CS 301
High-Performance Computing

Lab 6

Problem 2: How does the interaction between charged particles influence their movement?

Group: 11

Dweej Pandya (202101432)
Aarsh Bhavsar (202101474)

March 28, 2024

Contents

1 Introduction

This problem includes calculating the final position of the charged particles under coulombic force. The problem's complexity is examined using LAB207 PC and HPC Cluster.

We can calculate the distance by,

$$\Delta d = v\Delta t + \frac{1}{2}a(\Delta t)^2 \tag{1}$$

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 4
- On-line CPU(s) list: 0-3
- Thread(s) per core: 1
- Core(s) per socket: 4
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 60
- Model name: Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz
- Stepping: 3
- CPU MHz: 799.992
- CPU max MHz: 3700.0000
- CPU min MHz: 800.0000
- BogoMIPS: 6584.55
- Virtualization: VT-x

- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 6144K
- NUMA node0 CPU(s): 0-3
- Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm epb invpcid_single tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts

2.2 Hardware Details for HPC Cluster (Node gics1)

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 16
- On-line CPU(s) list: 0-15
- Thread(s) per core: 1
- Core(s) per socket: 8
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 63
- Model name: Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
- Stepping: 2
- CPU MHz: 1288.726
- BogoMIPS: 5204.73
- Virtualization: VT-x

- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 20480K
- NUMA node0 CPU(s): 0-7
- NUMA node1 CPU(s): 8-15

3 Problem

3.1 Brief description of the problem

A space contains n charged particles with their charge and initial x and y coordinates given. We need to determine their final positions after 10 seconds.The movement of these particles is influenced by the interactions between them due to their charges. Assuming only coulomb's force is in action.

3.1.1 Algorithm description

We are given the initial postions (x, y) along with the charge q for n particles. We need to calculate the force and hence the valocity and hence the final postion of the particles after 10 seconds, with $\Delta t = 0.0001s$.

The formulas for the same are given below:

$$F_i = k\frac{|q1q2|}{{r_i}^2} \tag{2}$$

$$a_i = \frac{F_i}{m} \tag{3}$$

$$v_i = v_{i-1} + a_{i-1}t \tag{4}$$

$$d_i = v_{i-1}t + \frac{1}{2}a_{i-1}t^2 \tag{5}$$

3.2 The complexity of the algorithm (serial)

The number of subintervals used in the calculation for Problem is n, and the serial complexity is O(n^2). This indicates that the time it takes for the method to finish will grow quadratically with n as the number of subintervals does. Because the method only must be run twice through the data to calculate the distance, this complexity is considered as quadratic. An algorithm's serial complexity gives an estimate of how long it will take to execute as the size of the input grows.

3.3 The complexity of the algorithm (parallel)

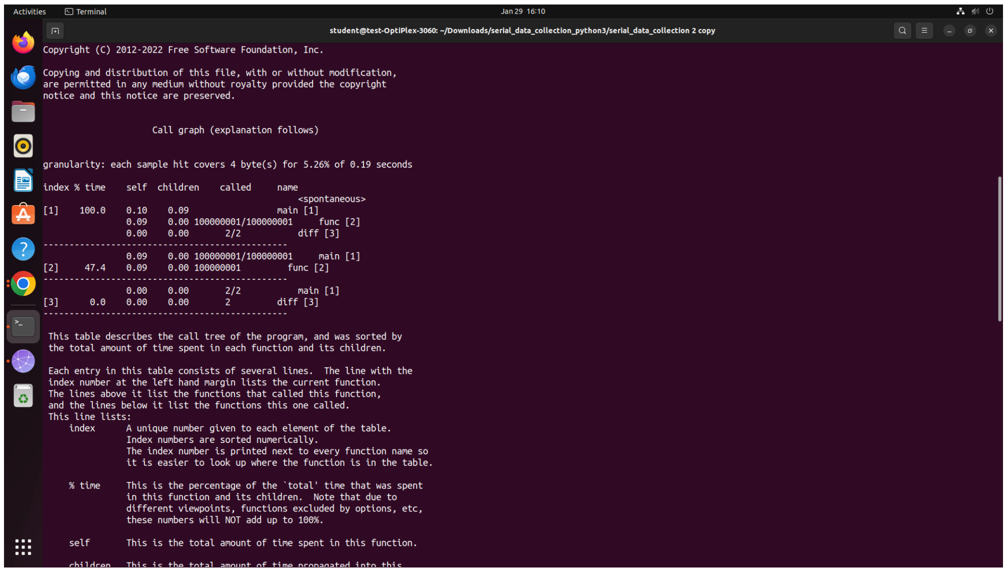
The parallelism reduces this complexity by dividing the work among t threads. Each thread computes a partial sum over a subset of the intervals. The time complexity of this parallel algorithm is $O(n^2/t)$, assuming that the work is evenly distributed among the threads and there are no overheads due to synchronization or communication.

3.4 Profiling information (using gprof)

Please note that profiling using gprof was done on codes without optimizing them as it was way faster and spontaneous and gprof was not showing any output.

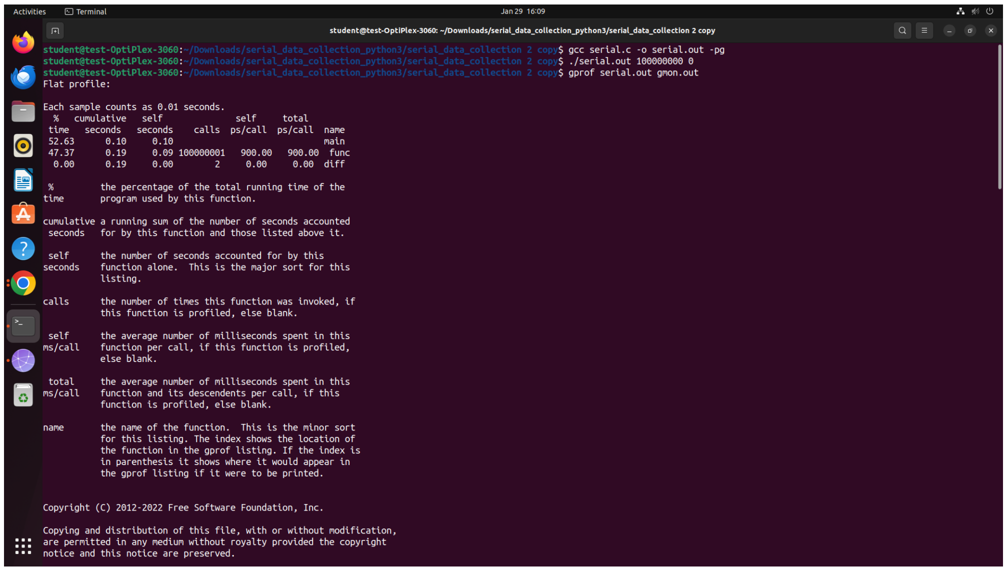
3.4.1 Profiling using LAB207 PCs (with gprof)

The screenshots of profiling using the LAB207 PCs are given below for Q1 (both from the terminal and from the text file).


Activities
Terminal
Jan 29 16:10
student@test-OptiPlex-3060: ~/Downloads/serial_data_collection_python3/serial_data_collection 2 copy

```
Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 5.26% of 0.19 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.10    0.09                 main [1]
                0.09    0.00 100000001/100000001     func [2]
                0.00    0.00       2/2           diff [3]
-----------------------------------------------
                0.09    0.00 100000001/100000001     main [1]
[2]     47.4    0.09    0.00 100000001         func [2]
-----------------------------------------------
                0.00    0.00       2/2           main [1]
[3]      0.0    0.00    0.00       2         diff [3]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

     % time     This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
                these numbers will NOT add up to 100%.

     self       This is the total amount of time spent in this function.

     children   This is the total amount of time propagated into this
```

Figure 1: Screenshot of terminal from LAB207 PC

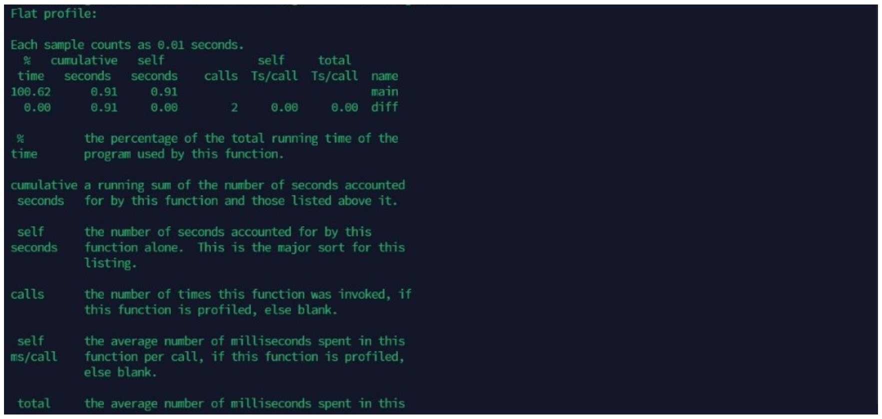


```
student@test-OptiPlex-3060:~/Downloads/serial_data_collection_python3/serial_data_collection 2 copy$ gcc serial.c -o serial.out -pg
student@test-OptiPlex-3060:~/Downloads/serial_data_collection_python3/serial_data_collection 2 copy$ ./serial.out 100000000 0
student@test-OptiPlex-3060:~/Downloads/serial_data_collection_python3/serial_data_collection 2 copy$ gprof serial.out gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  ps/call  ps/call  name    
 52.63      0.10     0.10                             main
 47.37      0.19     0.09 100000001   900.00   900.00  func
  0.00      0.19     0.00        2     0.00     0.00  diff

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

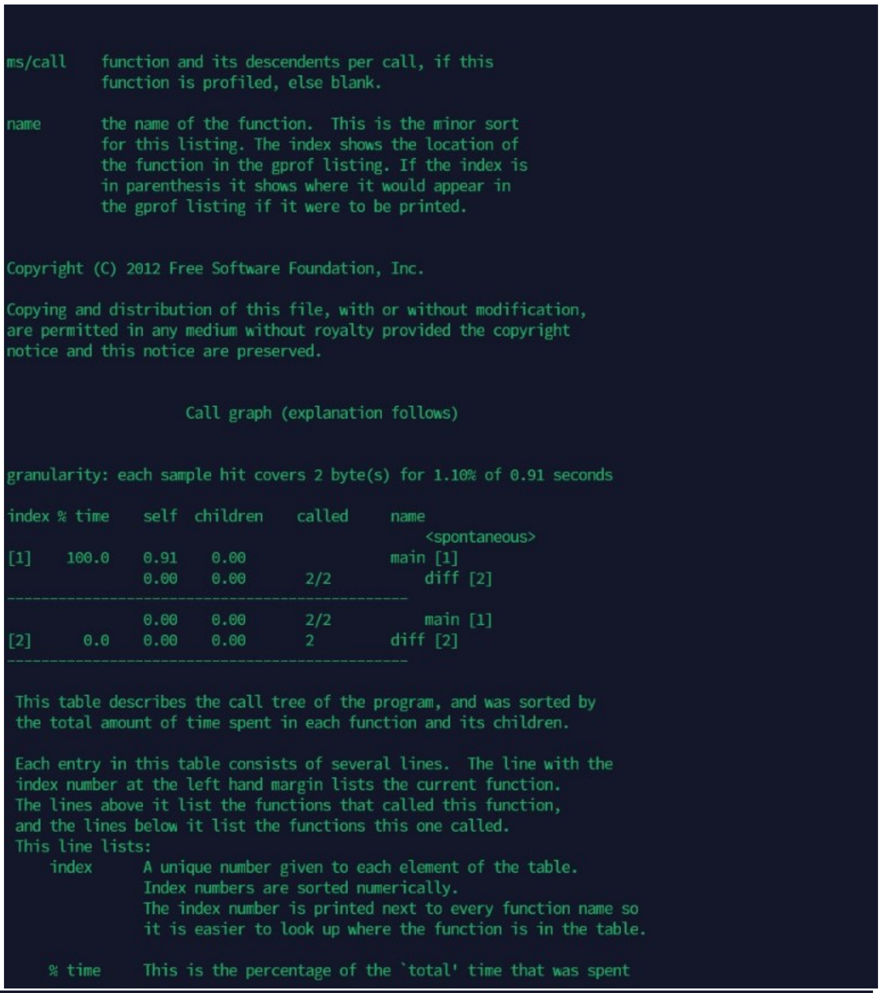
Figure 2: Screenshot of the gprof output(taken from a text file)

3.4.2 Profiling using HPC Cluster (with gprof)

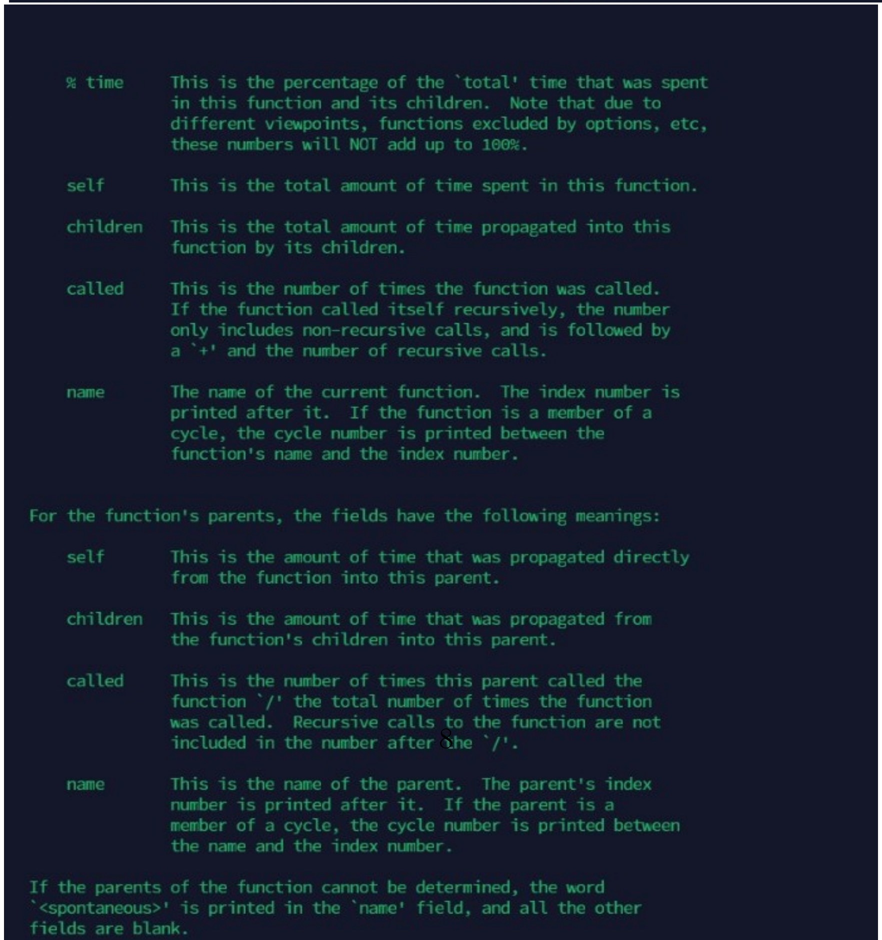
The screenshots of profiling using the HPC Cluster are given below for Q2 (both from the terminal and from the text file).



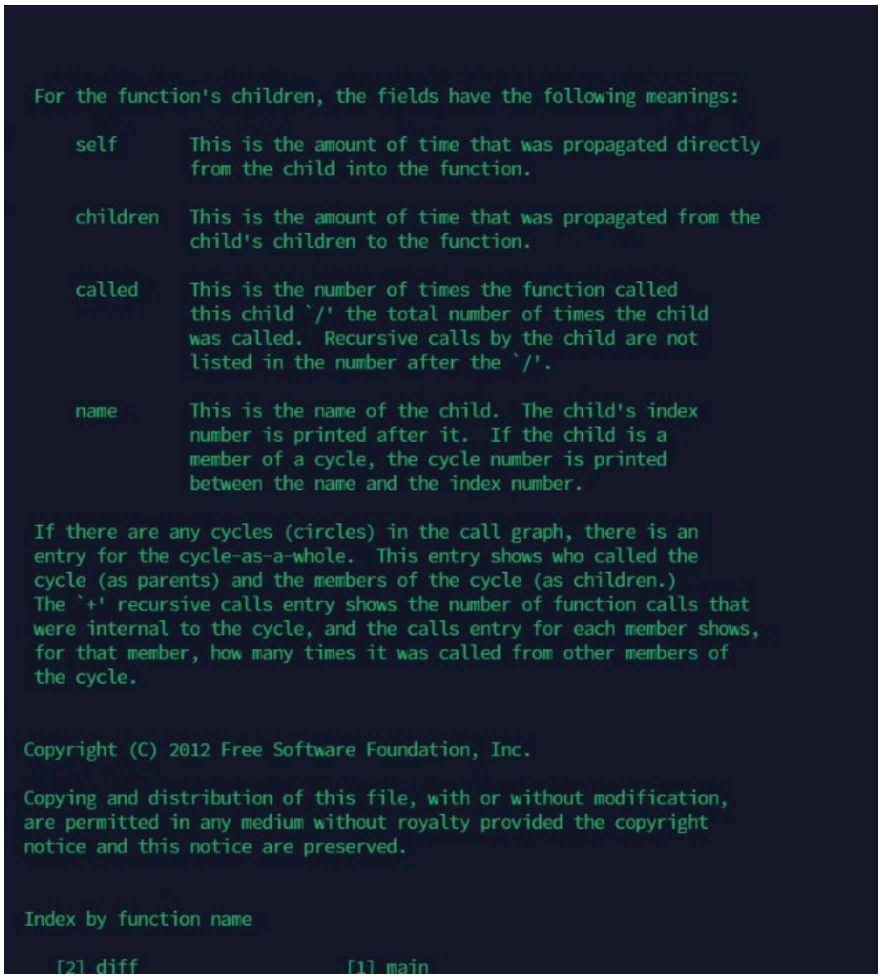
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  Ts/call  Ts/call  name    
100.62      0.91     0.91                             main
  0.00      0.91     0.00        2     0.00     0.00  diff

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
```



```
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 1.10% of 0.91 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.91    0.00                 main [1]
                0.00    0.00       2/2           diff [2]
-----------------------------------------------
                0.00    0.00       2/2           main [1]
[2]      0.0    0.00    0.00       2         diff [2]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index      A unique number given to each element of the table.
                Index numbers are sorted numerically.
                The index number is printed next to every function name so
                it is easier to look up where the function is in the table.

     % time     This is the percentage of the `total' time that was spent
```



```
     % time     This is the percentage of the `total' time that was spent
                in this function and its children.  Note that due to
                different viewpoints, functions excluded by options, etc,
                these numbers will NOT add up to 100%.

     self       This is the total amount of time spent in this function.

     children   This is the total amount of time propagated into this
                function by its children.

     called     This is the number of times the function was called.
                If the function called itself recursively, the number
                only includes non-recursive calls, and is followed by
                a `+' and the number of recursive calls.

     name       The name of the current function.  The index number is
                printed after it.  If the function is a member of a
                cycle, the cycle number is printed between the
                function's name and the index number.


 For the function's parents, the fields have the following meanings:

     self       This is the amount of time that was propagated directly
                from the function into this parent.

     children   This is the amount of time that was propagated from
                the function's children into this parent.

     called     This is the number of times this parent called the
                function `/' the total number of times the function
                was called.  Recursive calls to the function are not
                included in the number after the `/'.

     name       This is the name of the parent.  The parent's index
                number is printed after it.  If the parent is a
                member of a cycle, the cycle number is printed between
                the name and the index number.

 If the parents of the function cannot be determined, the word
 `<spontaneous>' is printed in the `name' field, and all the other
 fields are blank.
```



```
 For the function's children, the fields have the following meanings:

     self       This is the amount of time that was propagated directly
                from the child into the function.

     children   This is the amount of time that was propagated from the
                child's children to the function.

     called     This is the number of times the function called
                this child `/' the total number of times the child
                was called.  Recursive calls by the child are not
                listed in the number after the `/'.

     name       This is the name of the child.  The child's index
                number is printed after it.  If the child is a
                member of a cycle, the cycle number is printed
                between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.


Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Index by function name

   [2] diff                    [1] main
```

Figure 3: Screenshot of terminal from HPC Cluster

3.5 Optimization Strategy

The code uses OpenMP to parallelize the for loop that computes the sum of function evaluations:

```
    #pragma omp parallel for private(j)
for (i = 0; i < PARTICLES; i++) {
forces[i][0] = 0.0;
forces[i][1] = 0.0;

    for ( j = 0; j < PARTICLES; j++) {

    if (i != j) {
double dx = particles[j].x - particles[i].x;
double dy = particles[j].y - particles[i].y;
double r = sqrt(dx * dx + dy * dy);
double forceMagnitude = (k * particles[i].charge * particles[j].charge) / (r * r);

forces[i][0] += forceMagnitude * (dx / r); // Force in x-direction
forces[i][1] += forceMagnitude * (dy / r); // Force in y-direction
}
}
}
```

The #pragma omp parallel for directive instructs the compiler to generate multi-threaded code for this loop. The private(i) clause specifies that the variable i should be private to each thread. This means that each thread has its own copy of this variable and can modify it independently without interfering with other threads.

3.6 Graph of Number of Processors vs Runtime

3.6.1 Graph of Number of Processors vs Algorithm Runtime for LAB207 PC

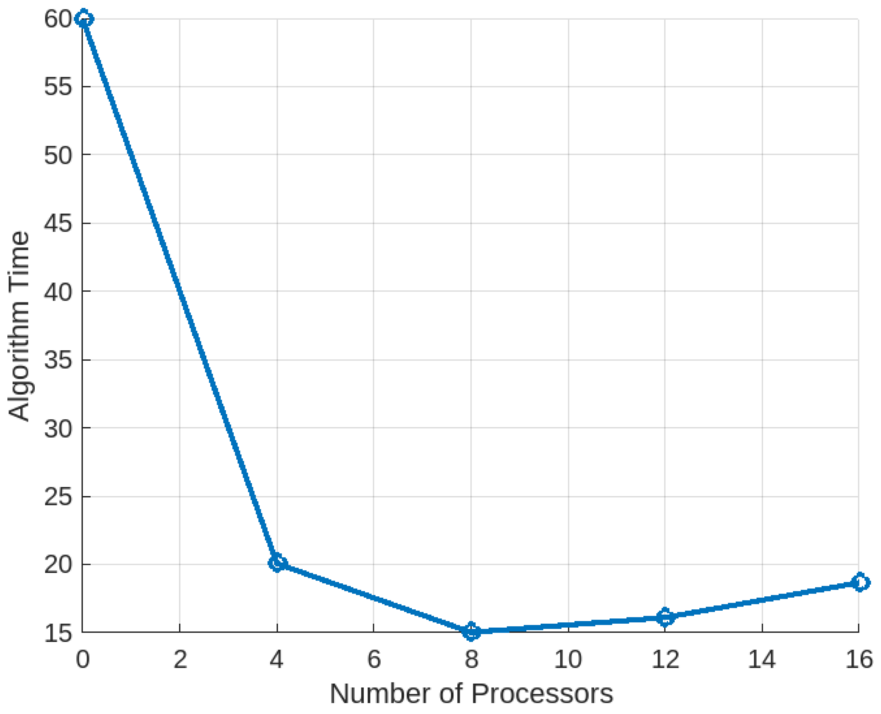


Figure 4: Algorithm execution time vs Number of Processors

3.6.2 Graph of Number of Processors vs Total Runtime for LAB207 PC

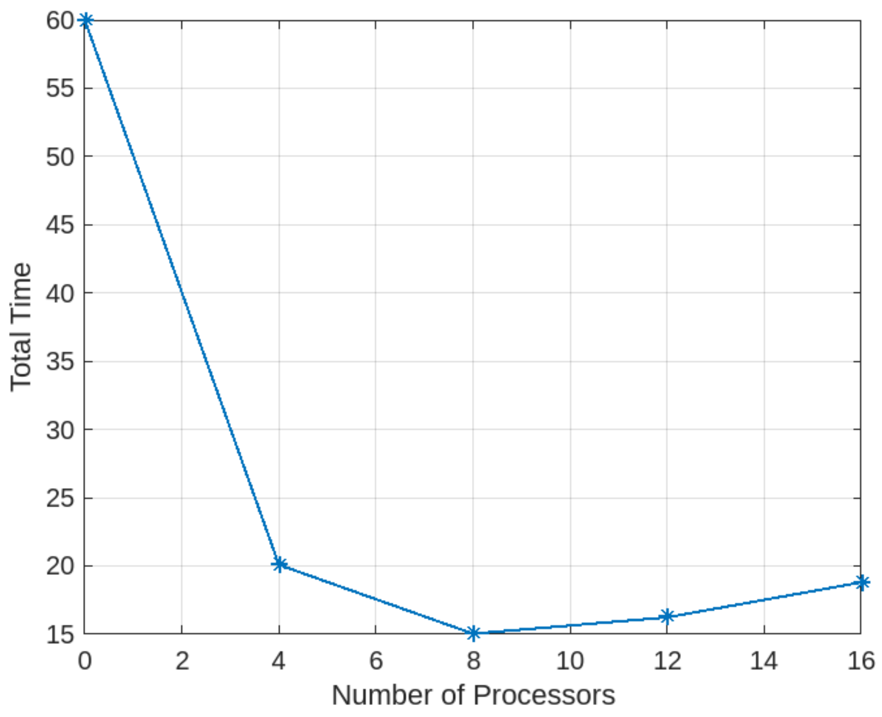


Figure 5: Total execution time vs Number of Processors

3.6.3 Graph of Number of Processors vs Algorithm Runtime for Cluster

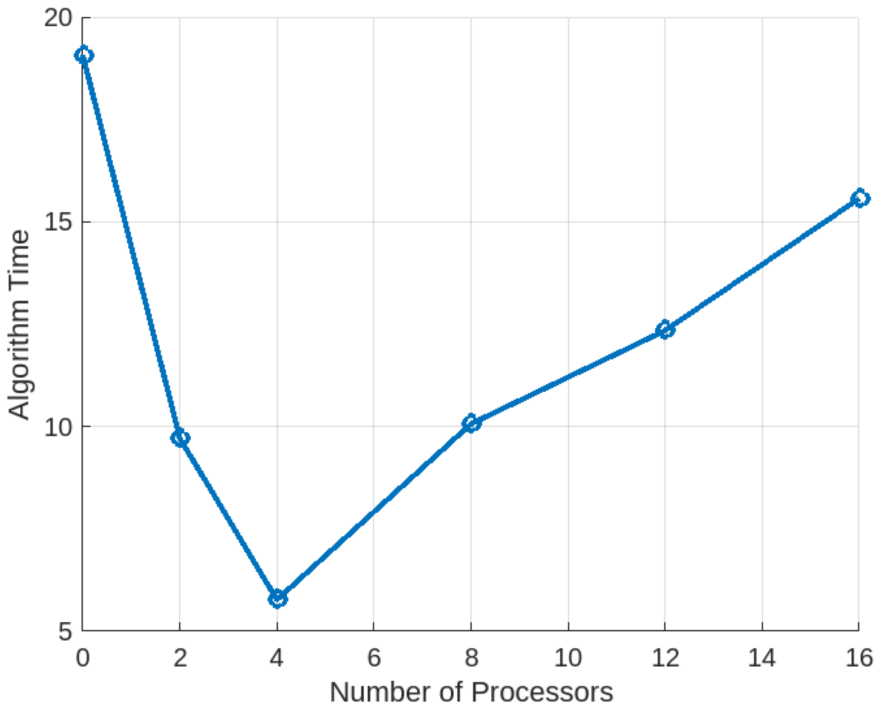


Figure 6: Algorithm execution time vs Number of Processors

3.6.4 Graph of Number of Processors vs Total Runtime for Cluster

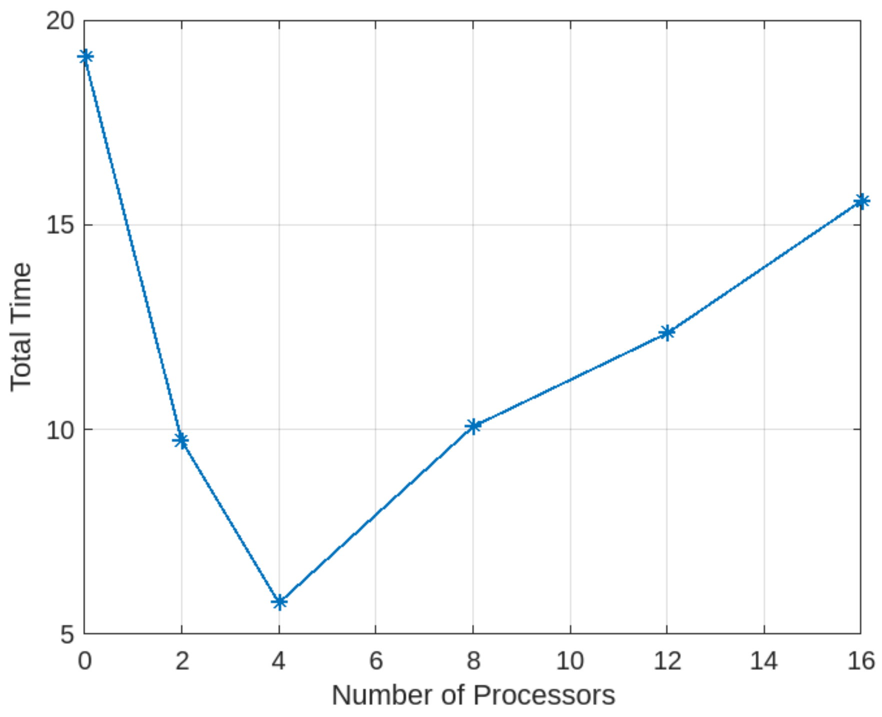


Figure 7: Total execution time vs Number of Processors

4 Conclusions

- We can observe that we get minimum time for 8 processors in Lab207PC.
- We can observe that we get minimum time for 4 processors in Cluster.