Software Integration Tests:
GPU N-Body Integration Toolkit

Prepared by:
Daniel Bagnell
Jason Economou
Rajkumar Jayachandran
Tim McJilton
Gabe Schwartz
Andrew Sherman

Advisor:
Prof. Jeremy Johnson

Stakeholders:
Prof. Steve McMillan
Alfred Whitehead

Contents

1 Introduction **4**

2 Test Assumptions and Exclusions **4**
2.1 Test Assumptions 4
2.2 Test Exclusions 4

3 Graphical User Interface **4**
3.1 Tool Integration 4
3.2 Network Interaction 5

4 Experiment Components **6**
4.1 Overview 6
4.2 Testing Experiment Export 6
4.3 Testing Experiment Import 6
4.4 Particles 6
4.4.1 Particle Class 6
4.5 Modules 7
4.5.1 Module Class 7
4.5.2 Parameter Class 7
4.5.3 Unit Class 7
4.5.4 CompoundUnit Class 7
4.5.5 UnitType Enumeration 8
4.5.6 SIPrefix Enumeration 8
4.5.7 AstrophysicalDomain Enumeration 8
4.5.8 StoppingConditions Enumeration 8
4.6 Diagnostics 8
4.6.1 Built-in Diagnostic Test 8
4.6.2 Custom Diagnostic Test 8
4.6.3 Diagnostic Condition Test 9
4.7 Logging 9

5 Command Line Tool **10**
5.1 Overview 10
5.2 Testing CLT Flags 10
5.2.1 Filename Flag Test 10
5.2.2 Number of Particles Flag Test 10
5.2.3 Time Flags Test 11
5.2.4 Help Prompt Flag Test 11
5.3 Testing The Produced N-Body Simulation 12
5.3.1 Introduction 12
5.3.2 Test Methodology 12
5.3.3 Test Data 12

6 Networking **13**

List of Tests

3.1 GUI Experiment Integration 4
3.2 Node Viewing 5
3.3 Distributed Experiment 5
4.1 Export Experiment XML 6
4.2 Import Experiment XML 6
4.3 Built-in Diagnostic Test 8
4.4 Custom Diagnostic Test 8
4.5 Diagnostic Condition Test 9
4.6 Logging 9
5.1 Filename Flag Test 10
5.2 Number of Particles Flag Test 10
5.3 Time Flags Test 11
5.4 Help Flag Test 11
5.5 N-Body Simulation Test 12
6.1 Control Instance - Node Instance Interaction: Status Query 13
6.2 Control Instance - Node Instance Interaction: Experiment Query . . 14

1 Introduction

This document describes the necessary integration tests of the GPUnit software system described by the GPUnit Software Design Document.

2 Test Assumptions and Exclusions

This sections provides the integration test assumptions as well as the components of the GPUnit system that will not be covered by the integration testing process.

2.1 Test Assumptions

It is assumed that all individual components of the GPUnit software system had unit tests run previously and that these unit tests cover all aspects not included in the Integration Test Plan. The Integration Test Plan covers:

- interaction of the GPUnit software components
- interaction of the GPUnit software components with system components

2.2 Test Exclusions

The Integration Test Plan does not cover:

- individual intra-component functionality covered by unit tests
- structural integrity of the source code

3 Graphical User Interface

3.1 Tool Integration

Name	Run an Experiment via GUI
Description	The user runs an experiment locally using the interface.
Pre-conditions	An experiment is open in the user interface.
Actions	• The user selects the "Run experiment..." option in the interface. • The user selects only the local machine from the available nodes.
Post-conditions	The experiment runs, generating any selected logging and diagnostic output.

Table 3.1: GUI Experiment Integration

3.2 Network Interaction

Name	View a List of Nodes
Description	The user views a list of nodes available in the cluster.
Pre-conditions	• A cluster of nodes is available and running the Node Instance code. • The user interface is open.
Actions	• The user opens the node viewer window in the interface.
Post-conditions	The node window appears with all available nodes in the cluster displayed along with their respective status.

Table 3.2: Node Viewing

Name	Run an Experiment on a Remote Node
Description	The user runs their experiment, sending different modules to run on different nodes.
Pre-conditions	• The user has loaded an experiment with multiple modules in the user interface. • The user has access to a remote machine(s) which are running the Node Instance code.
Actions	• The user selects the "Run experiment..." option in the interface. • The user assigns at least one module to at least one of the remote machines in the cluster.
Post-conditions	• The experiment runs, sending each module's work to the selected cluster node. • The logging and diagnostic output of the experiment return with the correct state of the system.

Table 3.3: Distributed Experiment

4 Experiment Components

4.1 Overview

Integration tests for the Experiment class.

4.2 Testing Experiment Export

Name	Export Experiment XML
Description	Writes an Experiment to XML
Pre-conditions	• User creates a valid experiment. • User chooses a filename to save the file to.
Actions	Call Experiment writeXMLFile method with the user's filename.
Post-conditions	• Experiment is written to the supplied filename in XML. • XML file matches experiment data/entities.

Table 4.1: Export Experiment XML

4.3 Testing Experiment Import

Name	Import Experiment XML
Description	Reads an Experiment from XML
Pre-conditions	• User chooses a valid experiment XML file from disk.
Actions	Call Experiment loadXMLFile method with the user's filename.
Post-conditions	• New experiment is created. • Experiment data/entities match XML file data.

Table 4.2: Import Experiment XML

4.4 Particles

4.4.1 Particle Class

4.4.1.1 Particle Attributes

4.4.1.2 Particle Methods

4.5 Modules

4.5.1 Module Class

4.5.1.1 Module Attributes

4.5.1.2 Module Operations

4.5.2 Parameter Class

4.5.2.1 Parameter Attributes

4.5.2.2 Parameter Operations

4.5.3 Unit Class

4.5.3.1 Unit Attributes

4.5.4 CompoundUnit Class

4.5.4.1 CompoundUnit Attributes

4.5.5 UnitType Enumeration

4.5.6 SIPrefix Enumeration

4.5.7 AstrophysicalDomain Enumeration

4.5.8 StoppingConditions Enumeration

4.6 Diagnostics

4.6.1 Built-in Diagnostic Test

Name	Built-in Diagnostic Test
Description	The user can add built-in diagnostics to an experiment.
Pre-conditions	An experiment is open in the user interface.
Actions	• The user selects to add a diagnostic. • The user selects one of the built-in diagnostics. • The user runs the experiment.
Post-conditions	The diagnostic has performed an update for each time step of the experiment.

Table 4.3: Built-in Diagnostic Test

4.6.2 Custom Diagnostic Test

Name	Custom Diagnostic Test
Description	The user can add a custom diagnostic to an experiment.
Pre-conditions	• An experiment is loaded in the user interface. • The user has a valid custom diagnostic.
Actions	• The user selects to add a diagnostic. • The user selects the custom diagnostics. • The user runs the experiment.
Post-conditions	The diagnostic has performed an update for each time step of the experiment.

Table 4.4: Custom Diagnostic Test

4.6.3 Diagnostic Condition Test

Name	Diagnostic Condition Test
Description	The user can add conditions to indicate when the diagnostics are updated.
Pre-conditions	• An experiment is loaded in the user interface. • A diagnostic is attached to the experiment.
Actions	• The user selects to modify the properties of the diagnostic. • The user enters conditions when the diagnostic is to be updated. • The user runs the experiment.
Post-conditions	The diagnostic has performed an update only when the conditions have been satisfied.

Table 4.5: Diagnostic Condition Test

4.7 Logging

Name	Output destination for logs
Description	The user selects the mode of logs output for the experiment
Pre-conditions	The experiment is selected and Logging is enabled
Actions	The user specifies one of the output location. The output destination could be a console, a file or window in the interface.
Post-conditions	When experiment is started the logs information appears in the specified output destination.

Table 4.6: Logging

5 Command Line Tool

5.1 Overview

5.2 Testing CLT Flags

5.2.1 Filename Flag Test

Name	Filename Flag Test
Description	User runs simulation using CLT with the -f flag.
Pre-conditions	• User has valid Experiment file to run • User has default Logging enabled
Actions	• User runs <CLT> -f <Experiment File> • User compares data printed out in the log to experiment file
Post-conditions	Experiment printed out in the log is equal to experiment file. Simulation begins running.

Table 5.1: Filename Flag Test

5.2.2 Number of Particles Flag Test

Name	Number of Particles Flag Test
Description	User runs simulation using CLT with the -n flag.
Pre-conditions	• User has valid Experiment file to run which has number of particles not equal to 100 • User has default Logging enabled
Actions	• User runs <CLT> -n 100 • User compares data printed out in the log to experiment file
Post-conditions	Experiment printed out in the log has number of particles set to 100. Simulation begins running.

Table 5.2: Number of Particles Flag Test

5.2.3 Time Flags Test

Name	Time Flags Test
Description	User runs simulation using CLT with the -t and -dt flag.
Pre-conditions	• User has valid Experiment file to run with time step of not .1 and end time of not 4 • User has default Logging enabled
Actions	• User runs <CLT> -t 4 -dt .1 • User compares data printed out in the log to experiment file
Post-conditions	Experiment printed out in the log is equal to experiment file except end time is 4 <time unit in experiment file> and the timestep is .1<time unit in experimental file>. Simulation begins running.

Table 5.3: Time Flags Test

5.2.4 Help Prompt Flag Test

Name	Help Prompt Flag Test
Description	User runs simulation using CLT with the -h flag.
Pre-conditions	• N/A
Actions	• User runs <CLT> -h
Post-conditions	Help prompt prints out how to use all flags and a brief description on how to use the CLT.

Table 5.4: Help Flag Test

5.3 Testing The Produced N-Body Simulation

Name	N-Body Simulation χ^2 Test
Description	1,000 Particles Simulation compared to actual data via χ^2 test.
Pre-conditions	The User has default logging scripts installed.
Actions	• User performs the initialization steps as in section 5.3.3.1 • User runs experiment as in section 5.3.3.2
Post-conditions	User uses the test methodology described in section 5.3.2 and compares the data to what is defined in section 5.3.3.3

Table 5.5: N-Body Simulation Test

5.3.1 Introduction

Testing whether the simulation returns good or bad data is vital in this project yet a difficult thing to do. The simulation will have a factor of randomness compared to existing simulations and data. This requires some statistical testing. To test whether the simulation is accurate or not we will do a χ^2 goodness of fit test.

5.3.2 Test Methodology

A χ^2 goodness of fit test requires a previous model or set of data to compare the new data to. We will call previous data the expected. We will call the data returned by the simulation created via GPUNIT the observed. The χ^2test requires a hypothesis. Our hypothesis will be that the observed will equal the expected. We then solve for χ^2 by using equation 5.1 .

$$\chi^2 = \sum_{i=1}^{M} \frac{(O_i - E_i)^2}{E_i} \tag{5.1}$$

5.3.3 Test Data

There are a few tests we will be doing on a set of particles. We will be comparing the Virial Radius, the ratio of Kinetic Energy over Potential Energy, and the Net Energy.

5.3.3.1 Initialization The particles we will be running on will be a set of 1,000 particles with a Plummer Model for distribution of the particles. These particles will also have a Salpeter Mass Distribution. User will create a new experiment selecting the appropriate fields to create this experiment. The experiment file will be saved in a local location.

5.3.3.2 Run Experiment The user runs the simulation with the experiment file being passed in with the -f flag. The simulation exports the custom log file with the total energy level, the ratio of the kinetic energy of the potential energy, and the virial radius of the particles. The calculations of all these values are held within the logging script.

5.3.3.3 Compare Data The data testing will require a set of trustworthy data or a function to use for comparing the data to from our stakeholders. We will run the χ^2 test on. We will compare the ratio of the kinetic energy and potential energy to $\frac{1}{2}$. Future information on this test will be passed on χ^2 test looking up the information in the table.

6 Networking

Name	Multicast Discovery Test
Description	Test the multicast node discovery feature.
Pre-conditions	Some number (≥ 2) of networked machines are running the Node Instance code.
Actions	• The user starts up a Control Instance on a machine connected to the same network as the nodes. • The user signals the Control Instance that they want to discover a list of nodes on the network (either via the UI or the command line).
Post-conditions	• The Control Instance sends out a multicast query packet. • All nodes send a response packet. • The tool used to generate the request receives a list of available nodes from the Control Instance.

Table 6.1: Control Instance - Node Instance Interaction: Status Query

Name	Experiment Status Test
Description	Test the experiment status query on remote nodes.
Pre-conditions	• Some number (≥ 2) of networked machines are running the Node Instance code. • One or more of the networked nodes have running experiments on them.
Actions	• The user starts up a Control Instance on a machine connected to the same network as the nodes. • The user signals the Control Instance that they want to request a list of experiments from one of the nodes running experiments (either via the UI or the command line).
Post-conditions	• The Control Instance sends out an experiment status query packet. • The selected nodes send a response packet with a list of experiments running on them. • The tool used to generate the request receives a list of nodes and corresponding experiments from the Control Instance.

Table 6.2: Control Instance - Node Instance Interaction: Experiment Query