GEANT4 GPU Port:

Test Report

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

Version 0
March 20, 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 tests have been omitted in order to keep this document readable.

List of Figures

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.

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

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 G4ParticleHPVector & operator = (const G4ParticleHPVector & right)

2.3.1 Test Description

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

2.3.2 Test Inputs

Table 4: Unit Tests - `=` (overloaded assignment operator)

Test #	Inputs `right`
1	Current vector

2.3.3 Results

Table 5: Test results - `=` (overloaded assignment operator)

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

2.3.4 Performance

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

2.4 const G4ParticleHPDataPoint GetPoint(G4int i)

2.4.1 Test Description

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

2.4.2 Test Inputs

Table 6: Unit Tests - `GetPoint`

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

2.4.3 Test Results

Table 7: Test Results – GetPoint

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

2.4.4 Performance

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

2.5 G4double GetX(G4int i)

2.5.1 Test Description

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

2.5.2 Test Inputs

Table 8: Unit Tests - `GetX`

Test #	Inputs `i`
7	-1
8	0
9	n/2
10	n-1
11	n

2.5.3 Test Results

Table 9: Test Results – GetX

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

2.5.4 Performance

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

2.6 G4double GetY(G4int i)

2.6.1 Test Description

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

2.6.2 Test Inputs

Table 10: Unit Tests - `GetY`

Test #	Inputs `i`
12	-1
13	0
14	n/2
15	n-1
16	n

2.6.3 Test Results

Table 11: Test Results – GetY

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

2.6.4 Performance

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

2.7 G4double GetXsec(G4int i)

2.7.1 Test Description

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

2.7.2 Test Inputs

Table 12: Unit Tests - `GetXsec`

Test #	Inputs `i`
17	-1
18	0
19	n/2
20	n-1
21	n

2.7.3 Test Results

Table 13: Test Results – GetXsec

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

2.7.4 Performance

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

2.8 G4double GetEnergy(G4int i)

2.8.1 Test Description

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

2.8.2 Test Inputs

Table 14: Unit Tests - `GetEnergy`

Test #	Inputs `i`
22	-1
23	0
24	n/2
25	n-1
26	n

2.8.3 Test Results

Table 15: Test Results – GetEnergy

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

2.8.4 Performance

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

2.9.1 Test Description

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

2.9.2 Test Inputs

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

Table 16: Unit Tests - `SetData`

Test #	Inputs		
	`i`	`x`	`y`
27	-1	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
28	0	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
29	n/2	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
30	n-1	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5
31	n	r1, r2, r3, r4, r5	r1, r2, r3, r4, r5

2.9.3 Test Results

Table 17: Test Results – SetData

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

2.9.4 Performance

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

2.10 void SetEnergy(G4int i, G4double e)

2.10.1 Test Description

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

2.10.2 Test Inputs

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

Table 18: Unit Tests - `SetEnergy`

Test #	Inputs	
	`i`	`e`
32	-1	r1, r2, r3, r4, r5
33	0	r1, r2, r3, r4, r5
34	n/2	r1, r2, r3, r4, r5
35	n-1	r1, r2, r3, r4, r5
36	n	r1, r2, r3, r4, r5

2.10.3 Test Results

Table 19: Test Results – SetEnergy

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

2.10.4 Performance

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

2.11 void SetXsec(G4int i, G4double e)

2.11.1 Test Description

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

2.11.2 Test Inputs

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

Table 20: Unit Tests - `SetXsec`

Test #	Inputs	
	`i`	`e`
37	-1	r1, r2, r3, r4, r5
38	0	r1, r2, r3, r4, r5
39	n/2	r1, r2, r3, r4, r5
40	n-1	r1, r2, r3, r4, r5
41	n	r1, r2, r3, r4, r5

2.11.3 Test Results

Table 21: Test Results – SetXsec

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

2.11.4 Performance

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

2.12 void SetX(G4int i, G4double e)

2.12.1 Test Description

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

2.12.2 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`
42	-1	r1, r2, r3, r4, r5
43	0	r1, r2, r3, r4, r5
44	n/2	r1, r2, r3, r4, r5
45	n-1	r1, r2, r3, r4, r5
46	n	r1, r2, r3, r4, r5

2.12.3 Test Results

Table 23: Test Results – SetX

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

2.12.4 Performance

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

2.13 void SetY(G4int i, G4double e)

2.13.1 Test Description

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

2.13.2 Test Inputs

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

Table 24: Unit Tests - `SetY`

Test #	Inputs	
	`i`	`e`
47	-1	r1, r2, r3, r4, r5
48	0	r1, r2, r3, r4, r5
49	n/2	r1, r2, r3, r4, r5
50	n-1	r1, r2, r3, r4, r5
51	n	r1, r2, r3, r4, r5

2.13.3 Test Results

Table 25: Test Results – SetY

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

2.13.4 Performance

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

2.14 Init

2.14.1 Unit Tests

Table 26: Unit Tests

Test #	Code	Description
52	Empty.Init()	Init an empty Vector
53	D.Init()	Init a Vector

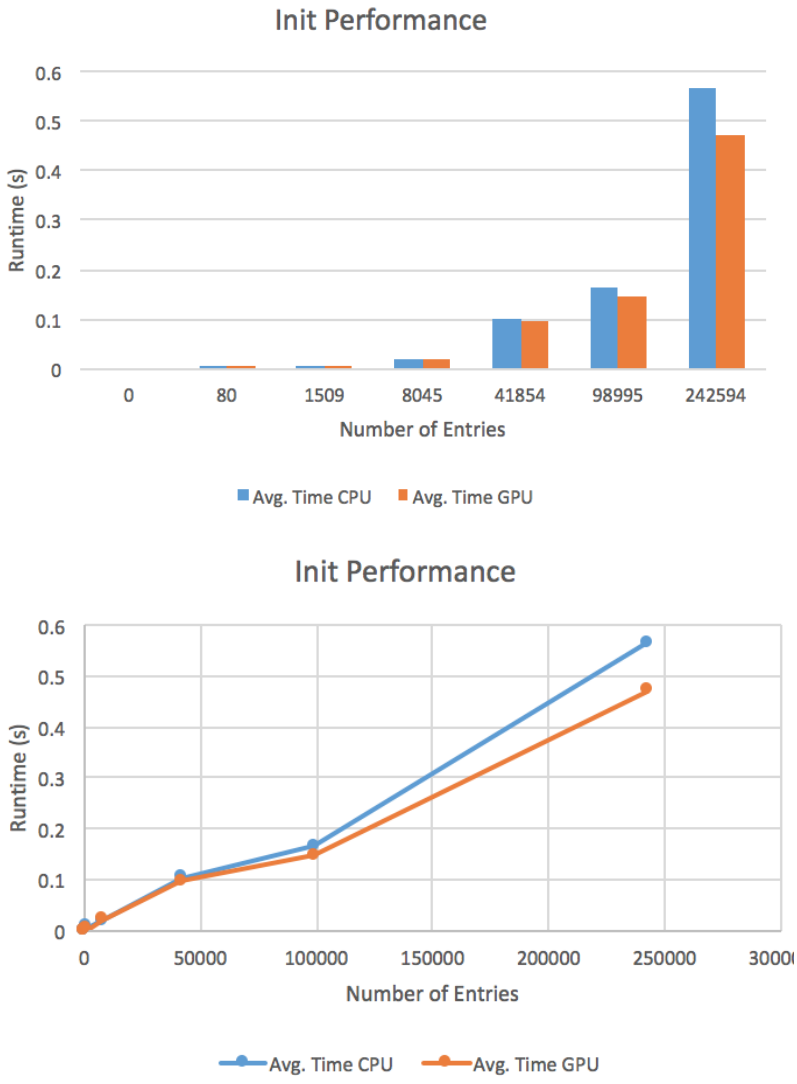
2.14.2 Accuracy

Table 27: Accuracy

Test #	Status
52	Pass
53	Pass

2.14.3 Performance

Figure 1: Performance results for `Init` function



2.15 G4double SampleLin()

2.15.1 Test Description

Performs samples of the vector with a linear interpolation scheme.

2.15.2 Test Inputs

Table 28: Unit Tests - `SampleLin`

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

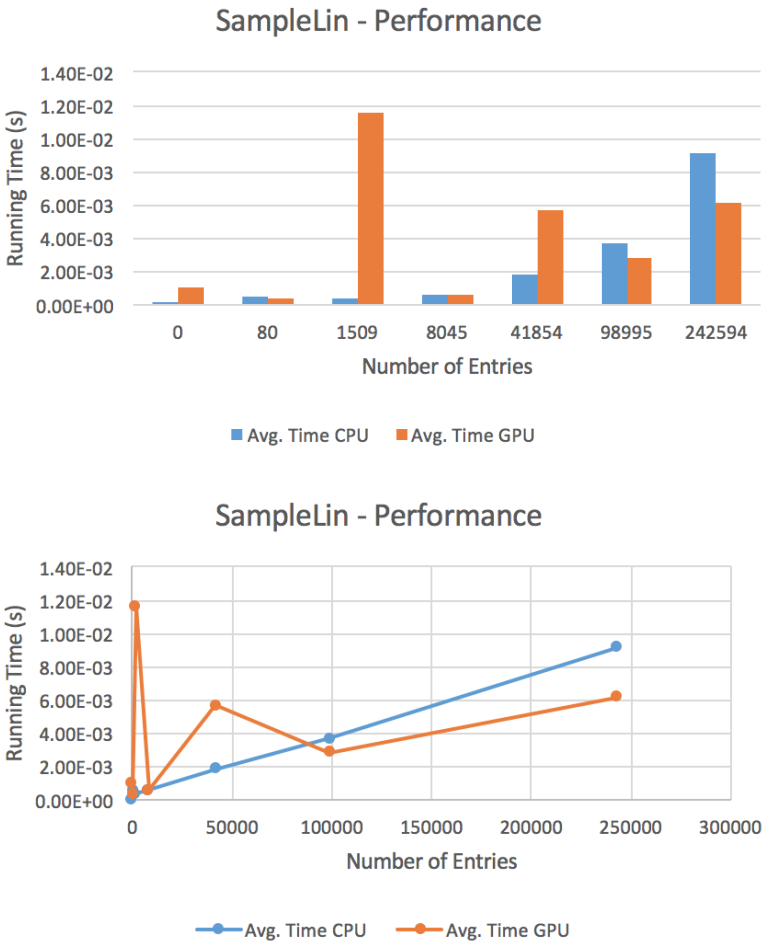
2.15.3 Test Results

Table 29: Test Results – SampleLin

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

2.15.4 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.

2.16 void Times(G4double factor)

2.16.1 Test Description

Multiplies every element in the vector by `factor`.

2.16.2 Test Inputs

Table 30: Unit Tests - `Times`

Test #	**Inputs** `factor`
55	r1
56	r2
57	r3
58	r4
59	r5

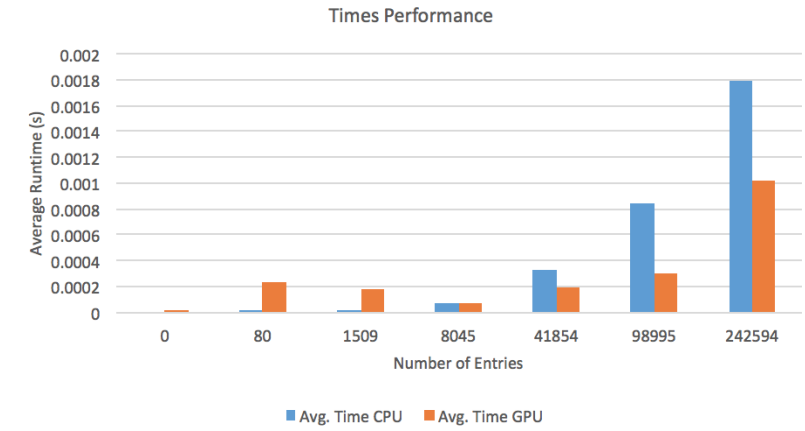
2.16.3 Test Results

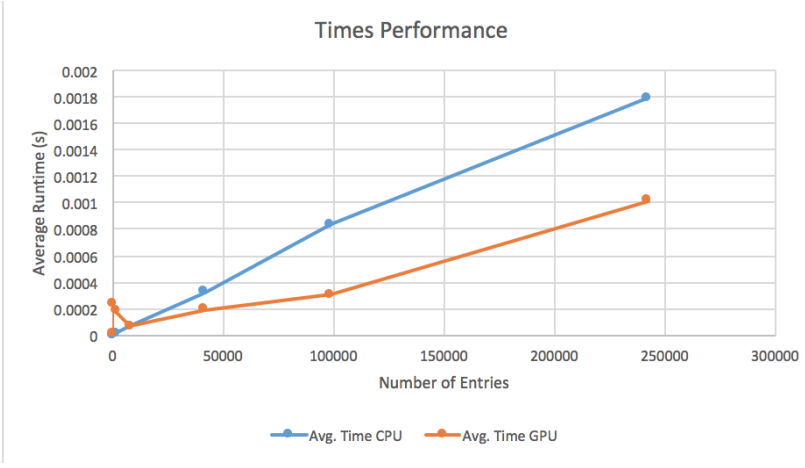
Table 31: Test Results – Times

Test #	**Test Result**						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
55	Pass	Pass	Pass	Pass	Pass	Pass	Pass
56	Pass	Pass	Pass	Pass	Pass	Pass	Pass
57	Pass	Pass	Pass	Pass	Pass	Pass	Pass
58	Pass	Pass	Pass	Pass	Pass	Pass	Pass
59	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.16.4 Performance

Figure 3: Performance results for `Times` function





2.17 void ThinOut(G4double precision)

2.17.1 Test Description

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

2.17.2 Test Inputs

Table 32: Unit Tests - `ThinOut`

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

2.17.3 Test Results

Table 33: Test Results – ThinOut

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

2.17.4 Performance

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

2.18 G4double Sample()

2.18.1 Test Description

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

2.18.2 Test Inputs

Table 34: Unit Tests - `Sample`

Test #	Inputs `N/A`
65	N/A

2.18.3 Test Results

Table 35: Test Results – Sample

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

2.18.4 Performance

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

2.19 SetPoint

2.19.1 Unit Tests

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

Table 36: Unit Tests

Test #	Code	**Description**
66	Empty.SetPoint(-1, rPoint)	Set a point at a negative index of an empty vector
67	Empty.SetPoint(0, rPoint)	Set a point at a the first index of an empty vector
68	Empty.SetPoint(1, rPoint)	Set a point at an index out of bounds of an empty vector
69	D.SetPoint(-1, rPoint)	Set a point at a negative index
70	D.SetPoint(0, rPoint)	Set a point at a the first index
71	D.SetPoint(n/2, rPoint)	Set a point at an index within the vector
72	D.SetPoint(n-1, rPoint)	Set a point at the last index
73	D.SetPoint(n, rPoint)	Set a point at an index our of bounds
74	D.SetPoint(0, nPoint)	Set a negative point
75	D.SetPoint(0, zPoint)	Set a zero point

2.19.2 Accuracy

Table 37: Accuracy

Test #	Status
66	Pass
67	Pass
68	Pass
69	Pass
70	Pass
71	Pass
72	Pass
73	Pass
74	Pass
75	Pass

2.19.3 Performance

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

2.20 G4double GetXsec(G4double e)

2.20.1 Test Description

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

2.20.2 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`
76	r1, r2, r3, r4, r5
77	r1, r2, r3, r4, r5
78	r1, r2, r3, r4, r5
79	r1, r2, r3, r4, r5
80	r1, r2, r3, r4, r5

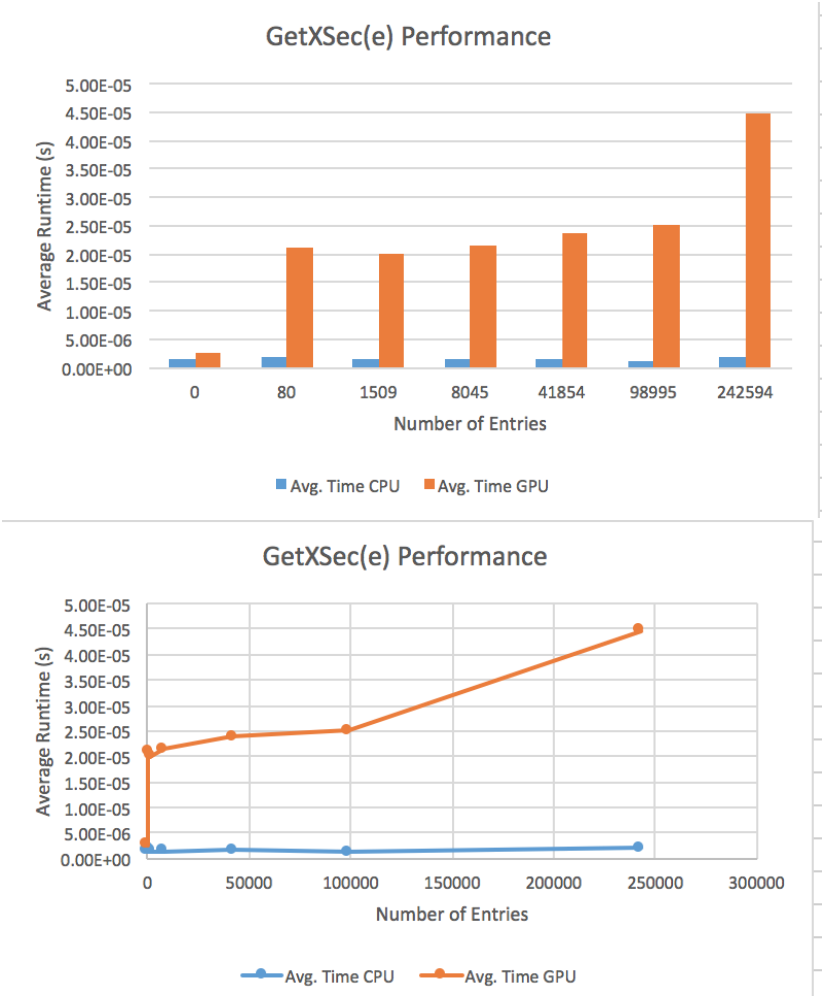
2.20.3 Test Results

Table 39: Test Results – GetXsec

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

2.20.4 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)

2.21.1 Test Description

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

2.21.2 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`
81	r1, r2, r3, r4, r5	-1
82	r1, r2, r3, r4, r5	0
83	r1, r2, r3, r4, r5	n/2
84	r1, r2, r3, r4, r5	n-1
85	r1, r2, r3, r4, r5	n

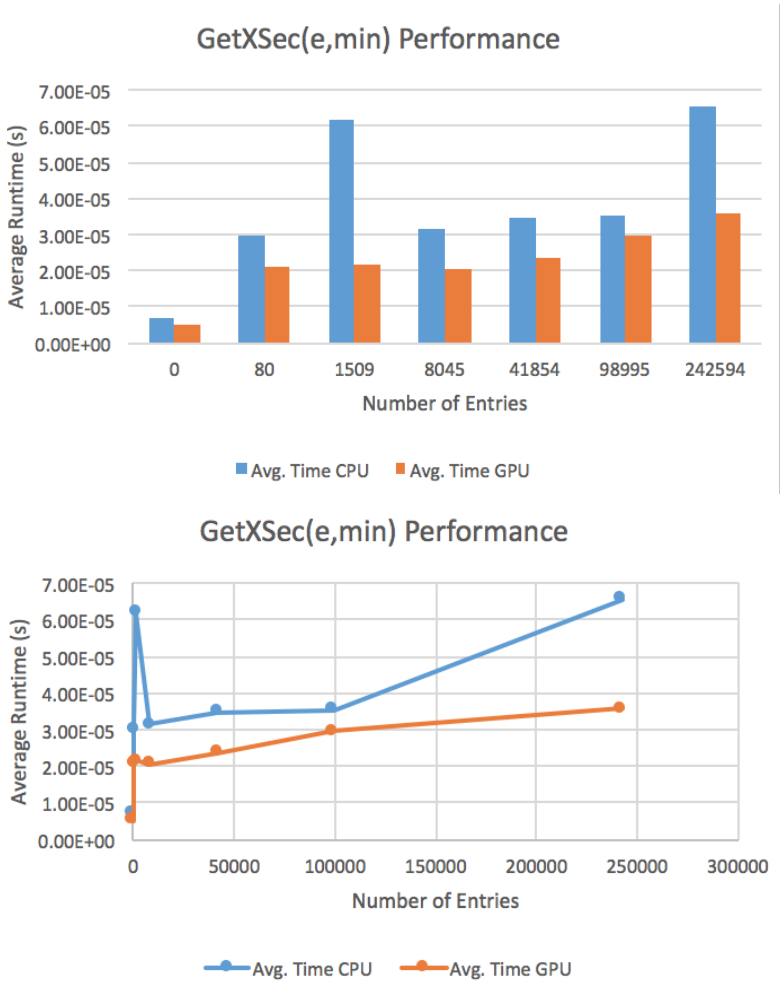
2.21.3 Test Results

Table 41: Test Results – GetXsec

Test #	**Test Result**						
	vec0	vec1	vec2	vec3	vec4	vec5	vec6
81	Pass	Pass	Pass	Pass	Pass	Pass	Pass
82	Pass	Pass	Pass	Pass	Pass	Pass	Pass
83	Pass	Pass	Pass	Pass	Pass	Pass	Pass
84	Pass	Pass	Pass	Pass	Pass	Pass	Pass
85	Pass	Pass	Pass	Pass	Pass	Pass	Pass

2.21.4 Performance

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



2.22 G4double Get15percentBorder()

2.22.1 Test Description

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

2.22.2 Test Inputs

Table 42: Unit Tests - `Get15percentBorder`

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

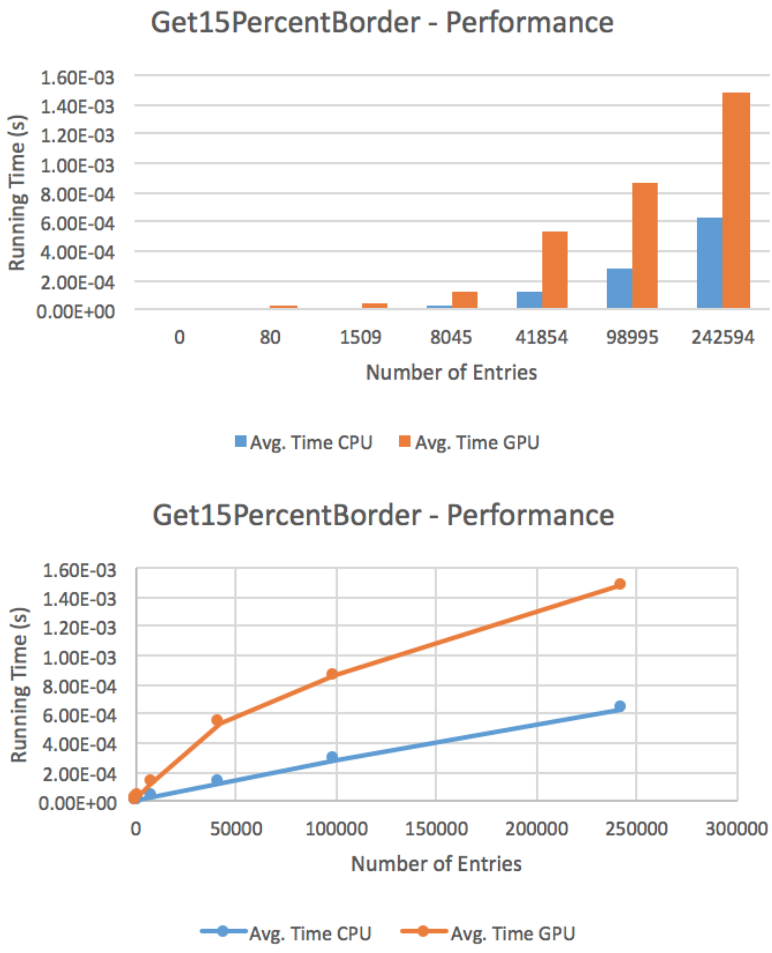
2.22.3 Test Results

Table 43: Test Results – Get15percentBorder

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

2.22.4 Performance

Figure 6: Performance results for `Get15PercentBorder`



2.23 G4double Get50percentBorder()

2.23.1 Test Description

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

2.23.2 Test Inputs

Table 44: Unit Tests - `Get50percentBorder`

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

2.23.3 Test Results

Table 45: Test Results – Get50percentBorder

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

2.23.4 Performance

Figure 7: Performance results for `Get50PercentBorder` function

3 System Tests

3.1 Summary of Tests Performed

System tests will be 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 number of events. The values and conditions that are changed per test are detailed in the table below.

Table 46: System Tests

Test #	Name	Inputs	Outputs	Description
88	System Test - Water, 2000 events	Events = 2000 Material = Water	Same output as non-GPU GEANT4	HADR04 no changes
89	System Test - Uranium, 2000 events	Events = 2000 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – basic example
90	System Test - Water, 600 events	Events = 600 Material = Water	Same output as non-GPU GEANT4	HADR04 – Shorter test
91	System Test - Uranium, 600 events	Events = 600 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – Shorter test
92	System Test - Uranium, 20000 events	Events = 20000 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – Long simulation stress Test
93	System Test - Uranium, 0 events	Events = 0 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – no runs, Edge case

3.2 System Tests Results

This section will summarize all of the results from running tests 39 through 44. Each test has an accuracy section as well as a performance section. The accuracy of the results will be based on how well the values generated on the GPU match up with the values generated on the CPU. The performance metrics used will include user, system and real time required to run each system test.

3.3 System Test - Water, 2000 events

This test simply runs the Hadr04 example on both the GPU and the CPU without changing the source files. The code for this example is bundled with the GEANT4 installation.

3.3.1 Accuracy

Table 47: Accuracy Test # 88

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	423181	441475	NA
nCapture	1998	2000	NA
neutronInelastic	2	0	NA
Parcours of incident neutron			
collisions	212.59	221.74	NA
track length	93.387 cm	96.948 cm	NA
time of flight	202.14 mus	210.93 mus	NA
Generated particles			
C14			
# of particles	2	NA	NA
Emean	404.13 keV	NA	NA
Range	404.02 keV-404.25 keV	NA	NA
O16			
# of particles	5948	6146	NA
Emean	39.577 keV	40.32 keV	NA
Range	0.73669 meV-447.06 keV	1.417 meV-446.59 keV	NA
O17			
# of particles	4	NA	NA
Emean	1.4823 keV	NA	NA
Range	187.15 eV-4.9785 keV	NA	NA
O18			
# of particles	3	NA	NA
Emean	52.362	NA	NA
Range	9.3256 eV-153.04 keV	NA	NA
Alpha			
# of particles	2	NA	NA
Emean	1.4146 MeV	NA	NA
Range	1.4145 MeV-1.4147 MeV	NA	NA
Deuteron			
# of particles	1996	NA	NA
Emean	1.3185 keV	NA	NA
Range	237.52 meV-1.7025 keV	NA	NA
Gamma			
# of particles	2000	NA	NA
Emean	2.2239 MeV	NA	NA
Range	870.8 keV-4.1431 MeV	NA	NA
Proton			
# of particles	45355	NA	NA
Emean	83.046 keV	NA	NA
Range	0.13176 meV-1.9993 MeV	NA	NA

3.3.2 Performance

Table 48: Performance Test # 88

Type	CPU Time	GPU Time
User	55.09s	NA
Real	55.19s	NA
System	0.05s	NA

3.4 System Test - Uranium, 2000 events

This test simply runs the Hadr04 example on both the GPU and the CPU without changing the source files. The code for this example is bundled with the GEANT4 installation.

3.4.1 Accuracy

Table 49: Accuracy Test # 89

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	NA	NA	NA
nCapture	NA	NA	NA
neutronInelastic	NA	NA	NA
Parcours of incident neutron			
collisions	NA	NA	NA
track length	NA	NA	NA
time of flight	NA	NA	NA
Generated particles			
U235			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
U238			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
U239			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Gamma			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Neutron			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA

3.4.2 Performance

Table 50: Performance Test # 89

Type	CPU Time	GPU Time
User	NA	NA
Real	NA	NA
System	NA	NA

3.5 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 code for this example is bundled with the GEANT4 installation.

3.5.1 Accuracy

Table 51: Accuracy Test # 90

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	NA	NA	NA
nCapture	NA	NA	NA
neutronInelastic	NA	NA	NA
Parcours of incident neutron			
collisions	NA	NA	NA
track length	NA	NA	NA
time of flight	NA	NA	NA
Generated particles			
O16			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O17			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O18			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Deuteron			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Gamma			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Proton			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA

3.5.2 Performance

Table 52: Performance Test # 90

Type	CPU Time	GPU Time
User	NA	NA
Real	NA	NA
System	NA	NA

3.6 System Test - Uranium, 600 events

This test simply runs the Hadr04 example on both the GPU and the CPU without changing the source files. The code for this example is bundled with the GEANT4 installation.

3.6.1 Accuracy

Table 53: Accuracy Test # 91

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	NA	NA	NA
nCapture	NA	NA	NA
neutronInelastic	NA	NA	NA
Parcours of incident neutron			
collisions	NA	NA	NA
track length	NA	NA	NA
time of flight	NA	NA	NA
Generated particles			
O16			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O17			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O18			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Deuteron			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Gamma			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Proton			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA

3.6.2 Performance

Table 54: Performance Test # 91

Type	CPU Time	GPU Time
User	NA	NA
Real	NA	NA
System	NA	NA

3.7 System Test - Uranium, 20000 events

This test simply runs the Hadr04 example on both the GPU and the CPU without changing the source files. The code for this example is bundled with the GEANT4 installation.

3.7.1 Accuracy

Table 55: Accuracy Test # 92

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	NA	NA	NA
nCapture	NA	NA	NA
neutronInelastic	NA	NA	NA
Parcours of incident neutron			
collisions	NA	NA	NA
track length	NA	NA	NA
time of flight	NA	NA	NA
Generated particles			
O16			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O17			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O18			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Deuteron			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Gamma			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Proton			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA

3.7.2 Performance

Table 56: Performance Test # 92

Type	CPU Time	GPU Time
User	NA	NA
Real	NA	NA
System	NA	NA

3.8 System Test - Uranium, 0 events

This test simply runs the Hadr04 example on both the GPU and the CPU without changing the source files. The code for this example is bundled with the GEANT4 installation.

3.8.1 Accuracy

Table 57: Accuracy Test # 93

Data	CPU Values	GPU Values	Difference
Process Calls			
hadElastic	NA	NA	NA
nCapture	NA	NA	NA
neutronInelastic	NA	NA	NA
Parcours of incident neutron			
collisions	NA	NA	NA
track length	NA	NA	NA
time of flight	NA	NA	NA
Generated particles			
O16			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O17			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
O18			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Deuteron			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Gamma			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
Proton			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA

3.8.2 Performance

Table 58: Performance Test # 93

Type	CPU Time	GPU Time
User	NA	NA
Real	NA	NA
System	NA	NA

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 59: 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 60: 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.