ECE 6374 PARALLEL COMPUTATIONS

Project 2 Report

By

Bijon Sahu(1647781)

&

Manoj Kumar Cebol Sundarrajan(1620546)

Date

04/14/2018

Problem Statement

**Particle Simulation**

1. Divide the simulation time range into discrete time steps (dt) that are small enough to capture the changes.

2. For each particle in each time step,

a. Calculate the forces on that particle from all other particles in the system.

i. cutoff/100 < r < cutoff (cutoff = 0.01)

ii. Coulomb's law

b. From the total force, we get the acceleration.

i. Newton's second law (F=ma)

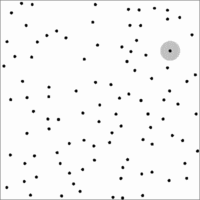
c. Move the particle based on the acceleration.

i. v = a \* dt

ii. x = v \* dt.

The purpose of this assignment is introduction to programming in shared and distributed memory models.

Your goal is to parallelize a toy particle simulator (similar particle simulators are used in [mechanics](http://www.thp.uni-duisburg.de/~kai/index_1.html), [biology](http://www.ks.uiuc.edu/Research/namd/), [astronomy](http://www.mpa-garching.mpg.de/gadget/clusters/index.html), etc.) that reproduces the behavior shown in the following diagram:



The range of interaction forces is limited as shown in grey for a selected particle. Density is set sufficiently low so that given *n* particles, only *O*(*n*) interactions are expected.

Suppose we have a code that runs in time *T = O(n)* on a single processor. Then we'd hope to run in time *T/p* when using *p* processors. We'd like you to write parallel codes that approach these expectations.

**All of the above results are from Opuntia cluster from CACDS,UH**

**MPI Algorithm**

Code gathers all neighboring particles on local node and then compares with local particles. So here we tried setting up the generic initiations **MPI\_init** , **MPI\_Comm\_size** and **MPI\_Comm\_rank** and then the allocation of generic resources **MPI\_Datatype** , **MPI\_Type\_contiguous** and **MPI\_Type\_commit** .Then we tried to distribute the particle via **MPI\_Bcast** from node 0 to other respective nodes . Although each worker has all particles, we only access particles within my\_bins\_start, my\_bins\_end.Then after we computed the local forces we used reduction commands **MPI\_Reduce** to move, but not rebin .Then finally we used **MPI\_Gather** to consolidate the scattered particles .

**Non-optimized MPI Code**

n = 5000, simulation time = 89.5586 seconds

Hello from compute-2-19.local

n = 5000, simulation time = 89.5831 seconds

Hello from compute-2-19.local

Hello from compute-2-20.local

n = 5000, simulation time = 44.8791 seconds

Hello from compute-2-19.local

Hello from compute-2-21.local

Hello from compute-2-20.local

Hello from compute-2-22.local

n = 5000, simulation time = 45.1239 seconds

Hello from compute-2-19.local

Hello from compute-2-20.local

Hello from compute-2-21.local

Hello from compute-2-23.local

Hello from compute-2-22.local

Hello from compute-2-31.local

n = 5000, simulation time = 33.1039 seconds

Hello from compute-2-19.local

Hello from compute-2-20.local

Hello from compute-2-21.local

Hello from compute-2-32.local

Hello from compute-2-23.local

Hello from compute-2-31.local

Hello from compute-2-22.local

Hello from compute-2-33.local

n = 5000, simulation time = 34.0556 seconds

Hello from compute-2-19.local

Hello from compute-2-20.local

n = 10000, simulation time = 178.895 seconds

Hello from compute-2-19.local

Hello from compute-2-20.local

Hello from compute-2-22.local

Hello from compute-2-21.local

n = 20000, simulation time = 779.657 seconds

Hello from compute-2-19.local

Hello from compute-2-21.local

Hello from compute-2-23.local

Hello from compute-2-20.local

Hello from compute-2-22.local

Hello from compute-2-31.local

n = 30000, simulation time = 1081.67 seconds

Hello from compute-2-19.local

Hello from compute-2-20.local

Hello from compute-2-22.local

Hello from compute-2-31.local

Hello from compute-2-21.local

Hello from compute-2-23.local

Hello from compute-2-32.local

Hello from compute-2-33.local

n = 60000, simulation time = 6824.32 seconds

**Strong scaling estimates are:**

    1.00    2.00    1.00    3.96    2.00    1.98    2.71    2.63    0.50    0.33 (**speedup**)

    1.00    1.00    1.00    0.99    1.00    0.50    0.45    0.33    0.25    0.33 (**efficiency**)    for

       1       2       1        4        2        4        6        8        2        1 **threads/processors**

**Average strong scaling efficiency:**    0.68

**Weak scaling estimates are:**

    1.00    0.33    0.50    0.64    0.64    0.25    0.15    0.15    0.11    0.08 (**efficiency**)    for

       1       2       1        4       2       4        6       8       2        1 **threads/processors**

**Average weak scaling efficiency:**    0.39

**Optimized MPI Code .**

n = 5000, simulation time = 0.901767 seconds

n = 5000, simulation time = 1.50998 seconds

n = 5000, simulation time = 0.763576 seconds

n = 5000, simulation time = 0.846376 seconds

n = 5000, simulation time = 1.21955 seconds

n = 5000, simulation time = 0.777148 seconds

n = 10000, simulation time = 1.51376 seconds

n = 20000, simulation time = 3.49092 seconds

n = 30000, simulation time = 11.3028 seconds

n = 60000, simulation time = 13.7605 seconds

**Strong scaling estimates are :**

    0.60    1.18    1.07    0.74    1.16 (**speedup**)

    0.60    0.59    0.27    0.12    0.15 (**efficiency**)    for

       1       2       4        6       8 **threads/processors**

**Average strong scaling efficiency:**    0.34

**Weak scaling estimates are :**

    0.60    0.60    0.26    0.08    0.07 **(efficiency)**    for

       1       2       4        6       8 **threads/processors**

**Average weak scaling efficiency:**    0.32

**Optimized Serial Algorithm**

Algorithm compares each particle with every other particle and checks for the interaction radius. We created bins and then search over nearby 8 bins and itself. Then we try checking items which are still inside the bin and store particles that have changed bin. These stored items are then removed from the current bin and put them into a new bin.

**Serial optimized**

n = 5000, simulation time = 0.895884 seconds

n = 10000, simulation time = 1.99829 seconds

n = 20000, simulation time = 4.5122 seconds

n = 40000, simulation time = 15.8825 seconds

**Serial code** is O (N ^ slope)

**Slope estimates** are: 1.157382 1.175065 1.815535

**Slope estimate** **for line fit is**: 1.361901

**Serial non-optimized**

n = 5000, simulation time = 86.8722 seconds

n = 10000, simulation time = 346.321 seconds

n = 20000, simulation time = 1389.85 seconds

n = 40000, simulation time = 5577.51 seconds

**Serial code** is O (N ^ slope).

**Slope estimates** are: 2.005978 1.998021 1.997886

**Slope estimate for line fit is**: 2.000368

**Optimized OpenMP**

Split the particles into p equal sized subsets and accesses all non-owned particles in the interaction check.We used **pragma omp parallel private** and **pragma omp master** to instantiate the master thread and then we reduced the function by using **pragma omp for reduction** (+:navg) **reduction**(+:davg) . Each thread has a seperate tmp vector and then we try checking items which are still inside the bin and store particles that have changed bin. These stored items are then removed from the current bin and put them into a new bin. Scan over all tmp vectors using one thread using **pragma omp master** and **put** them into a new bin. Using **pragma omp barrier to** wait for all threads (mostly the master) to finish.

**Openmp unoptimized**

n = 5000, simulation time = 78.1145 seconds

n = 5000,threads = 1, simulation time = 84.5535 seconds

n = 5000,threads = 2, simulation time = 45.3338 seconds

n = 5000,threads = 4, simulation time = 23.8226 seconds

n = 5000,threads = 6, simulation time = 16.4056 seconds

n = 5000,threads = 12, simulation time = 8.70638 seconds

n = 5000,threads = 18, simulation time = 5.86576 seconds

n = 5000,threads = 24, simulation time = 8.62614 seconds

n = 5000,threads = 32, simulation time = 6.8432 seconds

n = 10000,threads = 2, simulation time = 179.056 seconds

n = 20000,threads = 4, simulation time = 372.332 seconds

n = 30000,threads = 6, simulation time = 571.54 seconds

n = 60000,threads = 12, simulation time = 1172.93 seconds

n = 90000,threads = 18, simulation time = 1945.88 seconds

n = 120000,threads = 24, simulation time = 3275.23 seconds

n = 160000,threads = 32, simulation time = 5710.02 seconds

**Strong scaling estimates are :**

    0.92    1.72    3.28    4.76    8.97   13.32    9.06   11.41 (**speedup**)

    0.92    0.86    0.82    0.79    0.75    0.74    0.38    0.36 (**efficiency**)    for

       1       2       4        6      12      18      24      32 **threads/processors**

**Average strong scaling efficiency:**    0.70

**Weak scaling estimates are :**

    0.92    0.44    0.21    0.14    0.07    0.04    0.02    0.01 **(efficiency)**for

       1       2       4        6      12      18      24       32 **threads/processors**

**Average weak scaling efficiency:**    0.23

**Openmp optimized**

n = 5000, simulation time = 0.865415 seconds

NThreads: 1, n = 5000, simulation time = 0.916516 seconds

NThreads: 2, n = 5000, simulation time = 0.483523 seconds

NThreads: 4, n = 5000, simulation time = 0.289815 seconds

NThreads: 6, n = 5000, simulation time = 0.203387 seconds

NThreads: 12, n = 5000, simulation time = 4.22424 seconds

NThreads: 18, n = 5000, simulation time = 5.33981 seconds

NThreads: 24, n = 5000, simulation time = 0.703145 seconds

NThreads: 32, n = 5000, simulation time = 0.759022 seconds

NThreads: 2, n = 10000, simulation time = 0.963991 seconds

NThreads: 4, n = 20000, simulation time = 1.12886 seconds

NThreads: 6, n = 30000, simulation time = 1.14804 seconds

NThreads: 12, n = 60000, simulation time = 6.81033 seconds

NThreads: 18, n = 90000, simulation time = 7.24459 seconds

NThreads: 24, n = 120000, simulation time = 3.59766 seconds

NThreads: 32, n = 160000, simulation time = 4.61634 seconds

**Strong scaling estimates are :**

    0.94    1.79    2.99    4.26    0.20    0.16    1.23    1.14 (**speedup**)

    0.94    0.89    0.75    0.71    0.02    0.01    0.05    0.04 (**efficiency**)    for

       1       2       4        6       12      18      24       32 **threads/processors**

**Average strong scaling efficiency:**    0.43

**Weak scaling estimates are :**

    0.94    0.90    0.77    0.75    0.13    0.12    0.24    0.19 (**efficiency**)    for

       1       2        4        6      12      18       24       32 **threads/processors**

**Average weak scaling efficiency:**    0.50