ThinOut

Test #	**Test Result** vec0	vec1	vec2	vec3	vec4	vec5	vec6
65	Pass	Pass	Pass	Pass	Pass	Pass	Pass
66	Pass	Pass	Pass	Pass	Pass	Pass	Pass
67	Pass	Pass	Pass	Pass	Pass	Pass	Pass
68	Pass	Pass	Pass	Pass	Pass	Pass	Pass
69	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.19.3 Performance

This method is not computationally heavy, so performance data was not included.

2.20 G4double GetXsec(G4double e)

Returns the first xSec from the current vector whose energy is greater than `e`.

2.20.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 38: Unit Tests - `GetXsec`

Test #	**Inputs**
	`e`
70	r1, r2, r3, r4, r5
71	r1, r2, r3, r4, r5
72	r1, r2, r3, r4, r5
73	r1, r2, r3, r4, r5
74	r1, r2, r3, r4, r5

2.20.2 Test Results

Table 39: Test Results – GetXsec

Test #	**Test Result**						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
70	Pass	Pass	Pass	Pass	Pass	Pass	Pass
71	Pass	Pass	Pass	Pass	Pass	Pass	Pass
72	Pass	Pass	Pass	Pass	Pass	Pass	Pass
73	Pass	Pass	Pass	Pass	Pass	Pass	Pass
74	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.20.3 Performance

Figure 4: Performance results for `GetXSec(e)` function

The GPU function is significantly slower. This is due to the lack of a hashing function as on the CPU which dramatically speeds up the time to find the first element in the data with energy at least e. It can be noted in figure 5 that when a minimum value is pre-declared the performance gap is much smaller.

2.21 G4double GetXsec(G4double e, G4int min)

Returns the first xSec from the current vector whose energy is greater than `e`.

2.21.1 Test Inputs

Commas denote multiple sub-test inputs. If one of the sub-tests fail then the whole test fails.

Table 40: Unit Tests - `GetXsec`

Test #	Inputs `e`	`min`
75	r1, r2, r3, r4, r5	-1
76	r1, r2, r3, r4, r5	0
77	r1, r2, r3, r4, r5	n/2
78	r1, r2, r3, r4, r5	n-1
79	r1, r2, r3, r4, r5	n

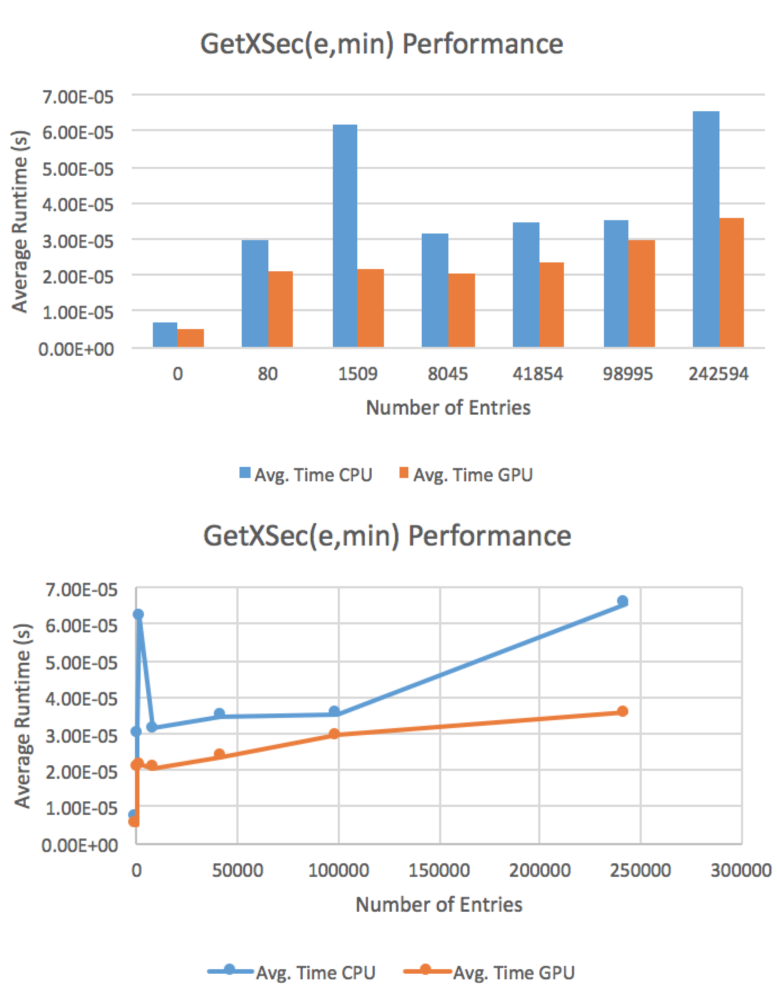
2.21.2 Test Results

Table 41: Test Results – GetXsec

Test #	Test Result vec0	vec1	vec2	vec3	vec4	vec5	vec6
75	Pass	Pass	Pass	Pass	Pass	Pass	Pass
76	Pass	Pass	Pass	Pass	Pass	Pass	Pass
77	Pass	Pass	Pass	Pass	Pass	Pass	Pass
78	Pass	Pass	Pass	Pass	Pass	Pass	Pass
79	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.21.3 Performance

Figure 5: Performance results for `GetXSec(e,min)` function



2.22 G4double Get15percentBorder()

Returns the integral from each data point to the last data point and returns the first one within 15% of the last data point.

2.22.1 Test Inputs

Table 42: Unit Tests - `Get15percentBorder`

Test #	Inputs `N/A`
80	N/A

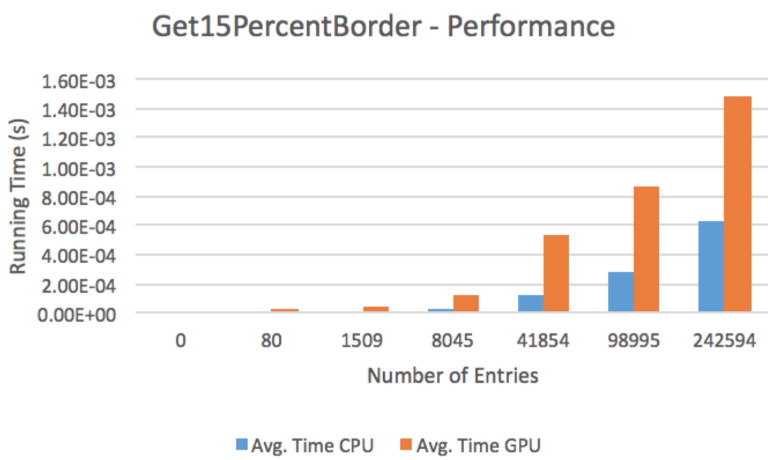
2.22.2 Test Results

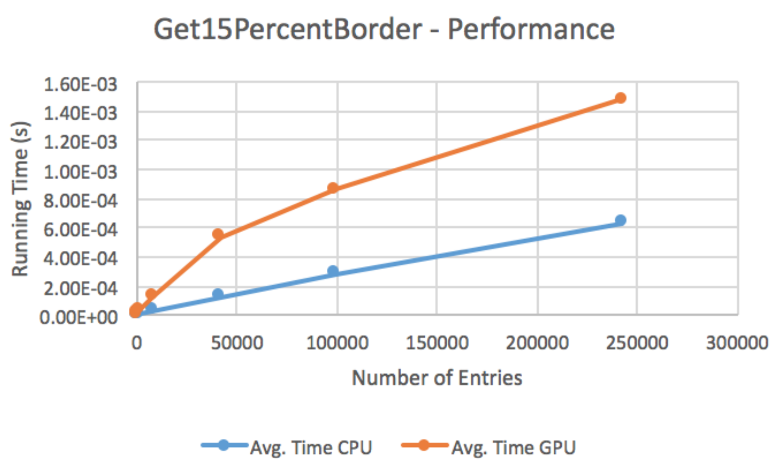
Table 43: Test Results – Get15percentBorder

Test #	Test Result						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
80	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.22.3 Performance

Figure 6: Performance results for `Get15PercentBorder`





2.23 G4double Get50percentBorder()

Returns the integral from each data point to the last data point and returns the first one within 50% of the last data point.

2.23.1 Test Inputs

Table 44: Unit Tests - `Get50percentBorder`

Test #	**Inputs** `N/A`
81	N/A

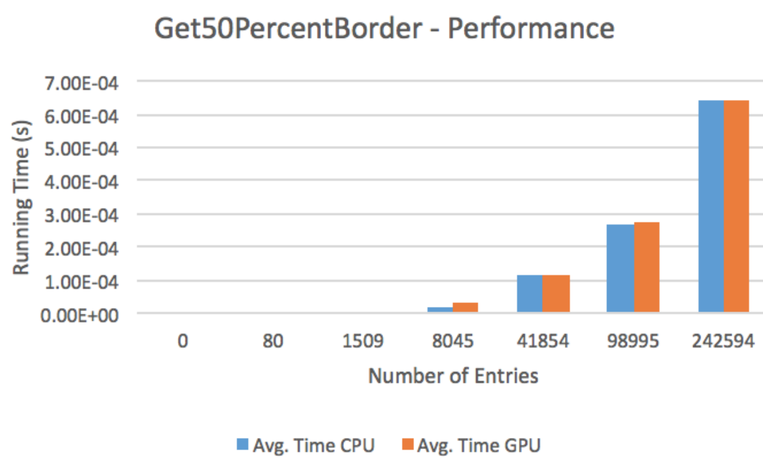
2.23.2 Test Results

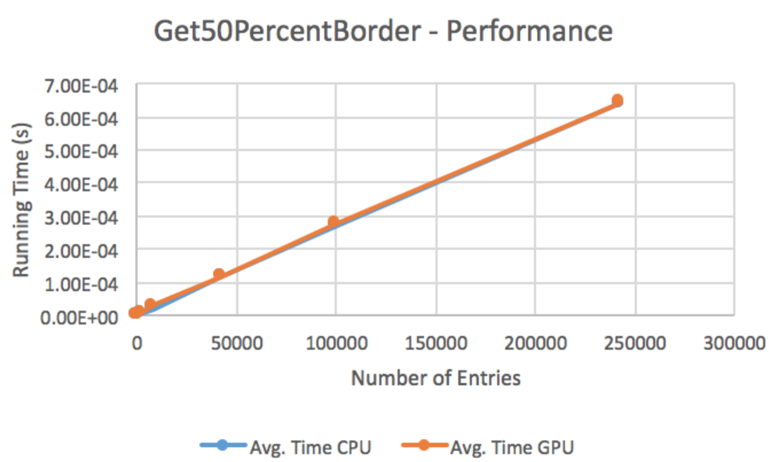
Table 45: Test Results – Get50percentBorder

Test #	**Test Result**						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
81	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.23.3 Performance

Figure 7: Performance results for `Get50PercentBorder` function





3 System Tests

3.1 Summary of System Tests

System tests are performed by running the sample code packaged with the GEANT4 installation. The Hadr04 example will be run with different materials (i.e water, uranium) and varying number of events. The values and conditions that are changed per test are detailed in table 46.

For each system test, a summary of the results is recorded. This consists of a section detailing the accuracy fo the results, and a section covering performance.. The accuracy of the results will be based on the differnece between the values generated on the GPU and the values generated on the CPU. The performance metric that is used is the time required to run each system test.

Table 46: System Tests

Test #	Name	Inputs	Outputs	Description
82	System Test - Water, 2000 events	Events = 2000 Material = Water	Same output as non-GPU GEANT4	Hadr04 no changes
83	System Test - Uranium, 2000 events	Events = 2000 Material = Uranium	Same output as non-GPU GEANT4	Hadr04 – basic example
84	System Test - Water, 600 events	Events = 600 Material = Water	Same output as non-GPU GEANT4	Hadr04 – Shorter test
85	System Test - Uranium, 600 events	Events = 600 Material = Uranium	Same output as non-GPU GEANT4	Hadr04 – Shorter test
86	System Test - Uranium, 20000 events	Events = 20000 Material = Uranium	Same output as non-GPU GEANT4	Hadr04 – Long simulation stress Test

3.2 System Test - Water, 2000 events

This test runs the Hadr04 example on both the GPU and the CPU. The code for the Hadr04 example is bundled with the GEANT4 installation.

3.2.1 Accuracy

Test output for water, 2000 events:

Table 47: Accuracy - Water, 2000 events

Data	CPU Values	GPU Values	Similarity
Process Calls			
hadElastic	423181	432423	far
nCapture	1998	1998	same
neutronInelastic	2	2	same
Parcours of incident neutron			
collisions	212.59	217.21	close
track length	93.387 cm	95.292 cm	close
time of flight	202.14 mus	206.94 mus	close
Generated particles			
C14			
# of particles	2	NA	different
Emean	404.13 keV	NA	different
C15			
# of particles	NA	2	different
Emean	NA	29.472 keV	different
O16			
# of particles	5948	6035	close
Emean	39.577 keV	40.436 keV	close
O17			
# of particles	4	5	close
Emean	1.4823 keV	320.93 eV	far
O18			
# of particles	3	6	close
Emean	52.362	8.5446 keV	far
Alpha			
# of particles	2	2	same
Emean	1.4146 MeV	7.8556 keV	far
Deuteron			
# of particles	1996	1994	close
Emean	1.3185 keV	1.3186 keV	close
Gamma			
# of particles	2000	2005	close
Emean	2.2239 MeV	2.2214 MeV	close
Proton			
# of particles	45355	45645	close
Emean	83.046 keV	82.301 keV	close

3.2.2 Performance

Table 48: Performance - Water, 2000 events

CPU Time	GPU Time	Speedup of GPU
54.55s	72.08s	-1.32×

3.3 System Test - Uranium, 2000 events

This test runs the Hadr04 example on both the GPU and the CPU with a modified source files to include Uranium as an element in the simulation. The number of events for this test has been set to 2000.

3.3.1 Accuracy

Table 49: Accuracy - Uranium, 2000 events

Data	CPU Values	GPU Values	Similarity
Process Calls			
hadElastic	1931	1932	close
nCapture	29	30	close
nFission	281	314	close
neutronInelastic	1690	1656	close
Parcours of incident neutron			
collisions	1.9655	1.966	close
track length	5.6484 cm	5.6517 cm	close
time of flight	2.896 ns	2.8983 ns	close
Generated particles			
U235			
# of particles	29	23	close
Emean	6.5841 keV	7.1217 keV	close
U236			
# of particles	NA	1	different
Emean	NA	10.474 keV	different
U238			
# of particles	3592	3565	close
Emean	8.8059 keV	8.8494 keV	close
U239			
# of particles	29	29	close
Emean	8.3978 keV	8.3269 keV	close
Gamma			
# of particles	4319	4351	close
Emean	496.96 keV	477.71 keV	close
Neutron			
# of particles	2449	2410	close
Emean	1.2746 MeV	1.2814 MeV	close

3.3.2 Performance

Table 50: Performance - Uranium, 2000 events

CPU Time	GPU Time	Speedup of GPU
0.63s	10.57s	-16.78×

3.4 System Test - Water, 600 events

This test simply runs the Hadr04 example on both the GPU and the CPU without changing the source files. The number of runs for this test has been changed to be 600, a low number of events relative to the other trials. The code for this example is bundled with the GEANT4 installation.

3.4.1 Accuracy

Table 51: Accuracy - Water, 600 events

Data	CPU Values	GPU Values	Similarity
Process Calls			
hadElastic	131588	131588	same
nCapture	600	600	same
Parcours of incident neutron			
collisions	220.31	220.31	same
track length	96.495 cm	96.495 cm	same
time of flight	210.57 mus	210.57 mus	same
Generated particles			
O16			
# of particles	1819	1819	same
Emean	41.788 keV	41.788 keV	same
O17			
# of particles	3	3	same
Emean	316.87 eV	316.87 eV	same
O18			
# of particles	1	1	same
Emean	9.3256 eV	9.3256 eV	same
Deuteron			
# of particles	597	597	same
Emean	1.319 keV	1.319 keV	same
Gamma			
# of particles	602	602	same
Emean	2.2229 MeV	2.2229 MeV	same
Proton			
# of particles	13860	13860	same
Emean	81.141 keV	81.141 keV	same

3.4.2 Performance

Table 52: Performance - Water, 600 events

CPU Time	GPU Time	Speedup of GPU
17.07s	22.11s	-1.29×

3.5 System Test - Uranium, 600 events

This test simply runs the Hadr04 example on both the GPU and the CPU with a modified source files to include Uranium as an element is the simulation. The number of events is set to 600, a lower number of events than the other tests, to see how fewer events will impact accuracy and/or performance The code for this example is bundled with the GEANT4 installation.

3.5.1 Accuracy

Table 53: Accuracy - Uranium, 600 events

Data	CPU Values	GPU Values	Similarity
Process Calls			
hadElastic	562	550	close
nCapture	8	9	close
nFission	89	100	close
neutronInelastic	508	491	close
Parcours of incident neutron			
collisions	1.9367	1.9167	close
track length	5.6204 cm	5.4727 cm	close
time of flight	2.8817 ns	2.8063 ns	close
Generated particles			
U235			
# of particles	6	9	close
Emean	6.131 keV	6.1807 keV	close
U238			
# of particles	1064	1032	close
Emean	8.9857 keV	9.0257 keV	close
U239			
# of particles	8	9	close
Emean	8.3308 keV	8.498 keV	close
Gamma			
# of particles	1261	1294	close
Emean	486.58 keV	473.83 keV	close
Neutron			
# of particles	743	737	close
Emean	1.3038 MeV	1.3173 MeV	close

3.5.2 Performance

Table 54: Performance - Uranium, 600 events

CPU Time	GPU Time	Speedup of GPU
0.22s	3.01s	-13.68×

3.6 System Test - Uranium, 20000 events

This test simply runs the Hadr04 example on both the GPU and the CPU with a modified source files to include Uranium as an element is the simulation. The macro file detailing the number of events had been changed such that 20000 events are run. The idea is to see if speed and/or accuracy gaps will widen or shrink as the number of events increases. The code for this example is bundled with the GEANT4 installation.

3.6.1 Accuracy

Table 55: Accuracy - Uranium, 20000 events

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	19526	19335	close
nCapture	245	268	close
nFission	2933	2920	close
neutronInelastic	16822	16812	close
Parcours of incident neutron			
collisions	1.9763	1.9667	close
track length	5.6512 cm	5.5873 cm	close
time of flight	2.898 ns	2.8651 ns	close
Generated particles			
U234			
# of particles	6	4	close
Emean	3.1299 keV	6.32 keV	close
U235			
# of particles	223	193	close
Emean	8.4136 keV	8.4444 keV	close
U236			
# of particles	2	3	close
Emean	9.5669 keV	9.192 keV	close
U238			
# of particles	36119	35950	close
Emean	8.8767 keV	8.8895 keV	close
U239			
# of particles	243	265	close
Emean	8.428 keV	8.4341 keV	close
Gamma			
# of particles	43826	44039	close
Emean	479.54 keV	479.03 keV	close
Proton			
# of particles	24374	24546	close
Emean	1.2749 MeV	1.2587 MeV	close

3.6.2 Performance

Table 56: Performance - Uranium, 20000 events

CPU Time	GPU Time	Speedup on GPU
6.4s	106.41s	-16.64×

4 Traceability

The following section is used to highlight the relations of implemented test cases to requirements and modules. In doing so, we hope to draw clear reasoning upon the inclusion of such tests.

4.1 Requirements

Below is a traceability table outlining test cases and the requirements they are related to:

Table 57: Tests and Requirements Relationship

Test #	**Description**	**Requirement**
1	Performance test of functions	Req. # 4 (Speed and Latency)
2	InitializeVector	Req # 5, 6, 7 (Precision, Reliability, Robustness)
3	SettersandGetters	Req # 5, 6, 7 (Precision, Reliability, Robustness)
4	GetXSec	Req # 5, 6, 7 (Precision, Reliability, Robustness)
5	ThinOut	Req # 5, 6, 7 (Precision, Reliability, Robustness)
6	Merge	Req # 5, 6, 7 (Precision, Reliability, Robustness)
7	Sample	Req # 5, 6, 7 (Precision, Reliability, Robustness)
8	GetBorder	Req # 5, 6, 7 (Precision, Reliability, Robustness)
9	Integral	Req # 5, 6, 7 (Precision, Reliability, Robustness)
10	Times	Req # 5, , 7 (Precision, Reliability, Robustness)

11	Assignment	Req # 5, 6, 7 (Precision, Reliability, Robustness)
12	System Test	Req # 1, 2, 8, 11 (Adjacent Systems, Access)

4.2 Modules

Similarly, the following is a traceability table explicitly relating test cases to modules:

Table 58: Tests and Modules Relationship

Test #	**Description**	**Module**
1	Performance test of functions	G4ParticleVector
2	InitializeVector	G4ParticleVector
3	SettersandGetters	G4ParticleVector
4	GetXSec	G4ParticleVector
5	ThinOut	G4ParticleVector
6	Merge	G4ParticleVector
7	Sample	G4ParticleVector
8	GetBorder	G4ParticleVector
9	Integral	G4ParticleVector
10	Times	G4ParticleVector
11	Assignment	G4ParticleVector
12	System Test	G4NeutronHPDataPoint & G4ParticleVector & CMake Files

5 Changes after Testing

Developing the unit testing system illuminated a variety of bugs and changes that needed to be made. These were predominantly related to edge cases – trying to access indices in arrays that are negative or greater than the number of elements in the array was a common theme. Some of these edge cases were not covered by Geant4 itself, so the required change was made to the original Geant4 source code as well as the modified CUDA code.

Aside from the handling of edge cases with if guards, there was one more significant change required to get the unit tests to pass. An important control flow statement in `GetXSec(e,min)` was supposed to branch if the difference between two values was below a certain threshold. Our implementation was missing a call to get the absolute value for this difference, and as such was returning the wrong result in cases where the second value was larger than the first.

In terms of performance, some performance testing had been done prior to the development of the unit testing system. That profiling data led us to reimplement nearly every function on the GPU using a hybrid approach wherein the data values are stored in both GPU and CPU memory, are modified mainly on the GPU and then the version in CPU memory is updated only when required. This gave very large performance improvements, with the GPU code going from 4.5X slower to 1.2X slower. Further performance tuning is planned for the future based on the results from individual unit tests.