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 Figures

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

Definitions and Acronyms

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 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

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 = (overloaded assignment operator)

2.3.1 Method Signature

```
G4ParticleHPVector & operator = (const G4ParticleHPVector & right)
```

2.3.2 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.

2.3.3 Test Inputs

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

Test #	**Inputs** `right`
1	Current vector

2.3.4 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

2.3.5 Performance

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

2.4 GetPoint

2.4.1 Method Signature

```
const G4ParticleHPDataPoint GetPoint(G4int i)
```

2.4.2 Test Description

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

2.4.3 Test Inputs

Table 6: Unit Tests - `GetPoint`

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

2.4.4 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

2.4.5 Performance

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

2.5 GetX

2.5.1 Unit Tests

Table 8: Unit Tests

Test #	Code	**Description**
7	Empty.GetX(-1)	Set an xSec at a negative index of an empty vector
8	Empty.GetX(0)	Set an xSec at a the first index of an empty vector
9	Empty.GetX(1)	Set an xSec at an index out of bounds of an empty vector
10	D.GetX(-1)	Set an xSec at a negative index
11	D.GetX(0)	Set an xSec at a the first index
12	D.GetX(n/2)	Set an xSec at an index within the vector
13	D.GetX(n-1)	Set an xSec at the last index
14	D.GetX(n)	Set an xSec at an index our of bounds

2.5.2 Accuracy

Table 9: Accuracy

Test #	Status
7	Pass
8	Pass
9	Pass
10	Pass
11	Pass
12	Pass
13	Pass
14	Pass

2.5.3 Performance

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

2.6 GetY

2.6.1 Unit Tests

Table 10: Unit Tests

Test #	Code	**Description**
15	Empty.GetY(-1)	Get a point at a negative index of an empty vector
16	Empty.GetY(0)	Get a point at a the first index of an empty vector
17	Empty.GetY(1)	Get a point at an index out of bounds of an empty vector
18	D.GetY(-1)	Get a point at a negative index
19	D.GetY(0)	Get a point at a the first index
20	D.GetY(n/2)	Get a point at an index within the vector
21	D.GetY(n-1)	Get a point at the last index
22	D.GetY(n)	Get a point at an index our of bounds

2.6.2 Accuracy

Table 11: Accuracy

Test #	Status
15	Pass
16	Pass
17	Pass
18	Pass
19	Pass
20	Pass
21	Pass
22	Pass

2.6.3 Performance

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

2.7 GetXsec

2.7.1 Unit Tests

Table 12: Unit Tests

Test #	Code	**Description**
23	Empty.GetXsec(-1)	Get an xSec with a negative energy from an empty vector
24	Empty.GetXsec(0)	Get a xSec with an energy of zero from an empty vector
25	Empty.GetXsec(r1)	Get a xSec with a normal energy from an empty vector
26	D.GetXsec(-1)	Get a xSec with a negative energy
27	D.GetXsec(0)	Get a xSec with a zero energy
28	D.GetXsec(r0)	Get a xSec with a small energy
29	D.GetXsec(r1)	Get a xSec with a normal energy
30	D.GetXsec(r2)	Get a xSec with a large energy

2.7.2 Accuracy

Table 13: Accuracy

Test #	Status
23	Pass
24	Pass
25	Pass
26	Pass
27	Pass
28	Pass
29	Pass
30	Pass

2.7.3 Performance

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

2.8 SetData

2.8.1 Unit Tests

Table 14: Unit Tests

Test #	Code	**Description**
31	Empty.SetData(-1, r1, r2)	Set a point at a negative index of an empty vector
32	Empty.SetData(0, r1, r2)	Set a point at a the first index of an empty vector
33	Empty.SetData(1, r1, r2)	Set a point at an index out of bounds of an empty vector
34	D.SetData(-1, r1, r2)	Set a point at a negative index
35	D.SetData(0, r1, r2)	Set a point at a the first index
36	D.SetData(n/2, r1, r2)	Set a point at an index within the vector
37	D.SetData(n-1, r1, r2)	Set a point at the last index
38	D.SetData(n, r1, r2)	Set a point at an index our of bounds
39	D.SetData(0, -1, -1)	Set a point with a negative energy and xSec
40	D.SetData(0, 0, 0)	Set a point with a zero energy and xSec

2.8.2 Accuracy

Table 15: Accuracy

Test #	Status
31	Pass
32	Pass
33	Pass
34	Pass
35	Pass
36	Pass
37	Pass
38	Pass
39	Pass
40	Pass

2.8.3 Performance

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

2.9 SetEnergy

2.9.1 Unit Tests

Table 16: Unit Tests

Test #	Code	**Description**
41	Empty.SetEnergy(-1, r1)	Set an energy at a negative index of an empty vector
42	Empty.SetEnergy(0, r1)	Set an energy at a the first index of an empty vector
43	Empty.SetEnergy(1, r1)	Set an energy at an index out of bounds of an empty vector
44	D.SetEnergy(-1, r1)	Set an energy at a negative index
45	D.SetEnergy(0, r1)	Set an energy at a the first index
46	D.SetEnergy(n/2, r1)	Set an energy at an index within the vector
47	D.SetEnergy(n-1, r1)	Set an energy at the last index
48	D.SetEnergy(n, r1)	Set an energy at an index our of bounds
49	D.SetEnergy(0, -1)	Set an energy at an index within the vector to a negative value
50	D.SetEnergy(0, 0)	Set an energy at an index within the vector to a zero value

2.9.2 Accuracy

Table 17: Accuracy

Test #	Status
41	Pass
42	Pass
43	Pass
44	Pass
45	Pass
46	Pass
47	Pass
48	Pass
49	Pass
50	Pass

2.9.3 Performance

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

2.10 SetXsec

2.10.1 Unit Tests

Table 18: Unit Tests

Test #	Code	**Description**
51	Empty.SetXsec(-1, r1)	Set an xSec at a negative index of an empty vector
52	Empty.SetXsec(0, r1)	Set an xSec at a the first index of an empty vector
53	Empty.SetXsec(1, r1)	Set an xSec at an index out of bounds of an empty vector
54	D.SetXsec(-1, r1)	Set an xSec at a negative index
55	D.SetXsec(0, r1)	Set an xSec at a the first index
56	D.SetXsec(n/2, r1)	Set an xSec at an index within the vector
57	D.SetXsec(n-1, r1)	Set an xSec at the last index
58	D.SetXsec(n, r1)	Set an xSec at an index our of bounds
59	D.SetXsec(0, -1)	Try to set a negative xSec
60	D.SetXsec(0, 0)	Try to set a zero xSec

2.10.2 Accuracy

Table 19: Accuracy

Test #	Status
51	Pass
52	Pass
53	Pass
54	Pass
55	Pass
56	Pass
57	Pass
58	Pass
59	Pass
60	Pass

2.10.3 Performance

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

2.11 SetX

2.11.1 Unit Tests

Table 20: Unit Tests

Test #	Code	**Description**
61	Empty.SetX(-1, r1)	Set an energy at a negative index of an empty vector
62	Empty.SetX(0, r1)	Set an energy at a the first index of an empty vector
63	Empty.SetX(1, r1)	Set an energy at an index out of bounds of an empty vector
64	D.SetX(-1, r1)	Set an energy at a negative index
65	D.SetX(0, r1)	Set an energy at a the first index
66	D.SetX(n/2, r1)	Set an energy at an index within the vector
67	D.SetX(n-1, r1)	Set an energy at the last index
68	D.SetX(n, r1)	Set an energy at an index our of bounds
69	D.SetX(0, -1)	Set a negative energy
70	D.SetX(0, 0)	Set a zero energy

2.11.2 Accuracy

Table 21: Accuracy

Test #	Status
61	Pass
62	Pass
63	Pass
64	Pass
65	Pass
66	Pass
67	Pass
68	Pass
69	Pass
70	Pass

2.11.3 Performance

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

2.12 SetY

2.12.1 Unit Tests

Table 22: Unit Tests

Test #	Code	**Description**
71	Empty.SetY(-1, r1)	Set an xSec at a negative index of an empty vector
72	Empty.SetY(0, r1)	Set an xSec at a the first index of an empty vector
73	Empty.SetY(1, r1)	Set an xSec at an index out of bounds of an empty vector
74	D.SetY(-1, r1)	Set an xSec at a negative index
75	D.SetY(0, r1)	Set an xSec at a the first index
76	D.SetY(n/2, r1)	Set an xSec at an index within the vector
77	D.SetY(n-1, r1)	Set an xSec at the last index
78	D.SetY(n, r1)	Set an xSec at an index our of bounds
79	D.SetY(0, -1)	Set a negative xSec
80	D.SetY(0, 0)	Set a zero xSec

2.12.2 Accuracy

Table 23: Accuracy

Test #	Status
71	Pass
72	Pass
73	Pass
74	Pass
75	Pass
76	Pass
77	Pass
78	Pass
79	Pass
80	Pass

2.12.3 Performance

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

2.13 Init

2.13.1 Unit Tests

Table 24: Unit Tests

Test #	Code	**Description**
81	Empty.Init()	Init an empty Vector
82	D.Init()	Init a Vector

2.13.2 Accuracy

Table 25: Accuracy

Test #	Status
81	Pass
82	Pass

2.13.3 Performance

2.14 SampleLin

2.14.1 Unit Tests

Table 26: Unit Tests

Test #	Code	**Description**
83	Empty.SampleLin()	Sample an empty Vector
84	D.SampleLin()	Sample a Vector

2.14.2 Accuracy

Table 27: Accuracy

Test #	CPU	GPU
83	CPU result	GPU result
84	CPU result	GPU result

2.14.3 Performance

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

2.15 Integrate

2.15.1 Unit Tests

Table 28: Unit Tests

Test #	Code	**Description**
85	Empty.Integrate()	Integrate an empty Vector
86	D.Integrate()	Integrate a Vector

2.15.2 Accuracy

Table 29: Accuracy

Test #	Status
85	Pass
86	Pass

2.15.3 Performance

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

2.16 IntegrateAndNormalise

2.16.1 Unit Tests

Table 30: Unit Tests

Test #	Code	**Description**
87	Empty.IntegrateAndNormalise()	Integrate and normalize an empty Vector
88	D.IntegrateAndNormalise()	Integrate normalize a Vector

2.16.2 Accuracy

Table 31: Accuracy

Test #	Status
30	Pass
30	Pass

2.16.3 Performance

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

2.17 Times

2.17.1 Unit Tests

Table 32: Unit Tests

Test #	Code	**Description**
89	Empty.Times(-1)	Times an empty vector by a negative factor
90	Empty.Times(0)	Times an empty vector by zero
91	Empty.Times(1)	Times an empty vector by 1
92	Empty.Times(r1)	Times an empty vector by a random factor
93	D.Times(-1)	Times a vector by a negative factor
94	D.Times(0)	Times a vector by zero
95	D.Times(1)	Times a vector by 1
96	D.Times(r1)	Times a vector by a random factor

2.17.2 Accuracy

Table 33: Accuracy

Test #	Status
89	Pass
90	Pass
91	Pass
92	Pass
93	Pass
94	Pass
95	Pass
96	Pass

2.17.3 Performance

2.18 ThinOut

2.18.1 Unit Tests

Table 34: Unit Tests

Test #	Code	**Description**
97	Empty.ThinOut(r1)	ThinOut an empty Vector
98	D.ThinOut(-1)	ThinOut a Vector using a negative value
99	D.ThinOut(0)	ThinOut a Vector using a zero value
100	D.ThinOut(r0)	ThinOut a Vector using a small value
101	D.ThinOut(r1)	ThinOut a Vector using a normal value
102	D.ThinOut(r2)	ThinOut a Vector using a large value

2.18.2 Accuracy

Table 35: Accuracy

Test #	Status
107	Pass
108	Pass
109	Pass
110	Pass
111	Pass
112	Pass

2.18.3 Performance

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

2.19 Sample

2.19.1 Unit Tests

Table 36: Unit Tests

Test #	Code	**Description**
103	Empty.Sample()	Sample an empty Vector
104	D.Sample()	Sample a Vector

2.19.2 Accuracy

Table 37: Accuracy

Test #	CPU	GPU
113	CPU result	GPU result
114	CPU result	GPU result

2.19.3 Performance

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

2.20 SetPoint

2.20.1 Unit Tests

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

Table 38: Unit Tests

Test #	Code	**Description**
105	Empty.SetPoint(-1, rPoint)	Set a point at a negative index of an empty vector
106	Empty.SetPoint(0, rPoint)	Set a point at a the first index of an empty vector
107	Empty.SetPoint(1, rPoint)	Set a point at an index out of bounds of an empty vector
108	D.SetPoint(-1, rPoint)	Set a point at a negative index
109	D.SetPoint(0, rPoint)	Set a point at a the first index
110	D.SetPoint(n/2, rPoint)	Set a point at an index within the vector
111	D.SetPoint(n-1, rPoint)	Set a point at the last index
112	D.SetPoint(n, rPoint)	Set a point at an index our of bounds
113	D.SetPoint(0, nPoint)	Set a negative point
114	D.SetPoint(0, zPoint)	Set a zero point

2.20.2 Accuracy

Table 39: Accuracy

Test #	Status
115	Pass
116	Pass
117	Pass
118	Pass
119	Pass
120	Pass
121	Pass
122	Pass
123	Pass
124	Pass

2.20.3 Performance

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

2.21 Get15percentBorder

2.21.1 Unit Tests

Table 40: Unit Tests

Test #	Code	**Description**
115	Empty.Get15percentBorder()	Get 15 percent Border of an empty vector
116	D.Get15percentBorder()	Get 15 percent Border of a vector

2.21.2 Accuracy

Table 41: Accuracy

Test #	Status
125	Pass
126	Pass

2.21.3 Performance

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

2.22 Get50percentBorder

2.22.1 Unit Tests

Table 42: Unit Tests

Test #	Code	**Description**
117	Empty.Get50percentBorder()	Get 50 percent Border of an empty vector
118	D.Get50percentBorder()	Get 50 percent Border of a vector

2.22.2 Accuracy

Table 43: Accuracy

Test #	Status
127	Pass
128	Pass

2.22.3 Performance

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

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 44: System Tests

Test #	Initial State	Inputs	Outputs	Description
119	Fresh start up	Events = 2000 Material = Water	Same output as non-GPU GEANT4	HADR04 no changes
120	Fresh start up	Events = 2000 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – basic example
121	Fresh start up	Events = 600 Material = Water	Same output as non-GPU GEANT4	HADR04 – Shorter test
122	Fresh start up	Events = 600 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – Shorter test
123	Fresh start up	Events = 20000 Material = Uranium	Same output as non-GPU GEANT4	HADR04 – Long simulation stress Test
124	Fresh start up	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 # 39

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 45: Accuracy Test #39

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			
C14			
# of particles	NA	NA	NA
Emean	NA	NA	NA
Range	NA	NA	NA
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
Alpha			
# 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

3.3.2 Performance

Table 46: Performance Test #39

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

3.4 System test # 40

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 47: Accuracy Test #40

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 48: Performance Test #40

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

3.5 System test # 41

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 49: Accuracy Test #41

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
Alpha			
# 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 50: Performance Test #41

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 51: 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 & 6& 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:

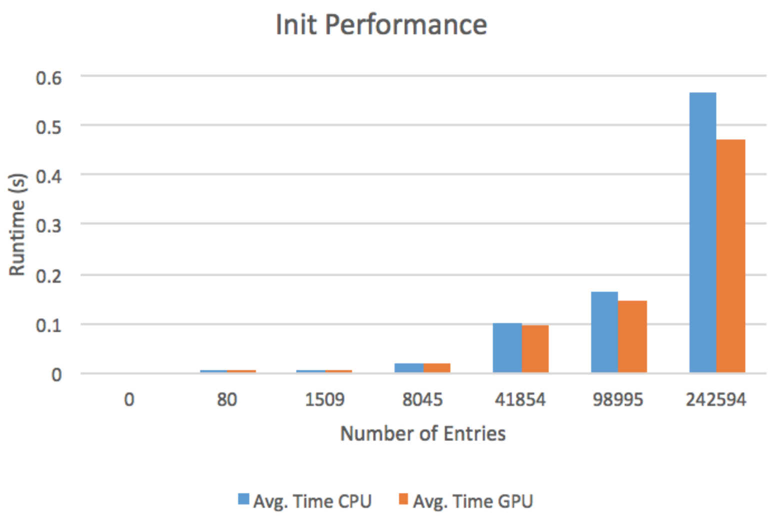
Table 52: Tests and Modules Relationship

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 was a common theme. In some of these cases the same case was not covered by Geant4 itself, so the change was made there as well.

Figure 1: Performance results for `Init` function



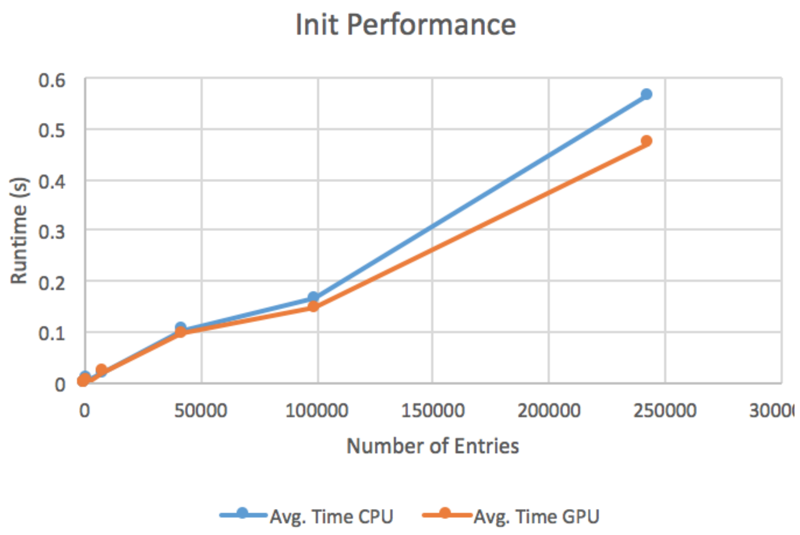
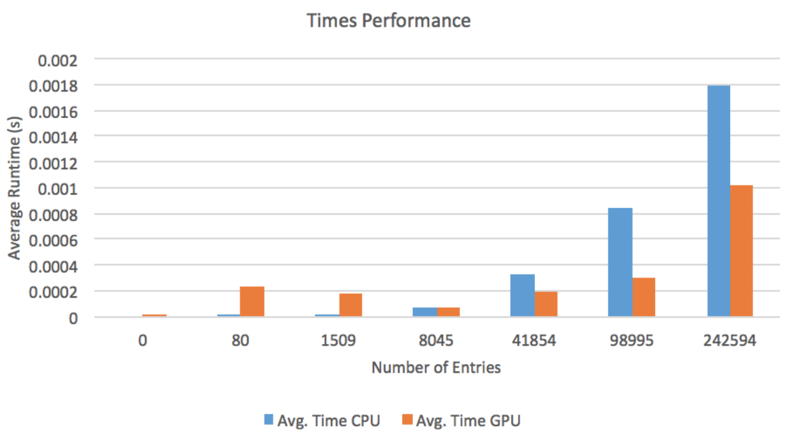
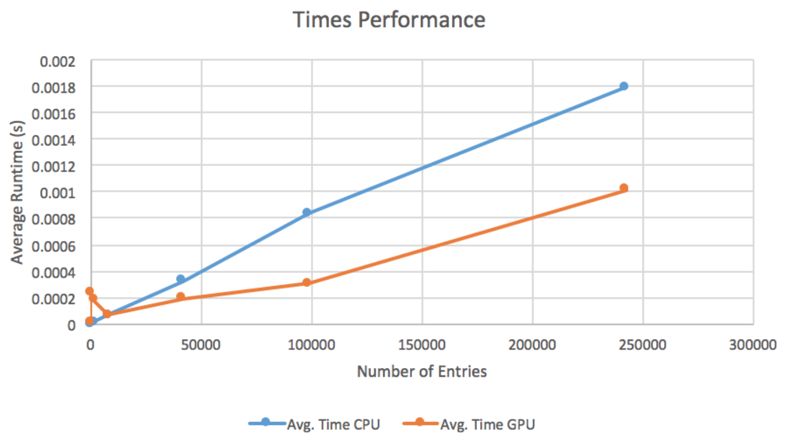


Figure 2: Performance results for `Times` function


Times Performance
Average Runtime (s)
Number of Entries
Avg. Time CPU
Avg. Time GPU


Times Performance
Average Runtime (s)
Number of Entries
Avg. Time CPU
Avg. Time GPU