

In class, we've seen that Chapel provides a high level parallel programming model for data parallelism, task parallelism, and nested parallelism. It also provides low-level control to reason about locality and load distribution. In this assignment, you will use Chapel to implement the Barnes-Hut algorithm. The key goal of this assignment is to get familiar with the parallel abstractions that Chapel provides. You are encouraged to explore the low-level performance tuning features of Chapel, but that is optional. You may work with a partner if you wish.

## 1 Algorithm

For details about the algorithm and various optimizations, refer to Assignment 5.

## 2 Parallel Programming Paradigm

**Data Parallelism** The Barnes-Hut algorithm will give you ample opportunities to exploit data parallelism as the same set of operations are applied on all bodies for force computation. You can use the built-in parallel operations such as reduce, scan, max and min to manipulate your data.

**Task Parallelism** Tree traversal operations will enable you to exploit task parallelism as well.

## 3 Setting up Chapel on CS machines

For this assignment, you will need to install Chapel on your CS account. Chapel can be installed by following the instructions at <http://chapel.cray.com/download.html>

## 4 Using Chapel

This is a good tutorial to learn about Chapel - <http://chapel.cray.com/tutorials.html>. The slides on *Base Language*, *Data Parallelism* and *Task Parallelism* should get you started for the assignment.

To compile your code, use the command - `chpl -o exec <source file>`

To run your code, use the command - `./exec --argname=<arg value>`

## 5 Input/Output/Parameters

As in Assignment 5, your program should accept the following four command line parameters:

1. number of iterations (argname=iterations)
2. timestep (argname=timestep)
3. input filename (argname=inputfile)
4. output filename (argname=outputfile)

The format of the input and output files and parameter values are the same as in Assignment 5.

You may assume that all initial velocities are 0 in the input file.

For simplicity, reset the acceleration of the body to 0 after each iteration.

Dimensions of the system: (-9999.0,-9999.0) to (9999.0,9999.0)

## 6 Test Inputs

Use the test inputs provided for Assignment 5 to verify your code and measure performance.

## 7 Performance

You are free to explore the various knobs provided by Chapel to improve performance. This, however, is not required for the assignment. For those interested in performance tuning, the following should get you started.

A *locale* in Chapel is an architectural unit on which the code runs. A locale's tasks have nearly uniform access to its local variables and other locale's variables are accessible at a price. Locales could be processors across workstations or nodes on an SMP system. You could optimize parameters for parallelism on a single locale or across locales.

**Per-locale Optimization** If you have a single locale to run your code, then you can control the single locale parallelism by tuning the following command line parameters - `dataParTasksPerLocale`, `dataParIgnoreRunningTasks` and `dataParMinGranularity`. More details about these can be found in `docs/README.executing` under your chapel directory. You can also control the maximum number of threads per locale by choosing `maxThreadsPerLocale` appropriately. More details on this parameter and querying the task pool dynamically in `docs/README.tasks`

**Multi-locale Optimization** To set up multiple locales, you will need to rebuild Chapel. The instructions for building for multiple locales are detailed in `docs/README.multilocale`. Once you have rebuilt, you can set the number of locales by using the `-nl` parameter. Chapel provides low-level control to manage the distribution of data and tasks across locales. The slides on `Locales and Domain Maps` explain programming and optimization across locales.

## What to Turn In

This assignment is due at 11:59pm on the due date. Use the **turnin** program to submit your solution, which should include the following:

1. A written report of the assignment in either plain text or PDF format. Please explain your approach towards implementing the algorithm in Chapel, and discuss any insights gained, etc.
2. In your report, also include a section that outlines your experience with the three parallel languages that you have tried over the duration of the course—Pthreads, MPI and Chapel. You can talk about about your insights on their programmability, parallel performance and performance portability.
3. Your source code, instructions to build it on CS machines.