**Blue Waters Petascale Semester Curriculum v1.0**

**Unit 1: Computation Across the Curriculum**

**Lesson 5: Why Parallel Programming?**

**Exercise Instructions for Students**

*Developed by Colleen Heinemann for the Shodor Education Foundation, Inc.*



*Except where otherwise noted, this work by The Shodor Education Foundation, Inc. is licensed under CC BY-SA 4.0. To view a copy of this license, visit*[*https://creativecommons.org/licenses/by-sa/4.0*](https://creativecommons.org/licenses/by-sa/4.0)

*Browse and search the full curriculum at*[*http://shodor.org/petascale/materials/semester-curriculum*](http://shodor.org/petascale/materials/semester-curriculum)

*We welcome your improvements! You can submit your proposed changes to this material and the rest of the curriculum in our GitHub repository at*[*https://github.com/shodor-education/petascale-semester-curriculum*](https://github.com/shodor-education/petascale-semester-curriculum)

*We want to hear from you! Please let us know your experiences using this material by sending email to* [*petascale@shodor.org*](mailto:petascale@shodor.org)

**Time to Science with GalaxSee**

This exercise is taken from the lab at the Blue Waters Institute 2018 prepared by Mobeen Ludin and Aaron Weeden at the Shodor Education Foundation

**Goals**

* Learn and understand the following concepts:
  + What is “Time to Science”
  + Strong and weak scaling
  + N-body models
  + Parameter sweep
* Practice:
  + Connecting to a supercomputer
  + Downloading code from GitHub
  + Compiling code on a supercomputer
  + Running a problem on a supercomputer with different numbers of processing elements (PEs)
  + Gathering and plotting runtime data
  + Analyzing how a program scales

**Introduction**

**Time to Science** is a measure of how long it takes for a scientific program to run. The goal of using parallel computing is one, or a combination of, the following:

1. Reduce the time to science for a given model
2. Solve a bigger problem in the same time
3. Solve a problem better in the same time

This covers the concept discussed of “better, faster, more”. If we adjust the number of **processing elements (PEs)** assigned to run a given problem, we can observe how the program **scales**, or how its **time to science** increases or decreases, depending on how many resources are assigned combined with the complexity of the program.

There are 2 types of scaling focused on here: **strong scaling** and **weak scaling**. **Strong scaling** means that the size of the overall problem being solved remains constant as the number of PEs is increased. **Weak scaling** means that the size of the problem being solved *by each PE* remains constant as the number of PEs are increased.

For example, if we run a program with 1000 pieces of data, that program can be run on 2, 4, 8, or any number of PEs desired. This is **strong scaling** because the size of the overall problem remains constant at 1000 pieces of data as the number of PEs continues to increase. Generally, the number of PEs is doubled each time, such as 2, 4, 8, 16, etc. PEs.

**Weak scaling** would be if there were 1000 pieces of data *per PE*. If 2 PEs are used, there would be 2000 total pieces of data; if there were 4 PEs, there would be 4000 total pieces of data, and so on. The overall size of the problem *IS* changing as the number of PEs is increased, but the size of the problem per PE is *NOT*changing. It stays constant at 1000 pieces of data.

The intent of this lab is to learn more about scaling by running a computationally intensive **N-body model**. An **N-body model** involves interactions and movements of a given number, known as N, of objects or bodies. The bodies in the model start with certain initial positions and velocities. Then, at each time step, based on their positions, they exert forces on all other bodies in the model. This means that there are N\* (N – 1) total forces in each time step. Based on these forces, the acceleration of each body can be calculated. This determines its new velocity, which can be used to determine its new position.

In addition to looking at the different types of scaling, this lab will also look at continually increasing the number of PEs to understand the concept of “more is not always better” and analyze why that is the case.

The program used for this lab is an N-body program called GalaxSee, available at:

**http://shodor.org/petascale/materials/UPModules/NBody/**

In this model, forces are exerted between two bodies according to the equation below, where **F** is the force, **G** is the gravitational constant, **m1** is the mass of one body, **m2** is the mass of the other body, and **r** is the distance between them.

**F = (G \* m1 \* m2) / (r ^2)**

Multiple PEs can work on the problem at the same time if each is given a specific portion of the total number of bodies, or a section of the total work, to work with. In this way, each PE has a fraction of the total work to do. But, because each body must exert forces on all the other bodies, there will need to be communication between the PEs for the model to work. This communication can add extra runtime to the program, known as **communication overhead**.

To execute this lab successfully, you must download the GalaxSee code, copy it to the supercomputer you are using, compile it, and run it using various combinations of different numbers of PEs and different numbers of bodies. Keep track of the runtime data you collect in the spreadsheet you can download from the link provided below. Then, use an interactive graphing tool to plot the results of each run. This will allow you to look at both the strong and weak scaling of the program. All of these instructions will be walked through in detail below.

**Activity**

1. Login to the supercomputer or cluster that you are using
2. Download the GalaxSee code from GitHub:

**git clone https://github.com/aaronweeden/pi2018-tts<ENTER>**

1. Change to the code directory:

**cd pi2018-tts<ENTER>**

1. Compile the GalaxSee code:

**make<ENTER>**

For this exercise, you will be running what is known as a **parameter sweep** of GalaxSee. This means that you will be changing the combination of parameters for each run and observe the resulting runtime, covering every combination of parameters possible.

For each set of parameters seen in the table below, run the program and record the resulting runtime in the spreadsheet you can download here:

<https://docs.google.com/spreadsheets/d/1lpQExBj90IhnrRCXrgd4GHK96CbjJS-q-7eYKZleb0E/edit?usp=sharing>

Download your own copy so you can keep your results separate from everyone else.

|  |  |
| --- | --- |
| **Number of PEs** | **Number of Bodies** |
| 8 | 16000 |
| 16 | 16000 |
| 32 | 16000 |
| 64 | 16000 |
| 128 | 16000 |
| 8 | 8000 |
| 16 | 8000 |
| 32 | 8000 |
| 64 | 8000 |
| 128 | 8000 |
| 8 | 4000 |
| 16 | 4000 |
| 32 | 4000 |
| 64 | 4000 |
| 128 | 4000 |
| 8 | 2000 |
| 16 | 2000 |
| 32 | 2000 |
| 64 | 2000 |
| 128 | 2000 |

To change the parameters for each run, there is a **tts.pbs** batch script file for you to use. This submits the job for you and runs the job with the given parameters that you specify. Find this file and become familiar with it. The run command is given below. Replace the yellow highlighted portions with the parameters for the given run. If the number of PEs that you are requesting is more than the number of PEs on a single node, make sure you increase the number of nodes you are requesting. It is also recommended that you change the name of each job in the .pbs file each time to avoid confusing the output files.

**aprun -n Number of PEs ./galaxsee.exe Number of bodies<ENTER>**

You are going to run each of the runs 5 times. There is space in the spreadsheet for you to record all 5 runs for each combination of parameters. There is an **Overall** row at the bottom that averages all of the results for each set of parameters. Use this number when plotting your data.

Once you have all of your data recorded in your spreadsheet for all of your runs in the parameter sweep, build plots of the strong and weak scaling using the sample results below as a template. You will be generating plots with Shodor’s Interactive tool called Simple Plot, available at:

<http://www.shodor.org/interactivate/activities/SimplePlot/>

**Sample result: strong scaling**

**Key:**

* **redgraph=16000 bodies**
* **orangegraph=8000 bodies**
* **greengraph=4000 bodies**
* **bluegraph=2000 bodies**
* **Data pairs are (Number of PEs, runtime)**

redgraph

8, 119.845133

16, 64.581847

32, 32.634440

64, 31.426864

96, 23.476438

128, 18.253617

orangegraph

8, 29.700608

16, 15.096680

32, 7.890904

64, 7.929359

96, 6.313430

128, 6.153810

greengraph

8, 7.369270

16, 3.862636

32, 2.163885

64, 2.397110

96, 2.399140

128, 1.985066

bluegraph

8, 1.874488

16, 1.066026

32, 0.645710

64, 0.703931

96, 1.088413

128, 1.149805

**Sample result: weak scaling**

**Key:**

* **redgraph=250 bodies / PE**
* **orangegraph=500 bodies / PE**
* **greengraph=1000 bodies / PE**
* **Data pairs are (Number of PEs, runtime)**

redgraph

8, 1.874488

16, 3.862636

32, 7.890904

64, 31.426864

orangegraph

8, 7.369270

16, 15.096680

32, 32.634440

greengraph

8, 29.700608

16, 64.581847