# DEMOGRAPHIC CLUSTERING OF PEPTIDE FRAGMENTS USING PARALLEL PROGRAMMING APPROACH

**Aman Sharma**

*Project Student, Indian Institute of Science, Bangalore,*

*Supercomputer Education and Research Centre*

*M. Tech Scholar CSE Student, School of Computing Science & Engineering,*

*Vellore Institute of Technology, Vellore, Tamilnadu, (India)*

## ABSTRACT

*An existing algorithm that is demographic clustering of peptide fragments that work in serial manner. Parallel programming approach used to reduce the delay and increased the efficiency of the algorithm. We thus show that the parallel programming can provide better results. Because serial processing works on single core that's why it takes more time if we have large amount of data and files at that time parallel cores can easily split the load among the processors and provide high efficiency. In this paper, our concept representing hash table modification and clustering using parallel programming. As the domain, a perl implementation dealing with increase the efficiency of the algorithm and reducing the time delay. The proposed work shows the efficiency difference between disk and RAM.*

***Keyword: Demographic Clustering, Peptide Fragments and Parallel Programming.***

## BACKGROUND

Program is some set of instructions. Process is program is in execution. During last year's performance is becoming the major issue. Today, almost everything is our life has a connection with how to reduce delay and increase high performance of the system. The performance has become one of the most important platforms for research in computer science field. The parallel programming plays the major role in terms of efficiency and performance.

Parallel programming is one major area to increase the performance by split the workload among all the threads. Parallel programming is running program for utilizing all the core and try to divide equal workload among all the processes.

In case of threads of execution is the smallest sequence of programmed instructions that can be managed independently by an OSS. OSS stands for operating system scheduler. Some regions behind for using threads like increase performance which is easy method to take advantage of multi-core, better utilization which means reduce latency and efficient data sharing which is sharing data through memory more efficient than message-passing. There is risks increase complexity of application, difficult to debug (data races, deadlocks etc.). Modern operating system load programs as process like resource holder and execution. A process starts executing at its entry point as a thread. Threads can create other threads within the process.

Concurrency means two or more threads are in progress at the same time. Parallelism means two or more threads are executing at same time.

## I. INTRODUCTION

Demographic clustering using peptide fragment problem based on the bioinformatics and computer science field with parallel programming language. The existing work of demographic clustering using peptide fragment was based on the serial manner. So that is why that was not giving the satisfactory performance. This work was taking more time and delay that was major problem for those people who are working in demographic clustering using peptide fragment topic. The problem behind demographic clustering using peptide fragment was based on serial manner. We have proposed the parallel version demographic clustering using peptide fragment that is more efficient than existing serial manner. The proposed work increased the efficiency of demographic clustering using peptide fragments with parallel programming so more fragment will be work fast and provide better results compare to serial version.

## II. PARALLEL PROGRAMMING APPROACH

The Demographic clustering using peptide fragments with parallel programming needed because increasing the efficiency of running fragments. Parent process forks child processes. Each child process reads a set of files and creates local hash tables. Then each child picks ($FR/Number of child processes) random fragments from local hash tables and sends to parent process. The parent adds the fragments received from the child processes and creates a final hash table which will have $FR entries finally.

## III. SEQUENTIAL VS. PARALLEL APPROACH

### 3.1 Sequential Algorithm for Demographic Clustering Of Peptide Fragments

The sequential algorithm for demographic clustering of peptide fragments has four inputs like a sequence of pairs of angles (phi_1, psi_1), (phi_2, and psi_2) and so on, fragment length FL, distance threshold R and angle difference ANG.

Firstly, a sequence of overlapping fragments, of length FL, is created from the sequence of pairs of angles: F_1, F_1.

Then, in two phases, the algorithm clusters these fragments such that each fragment in a cluster is within a distance R and angle ANG from the center or average of the cluster.

In the first phase, the algorithm picks each fragment F_k and checks each existing cluster center C_L as to whether they satisfy the distance and angle conditions, and if yes, the algorithm adds the fragment to the cluster and computes the new center, otherwise a new cluster is created with F_k as its first member.

In the second phase, the algorithm checks each fragment F_k as to whether it still satisfies the conditions with respect to the center of its cluster. If it does not, then F_k is removed from its cluster, and center of that cluster is recomputed. Then, a search is performed over the rest of the clusters to see if F_k can be accommodated within any of them. If such a cluster is found, then F_k is put in that cluster and its center is recomputed. Otherwise, a new cluster is created with F_k as its first member.

The second phase of the algorithm is repeated until the number of rejects in a round is less than or equal to the number of rejects in the previous round.

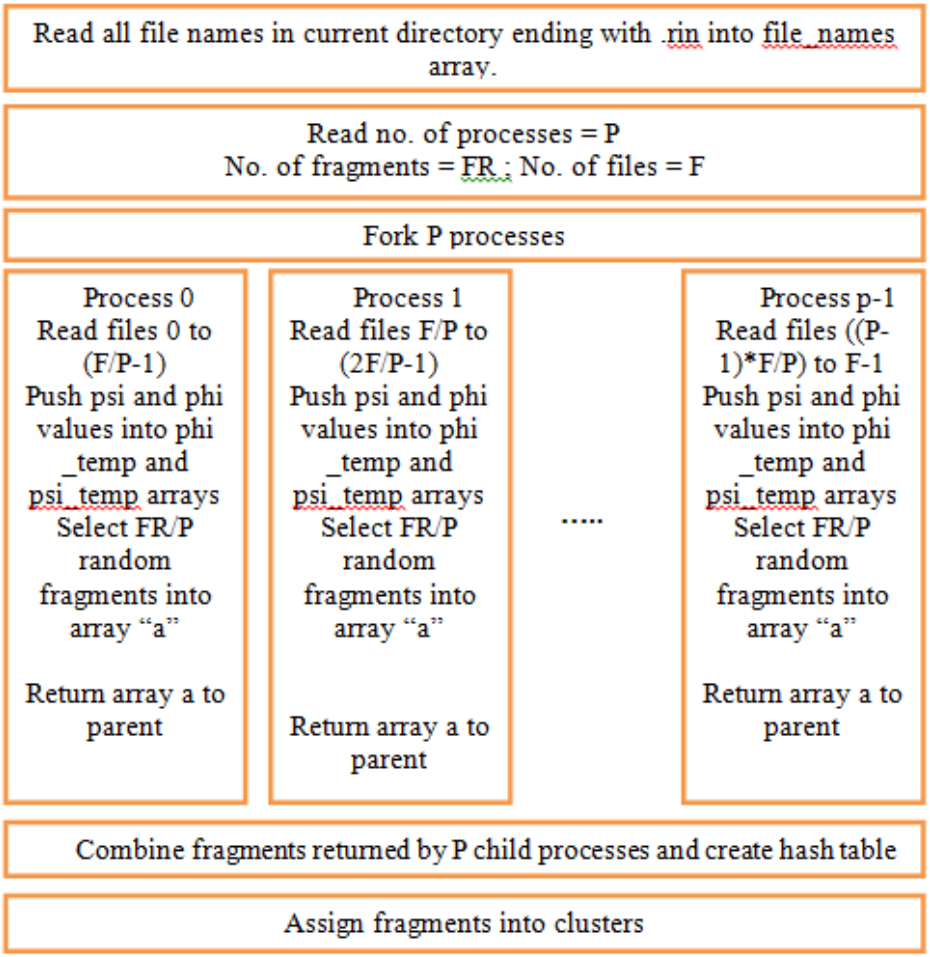
### 3.2 Parallel Algorithm for Demographic Clustering Of Peptide Fragments

Similarly the parallel algorithm for demographic clustering of peptide fragments has four inputs like a sequence of pairs of angles (phi_1, psi_1), (phi_2, and psi_2) and so on, fragment length FL, distance threshold R and angle difference ANG. In parallel algorithm for demographic clustering of peptide fragments has component's read all file names in current directory ending with .rin into file_names array, Read no. of processes = P, fork P processes, for starting process process 0 read files 0 to (F/P-1), push psi and phi values into phi _temp and psi_temp arrays, select FR/P random fragments into array "a", return array a to parent, same apply for all the process. Combine fragments returned by P child processes and create hash table and Assign fragments into clusters. Parent process forks child processes. Each child process reads a set of files and creates local hash tables. Then each child picks ($FR/Number of child processes) random fragments from local hash tables and sends to parent process. The parent adds the fragments received from the child processes and creates a final hash table which will have $FR entries finally.

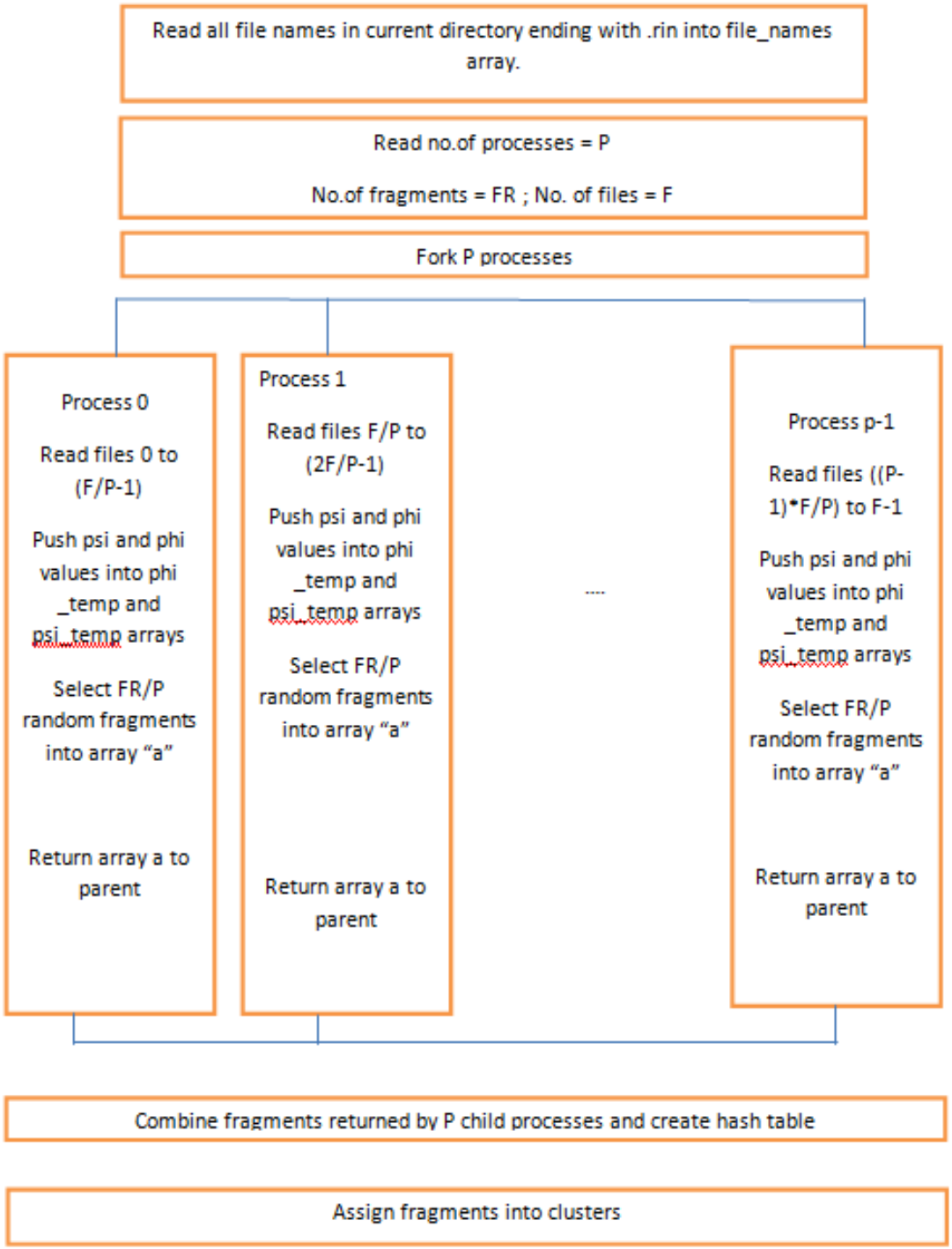
All clusters should be visible to all processes. Otherwise, each process would have its own set of clusters some of which would overlap partially with clusters present in other processes. After parallel clustering, the parent process should do additional processing to merge clusters formed by the child processes and reassign fragments.

### 3.3 Design of Parallel Architecture

Parallel algorithm architecture is a conceptual model that defines the structure, behavior and more views of the system.



The above represents read all file names in current directory ending with .rin into file_names array, read no. of processes = P, No. of fragments = FR ; No. of files = F, Fork P processes, Process 0 read files 0 to (F/P-1), Push psi and phi values into phi _temp and psi_temp arrays, select FR/P random fragments into array "a", return array a to parent, combine fragments returned by P child processes and create hash table and assign fragments into clusters.



Reading files to create fragment list is the most time consuming step in the serial code. Serial fragment list creation takes around 6seconds compared to around 1 second taken by clustering step. So by parallelizing fragment list creation step, the most time consuming part of the code has been parallelized. Also fragment list creation step is readily parallelizable compared to clustering since there is no synchronization needed between processes for fragment list creation. When clustering step is parallelized, the parent processes has to do additional work in combining the clusters returned by child processes in addition to combining the fragment list. Clusters returned by the child processes might overlap and hence the parent has to do additional work to reassign fragments to correct clusters. Clustering has less parallelism and more overhead. So parallelizing it doesn't improve the overall time.

## IV. RESULT ANALYSIS

The following results are based on hash table modification that shows the difference between disk time and RAM time. The parallel hash table code is also available based on the request

### 4.1 Disk Results Table

| Cores | total time | Hash time |
|---|---|---|
| 1 | 8.7 | 5.88 |
| 2 | 5.62 | 2.94 |
| 3 | 4.36 | 2.06 |
| 4 | 3.81 | 1.61 |
| 5 | 4.06 | 1.83 |
| 6 | 3.44 | 1.55 |
| 7 | 3.62 | 1.57 |
| 8 | 3.68 | 1.4 |

### 4.2 Ram Results Table

| Cores | Total time | Hash time |
|---|---|---|
| 1 | 7.34 | 5.79 |
| 2 | 4.99 | 2.95 |
| 3 | 4.34 | 2.05 |
| 4 | 4.06 | 1.82 |
| 5 | 3.52 | 1.84 |
| 6 | 3.36 | 1.56 |
| 7 | 3.27 | 1.49 |

| | | |
|---|---|---|
| 8 | 3.42 | 1.4 |

### 4.3 Disk Results with Different Cores

| Table 1. core 2 | Fragment length | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of Fragments | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 1000 | 4.76 | 6.85 | 6.34 | 8.06 | 8.96 | 9.57 | 10.32 |
| 2000 | 11.39 | 14.04 | 15.53 | 15.14 | 25.59 | 23.37 | 25.6 |
| 4000 | 23.71 | 54.83 | 56.02 | 93.64 | 68.37 | 90.85 | 85.09 |
| 6000 | 43.48 | 110.2 | 160.01 | 118.07 | 171.95 | 199.57 | 176.13 |
| 8000 | 102.34 | 170.45 | 188.97 | 238.95 | 351.92 | 212.38 | 309.43 |

| Table 2. core 4 | Fragment length | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of Fragments | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 1000 | 3.53 | 4.88 | 5.45 | 6.17 | 6.75 | 7.66 | 8.33 |
| 2000 | 8.41 | 11.25 | 14.58 | 16.59 | 21.31 | 22.82 | 22.38 |
| 4000 | 26.31 | 47.61 | 67.42 | 68.64 | 79.81 | 107.98 | 61.64 |
| 6000 | 53.21 | 77.022 | 91.18 | 118.48 | 175.32 | 196.152 | 173.55 |
| 8000 | 83.23 | 127.99 | 182.67 | 373.29 | 343.02 | 334.91 | 376.26 |

| Table 3. core 8 | Fragment length | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of Fragments | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 1000 | 3.26 | 4.37 | 5.22 | 5.94 | 6.45 | 7.17 | 7.68 |
| 2000 | 8.28 | 13.09 | 13.53 | 13.3 | 18.69 | 25.22 | 22.28 |
| 4000 | 1 | 45.8 | 43.7 | 55.69 | 77.09 | 85.62 | 96.41 |
| 6000 | 39.63 | 86.14 | 107.9 | 169.04 | 167.97 | 119.03 | 165.4 |

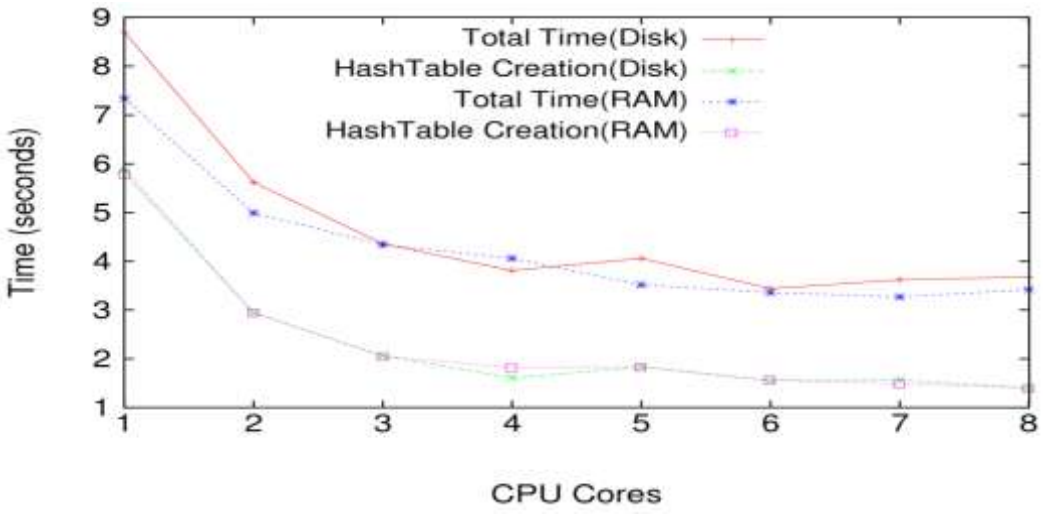
### 4.4 Ram Results with Different Cores

| Table 4. core 8 | Fragment length | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of Fragments | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 1000 | 5.91 | 6.7 | 6.97 | 7.96 | 8.85 | 9.48 | 10.31 |
| 2000 | 10.62 | 14.58 | 21.03 | 15.55 | 17.43 | 23.17 | 20.33 |
| 4000 | 24.93 | 60.28 | 55.47 | 69.18 | 65.34 | 77.73 | 100 |
| 6000 | 40.48 | 107.11 | 115.44 | 140.3 | 170.8 | 233.25 | 213.59 |
| 8000 | 85.86 | 167.09 | 224.19 | 250.67 | 297.24 | 332.38 | 369.3 |

| Table 5. core 8 | Fragment length | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of Fragments | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 1000 | 3.85 | 5.1 | 5.42 | 6.15 | 7.02 | 7.39 | 7.7 |
| 2000 | 8.26 | 10.97 | 22.76 | 20.12 | 22.7 | 22.7 | 23.06 |
| 4000 | 21.82 | 38.77 | 54.23 | 56.03 | 63.44 | 88.72 | 77.94 |
| 6000 | 55.68 | 101.94 | 155.62 | 190.47 | 197.46 | 189.85 | 212.02 |
| 8000 | 81.07 | 163.78 | 217.54 | 195.5 | 281.05 | 389.88 | 516.04 |

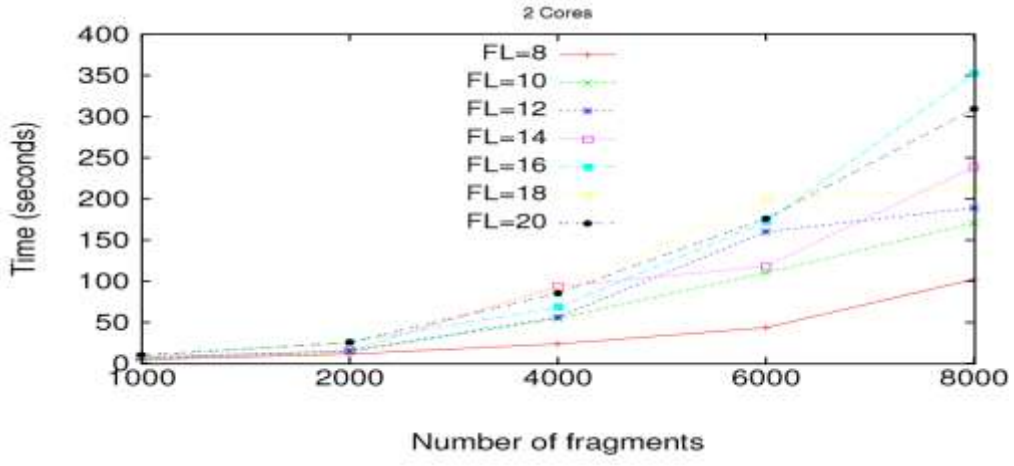
| Table 5. core 8 | Fragment length | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of Fragments | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 1000 | 3.62 | 5.08 | 5.29 | 5.96 | 6.54 | 7.14 | 7.54 |
| 2000 | 8.43 | 14.85 | 19.63 | 12.9 | 15.55 | 25.18 | 18.1 |
| 4000 | 28.75 | 53.11 | 43.55 | 94.32 | 79.56 | 55.67 | 97.63 |
| 6000 | 53.64 | 87.12 | 112.31 | 139.11 | 165.13 | 154.78 | 207.66 |
| 8000 | 64.35 | 118.03 | 183 | 332.11 | 397.41 | 202.18 | 289.7 |

### 4.5 Total Results Representation with Ram and Disk

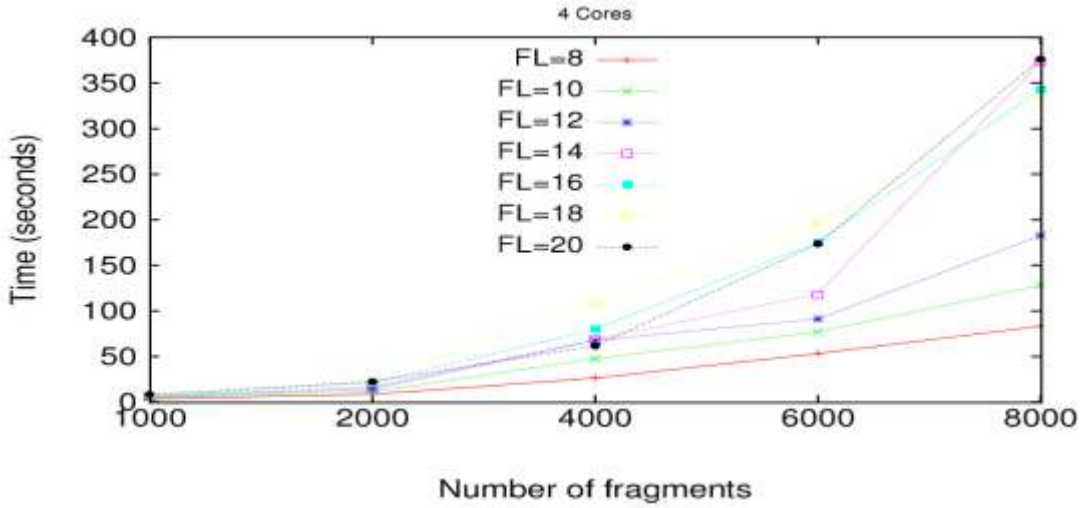



**GRAPH 1: DISK& RAM RESULTS TABLE**

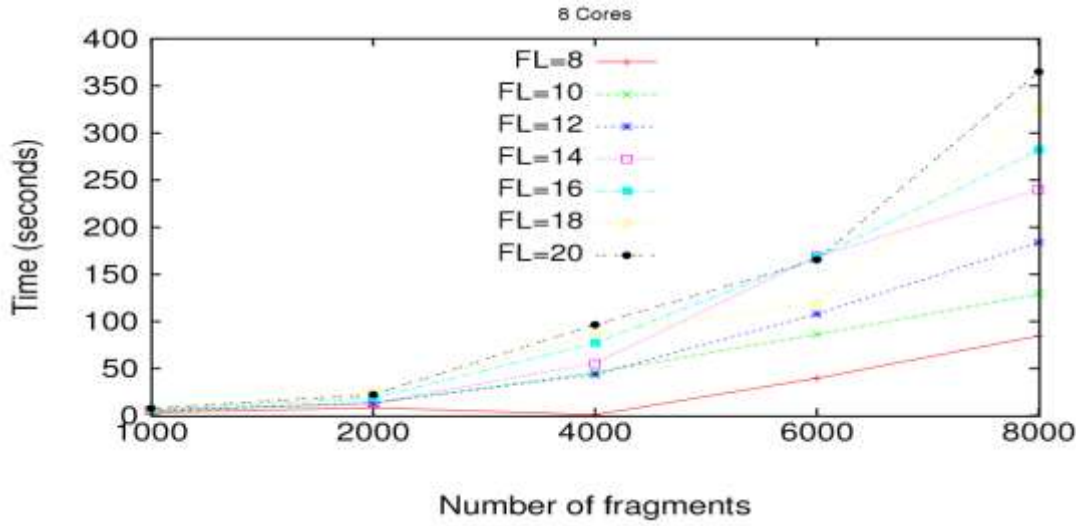
### 4.6 Disk Results with Different Cores and Graph




**GRAPH 2: DISK RESULTS TABLE WITH 2 CORES**

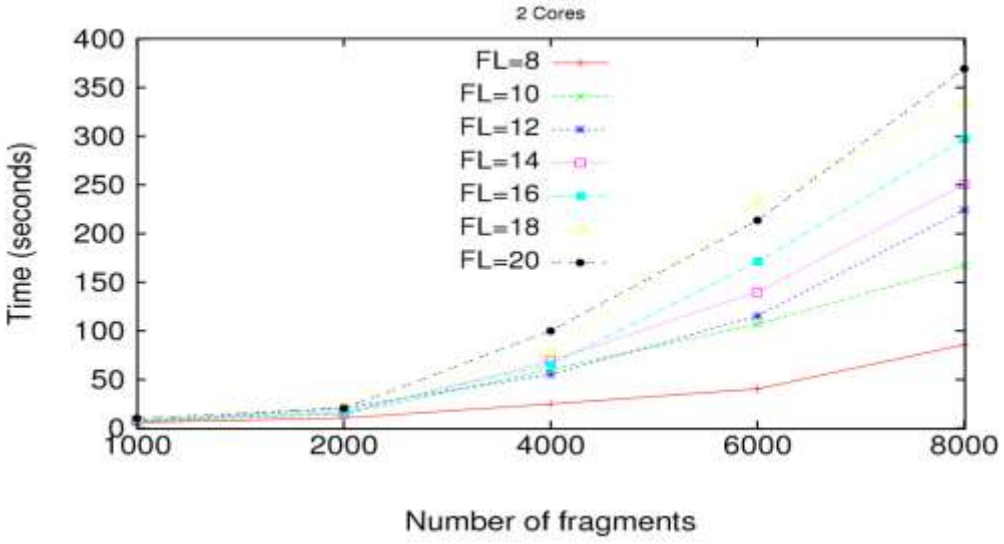



**GRAPH 3: DISK RESULTS TABLE WITH 4 CORES**

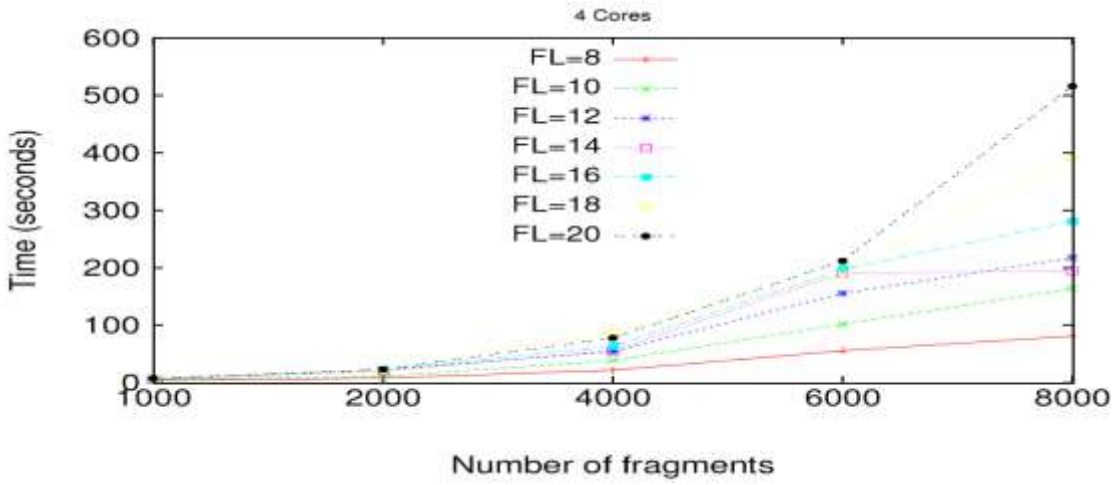



**GRAPH 4: DISK RESULTS TABLE WITH 8 CORES**

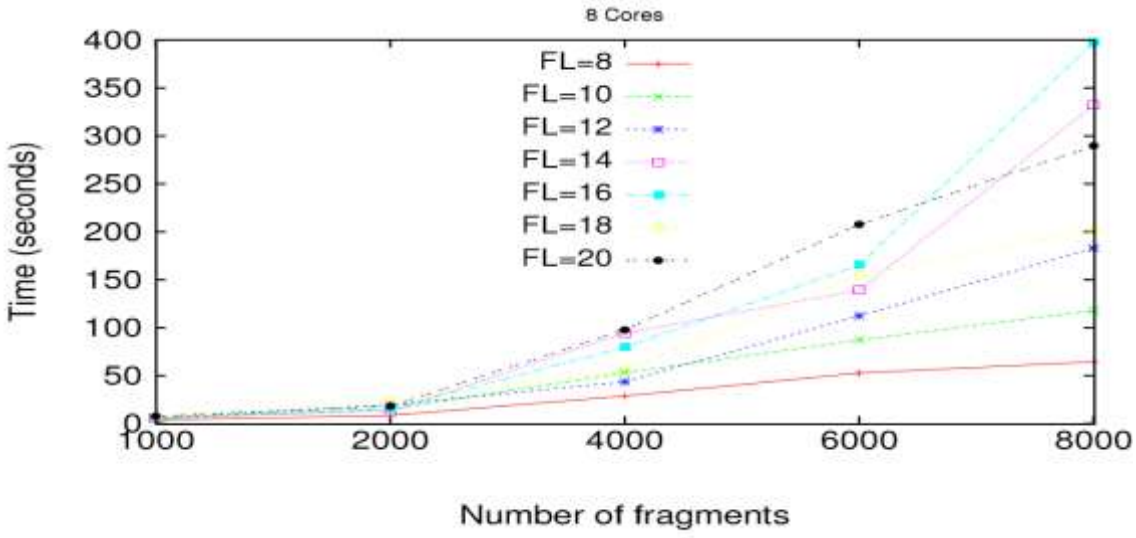
### 4.7 Ram Results with Different Cores and Graph




**GRAPH 5: RAM RESULTS TABLE WITH 2 CORES**

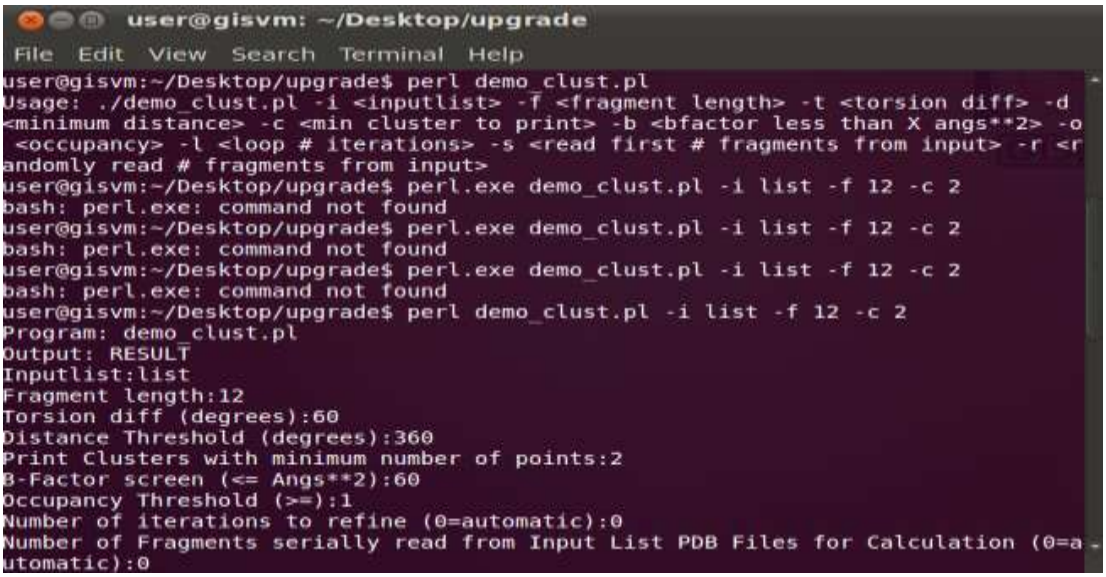


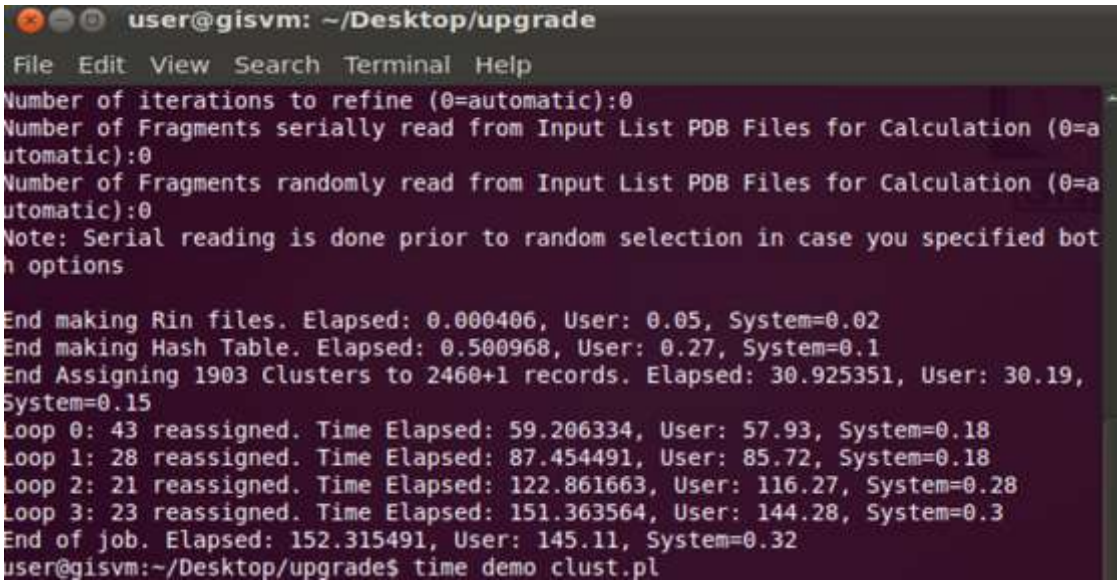

**GRAPH 6: RAM RESULTS TABLE WITH 4 CORES**




**GRAPH 7: RAM RESULTS TABLE WITH 8 CORES**

### 4.8 Snapshots






## V. CONCLUSION

We proposed one approach to parallelizing the demographic clustering of peptide algorithm. The approach is suitable for multi core computations and the computation we have modified based on serial manner of algorithm. The demographic clustering of peptide algorithm work based on reduction. Parallelization technique applied on sequence method for improving the efficiency of the algorithm. In proposed parallelization work we have modified demographic clustering of peptide fragments by parallel programming approach. All the existing methods are implemented in proper way both serial and parallel way. Finally after analyzing the algorithm performance and comparing the result or the algorithm in both parallel and serial approach we can come to a conclusion that our system is computationally efficient in parallel manner. The system should consider more on the CPU and memory usage as they are the main factor that should be reduced during parallel execution of the system by utilizing all the four cores of the processor.

## VI. FUTURE WORK

The future work in this work is to take million or billions of fragments and try to utilize more than four cores. After Parallelism these steps more efficiency will come if each cluster or for X clusters, fork/spawn X processes, In each process compute distance of fragment from fragment center, Unlabeled fragments that are out of cluster based on distance threshold from center.

## REFERENCES

[1] Karuppasmy Manikandan, DebnathPal, Suryanarayanarao Ramkumar, Nathan E Brener, Sitharama S Iyengar and Guna Seetharaman, "Functionally important segments in proteins dissected using gene ontology and geometric clustering ofpeptidefragments", Genome biology, 2008.

[2] http://en.wikipedia.org/wiki/Parallel_programming_model

[3] http://en.wikipedia.org/wiki/Thread_(computing)

[4] http://en.wikipedia.org/wiki/Cluster_analysis

[5] http://en.wikipedia.org/wiki/Multi-core_processor

[6] https://metacpan.org/pod/Parallel::ForkManager
[7] http://search.cpan.org/~dlux/Parallel-ForkManager-0.7.5/ForkManager.pm
[8] http://aaroncrane.co.uk/talks/multicore_for_mortals/paper.html
[9] http://aaroncrane.co.uk/talks/multicore_for_mortals/paper.html
[10] http://www.go4expert.com/articles/parallel-processing-faster-perl-scripts-t9083/
[11] http://perltricks.com/article/61/2014/1/21/Make-your-code-run-faster-with-Perl-s-secret-turbo-module
[12] http://act.yapc.eu/ye2013/talk/4946
[13] https://sandeepnamburi.wordpress.com/2011/07/31/parallel-processing-with-perl-for-bioinformatics/