Applying HT-NGH Algorithm On A Shared Memory Architecture For Time Performance Improvement

muhannad abu-hashem (mabohasm@kau.edu.sa)

Research article

Keywords: computational biology, high-performance computing, pair-wise alignment, parallel design, sequence alignment, shared-memory

Posted Date: May 8th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-27109/v1

APPLYING HT-NGH ALGORITHM ON A SHARED MEMORY ARCHITECTURE FOR TIME PERFORMANCE IMPROVEMENT

MUHANNAD A. ABU-HASHEM

Department of Geomatics, Faculty of Architecture and Planning, King Abdulaziz University, Jeddah, Saudi Arabia; Mabohasm@KAU.EDU.SA

Abstract

Background: Sequence comparison and alignment plays an important role in computational biology as they allow for rating the similarities between molecular sequences. Pair-wise alignment contributes significantly in calculating the similarity between sequences by constructing the optimal alignment. The Hash Table-N-Gram-Hirschberg (HT-NGH) algorithm, which is an extension to the Hashing-N-Gram-Hirschberg (HNGH) and N-Gram-Hirschberg (NGH) algorithms, represents a pair-wise alignment method that uses the capabilities of the hash table technique for the purpose of building the alignment. Due to the current technology, molecular databases have exponentially grown to highlight the needs for faster and efficient methods that can handle these amounts of data. On the other hand, the present technology provides a verity of high performance architectures and tools.

Results: This paper presents a parallel shared memory algorithm for protein pair-wise alignment method, namely, the HT-NGH algorithm in order to enhance the time performance of sequences' comparisons and alignments. The proposed parallel algorithm targets the transformation phase of the HT-NGH algorithm since it consumes about 10% of alignment's executional time. Datasets are decomposed up to sequence level for a more efficient utilization of processing units (no idle processing unit).

Conclusions: As a result, the proposed algorithm demonstrates a significant improvement in terms of time performance without sacrificing accuracy. The speed up pertaining to the parallel algorithm reaches 2.08, 2.88, and 3.87 when using 2, 3, and 4 cores, respectively. Furthermore, this algorithm gains high efficiency as it reaches 1.04, 0.96, and 0.97 when using 2, 3, and 4 cores, respectively.

Keywords: computational biology, high-performance computing, pair-wise alignment, parallel design, sequence alignment, shared-memory.

1. Background

Bioinformatics research is increasingly attracting the interest of many researchers. With the aid of current technologies in molecular sequencing, the size and number of protein, DeoxyriboNucleic Acid (DNA), and RiboNucleic Acid (RNA) databases are growing rapidly by producing enormous biological data. UniProt (Universal Protein Resource), RCSB (Research Collaboratory for Structural Bioinformatics), and EXPASY (Expert Protein Analysis System) [1-3] are examples of different protein database websites that demonstrate the growth of the database size. The rapid growth of molecular databases drives the need for efficient methods to manage and control large amounts of molecular data sizes [4]. This growth of file sizes witnesses serious challenges and motives for researchers to propose faster and efficient sequence comparison methods for controlling, analyzing, and organizing these sizes of daily emerging data.

Sequence Alignment (SA) method is used widely in order to compare sequences as it rates the similarity and distances between various molecular sequences. The calculated similarity may work as an indicator to structural, functional, or evolutionary relationships that the sequences share. Pair-wise alignment method is a SA method, which is used to compute the similarity/distance between two sequences based on calculating the matched blocks of amino acids between the two sequences.

The Pair-wise SA method has a significant role in constructing the distance matrix by building the guide tree, which facilitates the process of constructing the Multiple Sequence Alignment (MSA) algorithm [5]. Such an accurate pair-wise alignment contributes directly in improving and speeding up the process of building the MSA. Constructing the distance matrix requires $(n^2 - n)/2$ pair-wise alignments [6]. In particular, a faster pair-wise alignment could speed up the process of constructing the distance matrix more efficiently, which in turn, speeds up the constructional tree guidance leading to a faster construction of the MSA algorithm.

Developing fast and accurate sequence alignment methods have been the main challenge for many researchers. A large variety of methods have been proposed in order to achieve optimal results without sacrificing the time performance.

The Smith-Waterman [7] method is a pair-wise alignment method, which attempts to reach an optimal alignment where it can, however, sacrifice the execution time it consumes as it takes a time of O(MN) to build the alignment between any two sequences. Furthermore, the N-Gram-Hirschberg (NGH) method [8], which is an extension to the Hirschberg algorithm, aims at reducing time, and meanwhile, producing similar results to the results produced by the Smith-Waterman method. Additionally, the H-NGH [9, 10] and HT-NGH [11] methods are proposed as an enhancement to the NGH method. The H-NGH method enhances the execution time but at the cost of accuracy. On the other hand, the HT-NGH method, which is an enhancement to the NGH and H-NGH methods, presents a further executional time enhancement. This research proposes a method that allows the usage of the current high performance architectures for the purpose of improving the time performance of the HT-NGH algorithm.

Although pair-wise SA methods attain optimal accuracy [7, 8, 12], they are still considered as time consuming methods due to the large volume size of data. The NGH algorithm [8] attempts at decreasing the time including the space that is required to build the alignment. The comparison time between the two protein sequences is $O(mn/k)$, where k is the size term. The comparison results are best defined when k reaches 2 [8]. As molecular databases have rapidly grown, the need for faster algorithms is becoming an urgent necessity.

The outlines of the rest of this paper are organized as follows. Section 2 presents the related research that has been extensively investigated to solve and improve the SA problem. Section 3 explains the problem domain including the newly proposed algorithm. Section 4 demonstrates and discusses the experimental results of the proposed algorithm. Finally, Section 5 gives a conclusion about the proposed algorithm and draws the conclusion and future research pertaining to this study.

2. Related Researches

The Pair-wise SA method is used in order to compare two sequences by aligning them in a way that shows the set of similar blocks of amino acids between the sequences. The unmatched amino acids are shifted by gaps or set to face each other within the sequences. To reduce the number of gaps, a penalty is added to the final score of the alignment.

A massive number of researches along with a large diversity of methods have been conducted to solve and improve the pair-wise alignment challenge. Accordingly, this section shows the main and leading pair-wise alignment methods and discusses their main significant characteristics. Additionally, this section shows and discusses the former methods pertaining to this research. In this paper, different pair-wise alignment methods are classified based on the approach that is used to solve the problem incurred in many different heuristic-based algorithms and DP-based algorithms.

2.1. Heuristic-based Algorithms

The methods that follow the heuristic approaches tend to look for all achievable solutions and pick the best ones based on a number of criteria that is set in advance. Although the selected solution is considered as the best solution in comparison with the solution pool, reaching the optimal solution is a considered a great challenge towards several heuristic approaches. Meanwhile, the methods that apply heuristic approaches are seen to be fast and give results in a reasonable time. This section discusses the main methods that adapt the heuristic approaches in order to solve the pair-wise alignment problem.

FAST-All (FASTA) [26] is a pair-wise SA method that utilizes different heuristic approach capabilities for the purpose of aligning and comparing any involved sequences. The proposed method splits the sequences into a number of blocks, and then searches for any matched blocks. Every block points to a K tuple, which represents the number of matches within any two sequences in order to calculate the resemblance and construct the alignment [13-15]. After that, the BLAST

(Basic Local Alignment Search Tool) method is proposed to enhance the sensitivity in the FASTA method including the time performance [16, 17]. Furthermore, the Sequence Search and Alignment by Hashing Algorithm (SSAHA) is proposed to overcome this problem by applying a hash table technique in order to keep track of the K-tuple occurrence positions [18]. In [19], an improved particle swarm optimization (PSO) method is used in order to solve the pair-wise alignment problem. The gpALIGNER method [20] presents the segment-based DNA pair-wise alignment method that applies a similar score schema used in the DIalign-T [21] algorithm. A space-sufficient local pair-wise alignment [22] is proposed in order to reach an optimal un-gapped alignment.

2.2. Dynamic Programming based Algorithms

Many researches have used the Dynamic Programming (DP) approach in order to build and enhance the performance of the pair-wise alignment methods. In fact, the Needleman-Wunsch algorithm [23] is considered as the first method that uses the DP approach in solving the sequence alignment and comparison problems. In particular, it computes the resemblance between the sequences by constructing a matrix, called the similarity matrix, which is filled based on another matrix called the substitution matrix [24, 25]. Constructing the matrix reserves an O(MN) space, which is a considerable amount if the length of sequences is applied.

As a response to Needleman-Wunsch space issues, the Hirschberg algorithm [12] attempts at reducing the space requirement to O(min (m,n)) such that it decomposes the similarity matrix into two matrices. After that, it starts filling the matrix from top and bottom, independently.

In 1981, the smith-Waterman algorithm is proposed as an improvement to the local pair-wise alignment method. In fact, this algorithm is seen to be similar to the Needleman Wunsch algorithm, but except that it has some similarities between suffixes. To seek for an optimal accuracy, the Smith-Waterman algorithm sacrifices the issues related to time and space [7, 26]. Additionally, an extension to the Smith-Waterman algorithm, namely, the N-Gram-Smith-Waterman algorithm[27], is proposed in order to tackle the time and space issues related to the Smith Waterman algorithm without the need the need to decrease its sensitivity.

The Hirschberg algorithm [12], which applies the DP approach for solving the pair-wise alignment method is proposed in order to improve the space complexity. n [8], further refinements are performed along to the Hirschberg algorithm by proposing the N-Gram-Hirschberg (NGH), which aims to provide further space and time enhancements. In fact, the NGH algorithm enhances the Hirschberg algorithm based on three stages by starting to reduce the alphabet of the protein amino acid to 10 alphabets, and by cutting the sequences into terms based on the use of the N-Gram method, and finally, converting the terms into integer numbers.

Another method called Hashing-N-Gram-Hirschberg (H-NGH) [9, 10] is proposed in order to provide further enhancements to the time performance. The method is an improvement over the NGH algorithm by enhancing the transforming stage (converting the words into integer) of it based on the use of different capabilities related to the hash function. In fact, this function is used to convert the words that are generated by the N-Gram method into integer numbers, which speed up the comparison process and the alignment. Although the H-NGH algorithm enhances the time performance of the NGH algorithm, its accuracy is decreased.

Furthermore, the HT NGH method [6, 11] is proposed as a refinement to the NGH and H-NGH methods with the aim of building the distance matrix, which is used to construct several multiple sequence alignments. The proposed method uses the capabilities of the hash table in order to improve the performance of the transformation stage in the NGH and H-NGH methods. In conclusion, the proposed algorithm outperforms its former methods and demonstrates an improved accuracy and execution time. Additionally, many researches have long been conducted for the purpose of applying the DP approach in order to enhance the efficiency and performance of the pair-wise alignment through a wide diversity [11-12, 30-33]..

Despite the fact that the DP based pair-wise alignment methods have reached to an optimal alignment, reaching a high time performance remains a challenge itself. Many methods such as the NGH, H-NGH, and HT-NGH methods have been produced to reduce the execution time when the

time performance can be further improved. In this research, a parallel computing method is proposed to further enhance the time performance of the HT-NGH algorithm.

3. Problem Definition

In this paper, a multi-core shared memory design is proposed in order to speed up the HT-NGH pair-wise alignment algorithm, which is used to construct the distance matrix. The construction of the distance matrix is accomplished into two stages. The first stage is based on transforming the alphabet of the protein sequences from 20 amino acids represented by 20 different characters into 10 integer numbers. The second stage is based on building the pair-wise alignment and constructing the matrix. The research focuses on speeding the first stage by utilizing the capabilities of the shared memory architecture.

Each protein sequence passes through several reforming levels leading to sequences that are shorter and easier to be compared. This stage consists of many phases starting with the protein sequence alphabet reduction, and passing through further reduction of sequences by dividing them into terms (words) and ending by converting the terms into integer numbers based on the use of a hash table.

The alphabet of the protein sequences consists of a 20 amino acids where each amino acid is represented by a character. The protein alphabet reduction phase is done by applying some sort of clustering to the 20 amino acids based on the similarity of their properties such as physicochemical properties [34, 35]. After the completion of this phase, the 20 amino acids are distributed along into 10 clusters represented by numbers ranging from 0 to 9 where the protein sequences are represented by 10 numbers such that each number refers to a group of amino acids.

Splitting the sequences into terms is performed by applying the N-Gram method (see Table 1). All the terms have the same length while the term 'size' is set earlier. The term 'length' ranges from 2 to 5 amino acids. The final phase in the sequence transformation stage is to convert the term into numbers in which each term is represented by one integer value instead of having N integer numbers (N is the term 'length').

The HT-NGH algorithm uses the hash table technique in order to convert the involved terms from N numbers to an integer value. This process is conducted by using a multi-dimensional array of size 10^N, where N refers to the term 'size', which represents the hash table. The main purpose of the array with such a size is to give enough space to assign a unique integer value for every term. The array contains M dimensions where M is equal to the term 'length'. Each dimension in the array is associated with a number (letter) in the basis which fastens and simplifies the process of addressing the terms within the array. Therefore, each letter combination (word) is assigned with a unique value.

Finally, the protein sequences are aligned by applying a pair-wise alignment based on the use of the Hirschberg algorithm. The main purpose of the alignment is to construct the distance matrix by computing the values of similarities and distances among the involved sequences. To align the sequences, a similarity matrix is constructed and filled with the similarity values between the amino acids. Equation 1 below is used to calculate the similarity values between a number of amino acids in order to fill the similarity matrix appropriately.

$$S(i,j) = Max \begin{cases} S(i-1,j-1) + S(A_i, B_j) \\ S(i-1,j) \\ S(i,j-1) \\ 0 \end{cases} \quad (1)$$

A full pair-wise alignment is conducted, where each sequence is aligned against all other sequences in the dataset in order to build the distance matrix. Algorithm 1 demonstrates how the distance matrix is constructed without any encountered redundancy.

Algorithm1: Building The Distance Matrix

```
for (i=0; i< NumberOfSequences; i++)
    for(j= i+1; j< NumberOfSequences; j++)
    {
       Distance = Calculate_Distance(i,j);
       DistanceMatrix[i][j]= Distance;
    }
```

On the other hand, calculating the distances between the two protein sequences S_1 *and* S_2 is carried out by using Equation 2 indicated below.

$$d\ (S_1, S_2) = 1 - \frac{\text{Exact\_Matching}(S_1, S_2)}{\text{Max}(\text{Length}(S_1),\ \text{Length}(S_2))} \quad (2)$$

4. Analytical Experiments and Findings

To evaluate the proposed algorithm, a Swiss-Prot database is used by also using the FASTA format, which acts to represent the sequences. The proposed algorithm is tested based on the use of various inputs of data. The variation of these inputs relies on the number and length of the involved protein sequences. The results of the proposed algorithm are evaluated in terms of the accuracy and time performance. In fact, the accuracy is evaluated by matching the output results of the proposed algorithm with the obtained results of the original algorithm. Moreover, the time performance is evaluated by computing the speedup, efficiency, and gained performance. Consequently, this section discusses the hardware and software specifications, results measurements, datasets, time performance and accuracy in detail.

4.1. Hardware and Software Specifications

The experiments are run on an Intel® Core™ i7-8550U multi-core processor with an 8 GB RAM. The processor consists of 4 cores and the type of the system is 64 bits. The experiments are conducted on windows 10 Pro by using the C++ compiler.

4.2. Time Performance Measurements

The time performance is evaluated by computing the speedup, efficiency, and gained performance. Accordingly, this section discusses how the results are measured.

4.2.1. Speedup: refers to the performance of the parallel algorithm by relating the parallel algorithm results to the results pertaining to the sequential algorithm. Equation 1 shows how the speedup is calculated.

$$\boldsymbol{Speedup} = \frac{\boldsymbol{T_1}}{\boldsymbol{T_P}} \quad (\mathbf{3})$$

Where,

T_1 denotes the execution time for the fastest sequential program when using a single core.

T_P denotes the execution time of the parallel algorithm when using P cores.

4.2.2. Efficiency: it indicates whether or not the processing units are effectively employed by splitting the speedup on the number of processing units. Equation 2 highlights the calculation process of the efficiency employment.

$$\text{Efficiency} = \text{Speedup} / \text{P} \quad (4)$$

Where, P is the number of processing units.

4.2.3. Performance Gain: it refers to the percentage of the gained performance during the use of the parallel algorithm. Equation 3 shows how the gained performance is calculated.

$$PerformanceGain = \frac{T_{Alg1} - T_{Alg2}}{T_{Alg1}} \times 100\% \quad (5)$$

Where, T_{Alg1} denotes the execution time of the sequential algorithm, and T_{Alg2} denotes the execution time of the parallel algorithm.

4.3. Datasets

The Swiss-Prot database is used as a dataset for testing the proposed algorithm, while the protein sequences are represented in the FASTA format. Figure 1 illustrates an example of the protein sequence in the FASTA format.

The proposed algorithm is tested in diverse sizes of dataset parameters. The parameters represent the number of sequences, the database size in megabyte and the term length. Table 2 shows the ranges of such parameters.

4.4. Results

253 It is found to be proven from the proposed algorithm that the results demonstrate an improved
254 time performance without sacrificing the accuracy as compared with the original algorithm (i.e. the
255 HT-NGH algorithm). This section discusses the results in terms of the time performance and
256 accuracy.

257 4.4.1. The Obtained Results in Terms of Time

258 The experiments demonstrate that the proposed method outperforms different HT-NGH methods.
259 The time performance of the proposed algorithm is evaluated by computing the speedup, efficiency
260 and gained performance of the obtained results.

261 *Speedup*: Experiments show that the parallel algorithm obtains higher speedup. This enhancement of
262 speed performance is by means to indicate to the result of reducing the communication overhead
263 among different cores by providing the cores with the same task to perform. Additionally, load
264 balancing plays an important role in increasing the speed of the parallel algorithm since it allows this
265 algorithm to reach the most effective use of resources (no idle cores). Figures 2 – 4 show the values of
266 speedup that are taken for the sizes of the dataset that range from 97MB to 368MB, the term 'length'
267 ranging from 2 to 5 increments by one, and the number of cores ranging from 2 to 4 increments by
268 one. The results demonstrate that the number of cores has significant effects on the speedup that
269 increases when the number of cores increases at the same time. The main reason behind this refers
270 back to the increment in the number of resources when the number of processing units is increased.
271 Further, the length of the term 'word' has a considerable impact on the speedup. The highest
272 speedup is gained when the term 'size' is equal to 2. On the other hand, the size of datasets does not
273 have any considerable impacts over the speedup.

274 The highest speedup is obtained when 4 cores is mainly being used for the term 'length' that is
275 equal to 2.

Efficiency: The experiments show that the proposed algorithm obtains a high efficiency that is mainly gained due to the high speedup incurred into the algorithm. As shown in Equation 2, the speedup has a significant impact on the efficiency of the algorithm.

Unlike the speedup results, the highest efficiency is obtained when the number of cores is equal to two while the lowest is obtained at 4 cores. The highest efficiency is 1.044 that is obtained based on the term 'length' that is equal to 3 (3 Gram).

Figures 5 – 7 illustrate the values of efficiency that are taken for the sizes of the dataset and that are ranging from 97MB to 368MB, the term 'length' ranging from 2 to 5 increments by one, and the number of cores ranging from 2 to 4 increments by one.

It is shown to be proven from the obtained results that the value of speedup has a great impact on the efficiency score. Additionally, the number of cores has a contrast impact on the efficiency value in which the less cores use the highest gained efficiency values. In fact, this is due to the increment in communications such as the distribution of the data through to the cores and the gathering of the obtained results from the cores.

Performance Gain: The experiments show that the proposed algorithm obtains higher gains of performance. The obtained performance gain is mainly acquired because of the high speedup of the algorithm. As noticed in Equation 3, the high speedup leads to higher gains of performance. Figures 8 – 10 highlight the values of the gained performance that are taken for the sizes of the dataset ranging from 97MB to 368MB, the term 'length' ranging from 2 to 5 increments by one, and the number of cores ranging from 2 to 4 increments by one. As can be seen from these figures, the highest performance gain is 74.19 that is obtained when 4 cores are used along with the term 'length' that is equal to 2. On the other hand, the performance is decreased when the number of cores is also decreased; besides, the lowest performance gain is scored when 2 cores are used. Using more processing units for executing the algorithm leads to a more effective performance gain.

4.4.2. Results in Terms of the Accuracy

The accuracy of the proposed algorithm is evaluated in terms of matching the results of the proposed algorithm along with the results of the HT-NGH algorithm. The proposed algorithm produces the same results as the ones in the HT-NGH algorithm since the results of the two algorithms are seen to be fully matched. This amount of accuracy is achieved due to the decomposing technique that is used since the proposed method uses the dynamic load balancing method for the data decomposition technique such that the data is decomposed at a sequence level.

5. Conclusion

This paper presents a parallel algorithm for a pair-wise sequence alignment method called the HT-NGH method based on the use of the shared memory parallel architecture. This method is an enhancement to the H-NGH and NGH methods in which the time performance has increased without reducing its accuracy.

The SIMD architecture is used to distribute the data among different processing units where the data is decomposed into a level of sequences. The load balancing method is applied in order to avoid any idle processing units during the execution. Experiments demonstrate the superiority of the parallel algorithm based on the time performance over the HT-NGH algorithm without the need to leave the accuracy apart. The speedup reaches its highest results by using 4 cores while the highest efficiency is gained by using 2 cores.

This research focuses on the multi-core architecture as a parallel architecture whereas other parallel architectures such as multi-processors and GPU could be employed and used. Furthermore, a suitable combination/ hybridization between several different parallel architectures could have a high impact on the obtained results.

Abbreviations

BLAST	Basic Local Alignment Search Tool
DNA	DeoxyriboNucleic Acid
DP	Dynamic Programming

EXPASY	Expert Protein Analysis System
FASTA	FAST-All
HNGH	Hashing-N-Gram-Hirschberg
HT-NGH	HashTable-N-Gram-Hirschberg
MSA	Multiple Sequence Alignment
NGH	N-Gram-Hirschberg
PSO	particle swarm optimization
RCSB	Research Collaboratory for Structural Bioinformatics
RNA	RiboNucleic Acid
SA	Sequence Alignment
SSAHA	Sequence Search and Alignment by Hashing Algorithm
UniProt	Universal Protein Resource

6. Methodology

This paper proposes an algorithm that exploits the performance of the parallel architecture in order to speed up the HT-NGH algorithm. The Single Instruction Multiple Data (SIMD) architecture is used to manage the parallel process where the data is decomposed in such a way each core is assigned with a protein sequence at a time when processing units are used more efficiently. Furthermore, this architecture splits the data into different sub-datasets. After that, it manages these datasets by using the same piece of code.

6.1. The Parallel Algorithm

In order to parallelize the HT-NGH algorithm, the problem is decomposed into a data level where each core is assigned with a protein sequence at a given time. The OpenMP and C++ are both programming languages that are used to implement the parallel algorithm. Figure 11 shows the data decomposition architecture of the parallel algorithm where the sequences transformation algorithm is applied by all cores through different datasets.

353 6.1.1. Architecture

354 The parallel architecture is based on the multi-cores architecture where the quad-core processor
355 is used accordingly. Figure 12 highlights a schematic diagram of the parallel architecture. On the
356 other hand, the SIMD architecture is used to manage different parallel processes. This architecture is
357 applied to manage the decomposition process since the aim is to distribute the dataset over the cores,
358 which are executing the same code (task).

359 6.1.2. The Decomposition method

360 In this research, data decomposition is used where the dataset is decomposed into a number of
361 blocks and sent along to the cores. The number and size of the blocks are determined based on the
362 dynamic load balancing technique.

363 The dynamic load balancing technique is used to balance the loads among the cores where the
364 number of blocks is increased at the time the size of the blocks is decreased. Each core receives a
365 single protein sequence at a time. In fact, employing such a technique would increase the
366 communication overhead among the cores due to the increasing number of data blocks.
367 Additionally, faster cores get more sequences to process than slower cores, which lead to provide
368 more effective use of processing units when there are no idle cores to be processed within a
369 particular time. Moreover, the size of protein sequences varies from around 100 amino acids to
370 thousands of amino acids. When using a fixed size of blocks, the entire cores obtain the same
371 number of protein sequences that would slowdown some cores. The reason behind this is that
372 protein sequences do not share the same length, and implies that some cores should handle longer
373 sequences compared to other cores. Figure 13 illustrates the database partitioning process.

374 **Acknowledgments**

375 This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz
376 University, Jeddah, under grant No. D-139-137-1441. The authors, therefore, gratefully acknowledge
377 DSR technical and financial support. Also I would like to acknowledge that parts of this research

were discussed earlier in [37] such as the enhanced method (transformation phase of HTNGH), parallel architecture, and references. So it might be some similarity between the two works.

Declarations

Ethics approval and Consent to participate (Not Applicable)

Consent for publication (Not Applicable)

Availability of data and material (The data is available upon request)

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Conflicts of interest/Competing interests

The authors declare that they have no competing interests

Funding

This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under grant No. D-139-137-1441. The authors, therefore, gratefully acknowledge DSR technical and financial support.

Authors' contributions (Not Applicable)

Acknowledgements (Not Applicable)

References

1. Expasy. 2019. Database of protein domains, families and functional sites [Online]. Available: https://prosite.expasy.org [Accessed 10 Sep 2019 2019].
2. RCSB. 2019. RCSB Protein Data Bank [Online]. Rutgers and UCSD. Available: https://www.rcsb.org [Accessed 10 Sep 2019 2019].
3. UNIPROT. 2019. Universal Protein Resource [Online]. Available: https://www.uniprot.org [Accessed 10 Sep 2019 2019].
4. Abu-Hashem, M. A., Rashid, N. A. A., Abdullah, R., et al. 2010b. 3D Protein structure comparison and retrieval methods: investigation study. International Journal of Computer Science and Information Security (IJCSIS), 8(8). 223-227.
5. Abu-Hashem, M. A., Nur'Aini, A. R., Abdullah, R., Hasan, A. A., et al. 2015. Investigation study: an intensive analysis for MSA leading methods. Journal of Theoretical & Applied Information Technology, 75(1). 1-12.
6. Abu-Hashem, M. A., Nur'Aini, A. R., Abdullah, R., et al. 2016. Filtered distance matrix for constructing high-throughput multiple sequence alignment on protein data. Journal of Theoretical and Applied Information Technology, 86(3), 451.
7. Smith, T. F. & Waterman, M. S. 1981. Identification of common molecular subsequences. Journal of Molecular Biology, 147, 195-197.

8. Nur'Aini, A. R. 2008. Enhancement of hirschberg algorithm using n-gram and parallel methods for fast protein homologous search. PhD Universiti Sains Malaysia.
9. Abu-Hashem, M. A., & Nur'Aini, A. R. 2009. Enhancing N-Gram-Hirschberg algorithm by using hash function. Paper presented at the 2009 Third Asia International Conference on Modelling & Simulation. pp. 282-286.
10. Abu-Hashem, M. A., Nur'Aini, A. R., Abdullah, R., et al. 2010a. Parallel Hashing-N-Gram-Hirschberg algorithm. Paper presented at the 2010 2nd International Conference on Computer Technology and Development. pp. 37-41
11. Abu-Hashem, M. A., Nur'Aini, A. R., Abdullah, R., et al. 2012. The use of hash table for building the distance matrix in a pair-wise sequence alignment. Paper presented at the International Conference on Computer Technology and Development ICCTD. 283-294
12. Hirschberg, D. S. 1975. A linear space algorithm for computing maximal common subsequences. Commun. ACM, 18, 341-343.
13. Wilbur, W. J. & Lipman, D. J. 1983. Rapid similarity searches in nucleic acid and protein databanks. Proc. Natl. Acad. Sci. USA, Vol(80), 726-730.
14. Lipman, D. J. & Pearson, W. R. 1985. Rapid and sensitive protein similarity searches. Sceince 227, 1435-1441.
15. Pearson, W. R. & Lipman, D. J. 1988. Improved tools for biological sequence comparisons. Proc.Nat. Acad. Sci.USA. 85, 2444-2448.
16. Altschul, S. F., Gish, W., Miller, W., et al. 1990. Basic local alignment search tool. J.Mol.Biol 215 , 403-410.
17. Altschul, S. F., Madden, T. L., Schaffer, A. A., et al. 1997. Gapped plast and psi-blast : a new generation of protein database search programs. Nucleic Acids Research , Vol.25 (17), 3389-3402.
18. Ning, Z., Cox, A. & Mullikin, J. 2001. SSAHA: A fast search method for large dna databases. Genome Research, 11, 1725-1729.
19. Wenjie, Y., Shanshan, W. & Guoli, J. 2009. The Application of an Improved Particle Swarm Optimization (PSO) Algorithm in Pairwise Sequence Alignment. Presented at International Conference on Computational Intelligence and Security, 2009. CIS '09. pp. 125-128.
20. Hadian Dehkordi, M., Masoudi-Nejad, A. & Mohamad-Mouri, M. 2011. gpALIGNER: A fast algorithm for global pairwise alignment of dna sequences. Iran. J. Chem. Chem. Eng. Vol, 30. 139-146.
21. Subramanian, A., Weyer-Menkhoff, J., Kaufmann, M. et al. 2005. DIALIGN-T: an improved algorithm for segment-based multiple sequence alignment. BMC Bioinformatics, 6, 66.
22. Stojanov, D., Mileva, A. & Koceski, S. 2012. A new, space-efficient local pairwise alignment methodology. Advanced Studies in Biology, 4, 85.
23. Needleman, S. B. & Wunsch, C. D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48, 443-453.
24. Dayhoff, M. 1965. Atlas of protein sequence and structure, Nat Biomed Research Foundation.
25. Henikoff, S. & Henikoff, J. G. 1992. Amino acid substitution matrices from protein blocks. Proceedings of the National Academy of Sciences of the United States of America, 89, 10915-10919.
26. Gotoh, O. 1982. An improved algorithm for matching biological sequences. Journal of Molecular Biology, 162, 705-708.
27. Rashid, N. A., Abdullah, R., Abdullah & Ali, Z. 2006. Fast dynamic programming based sequence alignment algorithm. Presented at The 2nd International Conference on Distributed Frameworks for Multimedia Applications, 2006. pp. 1-7

28. Hara, T., Sato, K. & Ohya, M. 2010. MTRAP: Pairwise sequence alignment algorithm by a new measure based on transition probability between two consecutive pairs of residues. BMC bioinformatics, 11, 235.
29. Zivanic, M., Daescu, O. & Kurdia, A. 2011. Rigid region pairwise sequence alignment. Presented at International Conference on Bioinformatics and Biomedicine Workshops (BIBMW), 2011 IEEE. pp. 319-326 .
30. Choi, Y. 2012. A fast computation of pairwise sequence alignment scores between a protein and a set of single-locus variants of another protein. Presented at Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine. ACM. 414–417.
31. Deronne, k. W. & Karypis, G. 2013. Pareto optimal pairwise sequence alignment. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 10, 481-493.
32. Jayapriya, J. & Arock, M. 2016. Aligning two molecular sequences using genetic operators in grey wolf optimiser technique. International Journal of Data Mining and Bioinformatics, 15, 328-349.
33. Kaur, H. & Chand, L. 2016. Pairwise sequence alignment of biological database using soft computing approach. Reliability, Presented at 2016 5th International Conference on Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), IEEE, 72-77.
34. Mathew, C. K. & Van Holde, Z. E. 1995. Biochemistry, Benjamin Cumming, San Francisco, CA.
35. Sinha, N. & Nussinov, R. 2001. Point mutations and sequence variability in proteins: Redistributions of preexisting populations. Proceedings of the National Academy of Sciences of the United States of America, 98, 3139-3144.
36. Asaduzzaman, A. 2013. A power-aware multi-level cache organization effective for multi-core embedded systems. JCP, 8, 49-60.
37. Abu-Hashem, M. A., M.A., Uliyan, D.M. and Abuarqoub, A., 2017. A Shared Memory Method For Enhancing The HTNGH AlgorithmPerformance: Proposed Method. In Proceedings of the International Conference on Future Networks and Distributed Systems (pp. 1-6).

481 **Additional Files List**

File Name	File Format	Title of Data	Description
Additional File 1	FASTA	Dataset	Dataset in a Fasta file format. It contains the protein sequences in fasta format. The file consists of 222447 protein sequences. File size is around 97MB.
Additional File 2	Text file (TXT)	Dataset	Dataset in a TXT file format. It contains the protein sequences in fasta format. The file consists of 524501 protein sequences. File size is around 238MB.
Additional File 3	Text file (TXT)	Dataset	Dataset in a TXT file format. It contains the protein sequences in fasta format. The file consists of 753599 protein sequences. File size is around 339MB.
Additional File 4	C++ code file (CPP)	C++ Code	C++ code file that contains the sequential code of the process of reducing and dividing the amino acids into terms of 2 Grams. To run the code file use CodeBlocks software.
Additional File 5	C++ code file (CPP)	C++ Code	C++ code file that contains the sequential code of the process of reducing and dividing the amino acids into terms of 3 Grams. To run the code file use CodeBlocks software.
Additional File 6	C++ code file (CPP)	C++ Code	C++ code file that contains the sequential code of the process of reducing and dividing the amino acids into terms of 4 Grams. To run the code file use CodeBlocks software.
Additional File 7	C++ code file (CPP)	C++ Code	C++ code file that contains the sequential code of the process of reducing and dividing the amino acids into terms of 5 Grams. To run the code file use CodeBlocks software.
Additional File 8	C++ code file (CPP)	C++ Code	C++ code file that contains the Parallel code of the process of reducing and dividing the amino acids into terms of 2 Grams. To run the code file use CodeBlocks software. Also you need to install MinGW library to be able to use

			OpenMP library in CodeBlocks.
Additional File 9	C++ code file (CPP)	C++ Code	C++ code file that contains the Parallel code of the process of reducing and dividing the amino acids into terms of 3 Grams. To run the code file use CodeBlocks software. Also you need to install MinGW library to be able to use OpenMP library in CodeBlocks.
Additional File 10	C++ code file (CPP)	C++ Code	C++ code file that contains the Parallel code of the process of reducing and dividing the amino acids into terms of 4 Grams. To run the code file use CodeBlocks software. Also you need to install MinGW library to be able to use OpenMP library in CodeBlocks.
Additional File 11	C++ code file (CPP)	C++ Code	C++ code file that contains the Parallel code of the process of reducing and dividing the amino acids into terms of 5 Grams. To run the code file use CodeBlocks software. Also you need to install MinGW library to be able to use OpenMP library in CodeBlocks.
Note: To run the parallel code you need to install MinGW library to be able to use OpenMP library in CodeBlocks. Also to use the parallel code for certain amount of cores (2,3, or 4), you need to set the number of threads to the desired number by using the code (omp_set_num_threads(N);) then put the number of cores instead of N. To change the dataset file you need to set the path the desired dataset file using (InputFile.open("Dataset File path\\Additional File 1.fasta");).			

Figures legends

Figure 1. Protein sequences in FASTA format

Figure 2: The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 97MB (200k Protein Sequences)

Figure 3: The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 234MB (500k Protein Sequences)

Figure 4: The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 339MB (753k Protein Sequences)

492 **Figure 5** Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 97mb (200k protein
493 sequences)
494 **Figure 6** Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 234mb (500k protein
495 sequences)
496 **Figure 7** Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 339mb (753k protein
497 sequences)
498 **Figure 8** Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 97MB (200k Protein Sequences)
499 **Figure 9** Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 234MB (500k Protein
500 Sequences)
501 **Figure 10** Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 339MB (753k Protein Sequences)
502 **Figure 11.** Data decomposition architecture
503 **Figure 12.** Schematic diagram of Intel quad-core architecture [36]
504 **Figure 13.** Dataset decomposition
505
506

Table 1. Examples of cutting a sequence S down into terms using N-Gram

N	Example N-Gram terms For S= 812735964523
2	81, 27, 35, 96, 45, 23
3	812, 735, 964, 523
4	8127, 3596, 4523
5	81273, 59645, 23

507

Table 2. Experimental parameters and Ranges

Parameter	Range		Increment
	From	To	
Database size (in MB)	97	368	-
Term length	2	5	1
Number of sequences	200,000	800,000	-

508

>11S3_HELAN (P19084) 11S globulin seed storage protein G3 precursor (Helianthinin G3) [Contains: 11S globulin seed storage protein G3 acidic chain; 11S globulin seed storage protein G3 basic chain]

MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEALEPIEVIQAEAGVTEIWDAYD

QQFQCAWSILFDTGFNLVAFSCLPTSTPLFWPSSREGVILPGCRRTYEYSQEQQFSGEGG

RRGGGEGTFRTVIRKLENLKEGDVVAIPTGTAHWLHNDGNTELVVVFLDTQNHENQLDENQRRFFLAGNP

QAQAQSQQQQQRQPRQQSPQRQRQRQRQGQGQNAGNIFNGFTPELIAQSFNVDQETAQKLQGQNDQRGHI

VNVGQDLQIVRPPQDRRSPRQQQEQATSPRQQQEQQQGRR.....

>11SB_CUCMA (P13744) 11S globulin beta subunit precursor [Contains: 11S globulin gamma chain (11S globulin acidic chain); 11S globulin delta chain (11S globuln basic chain)]

MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVWQQHRYQSPRACRLENLRAQDPVR

Figure 1. Protein sequences in FASTA format

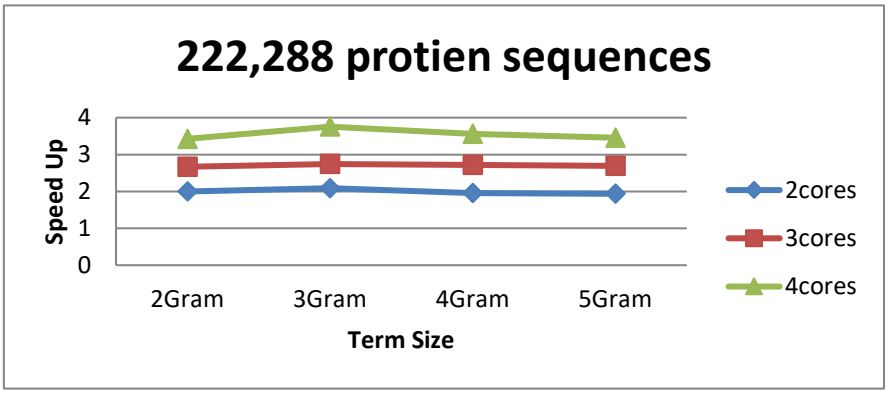

Figure 2: The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 97MB (200k Protein Sequences)

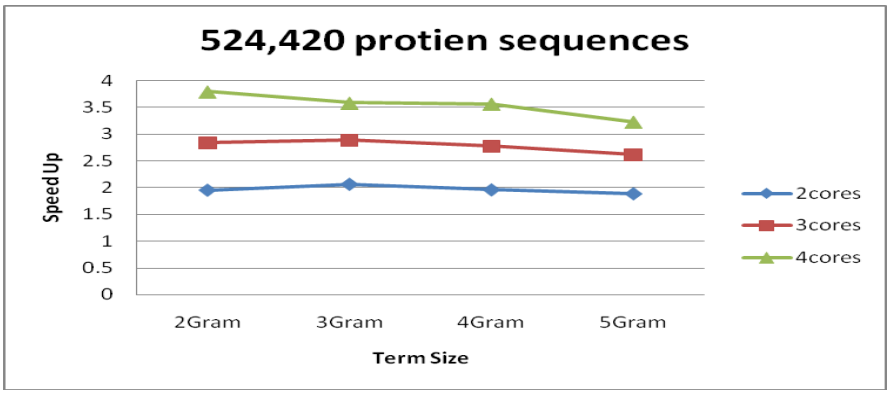

Figure 3: The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 234MB (500k Protein Sequences)

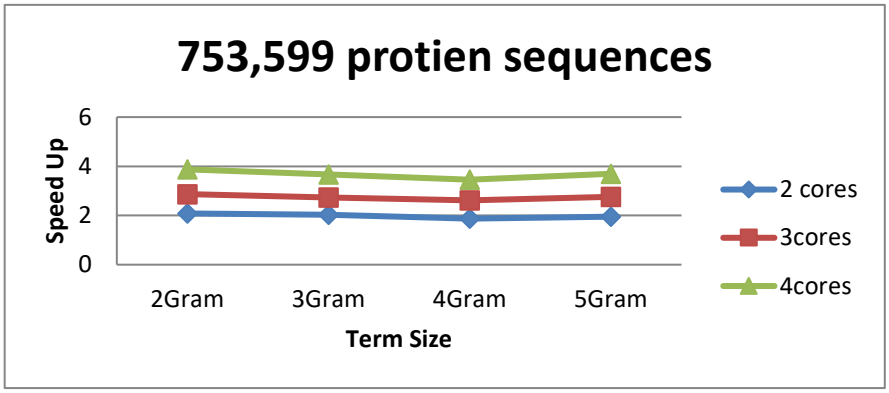

Figure 4: The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 339MB (753k Protein Sequences)

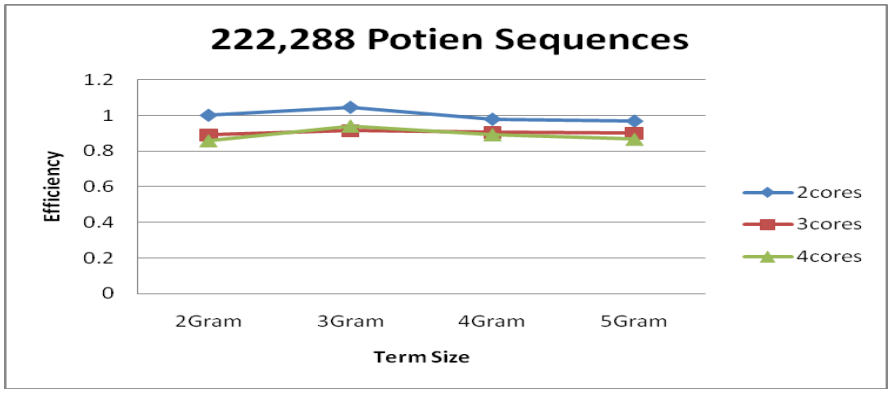

Figure 5 Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 97mb (200k protein sequences)

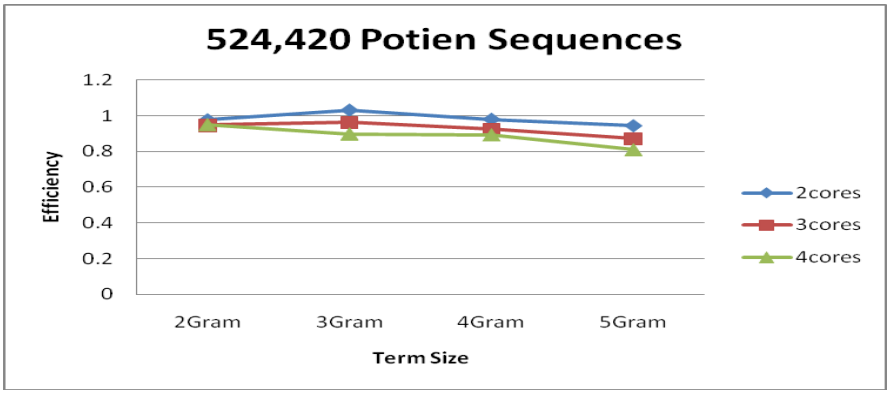

Figure 6 Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 234mb (500k protein sequences)

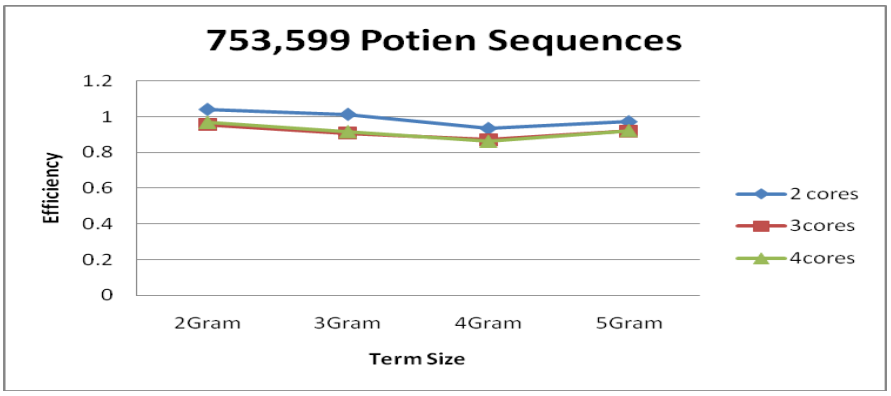

Figure 7 Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 339mb (753k protein sequences)

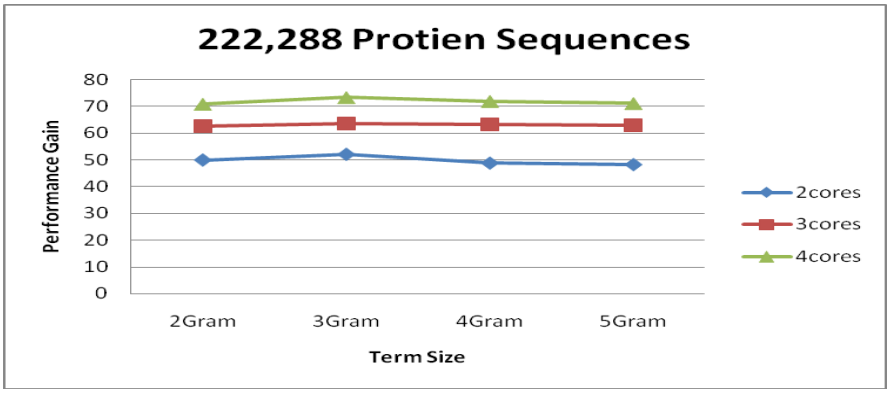

Figure 8 Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 97MB (200k Protein Sequences)

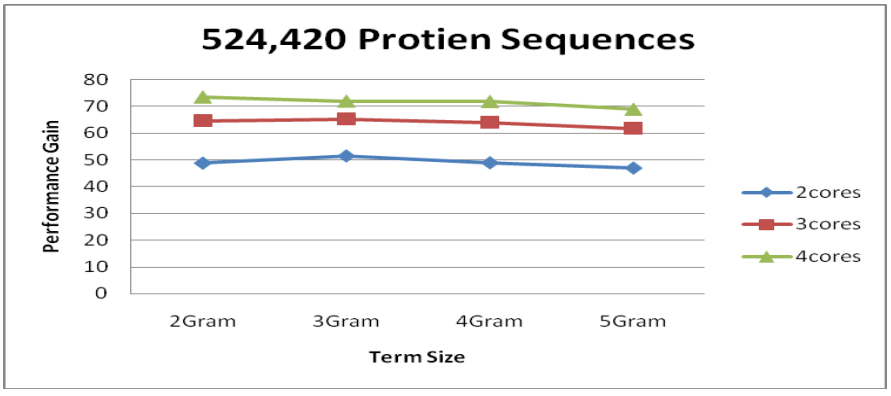

Figure 9 Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 234MB (500k Protein Sequences)

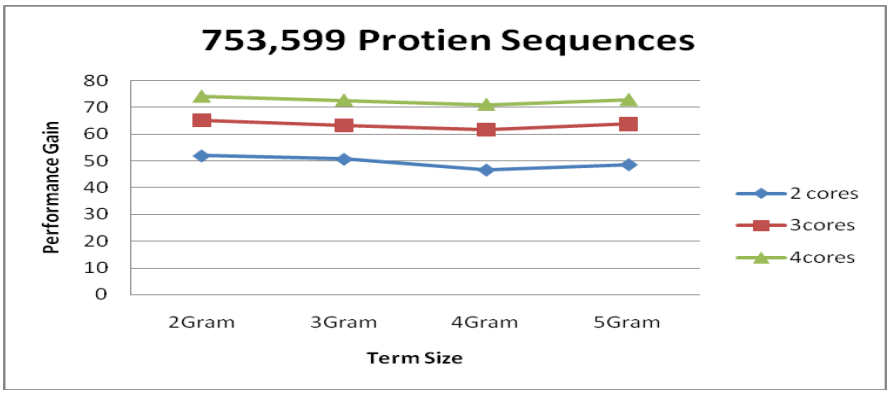

Figure 10 Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 339MB (753k Protein Sequences)

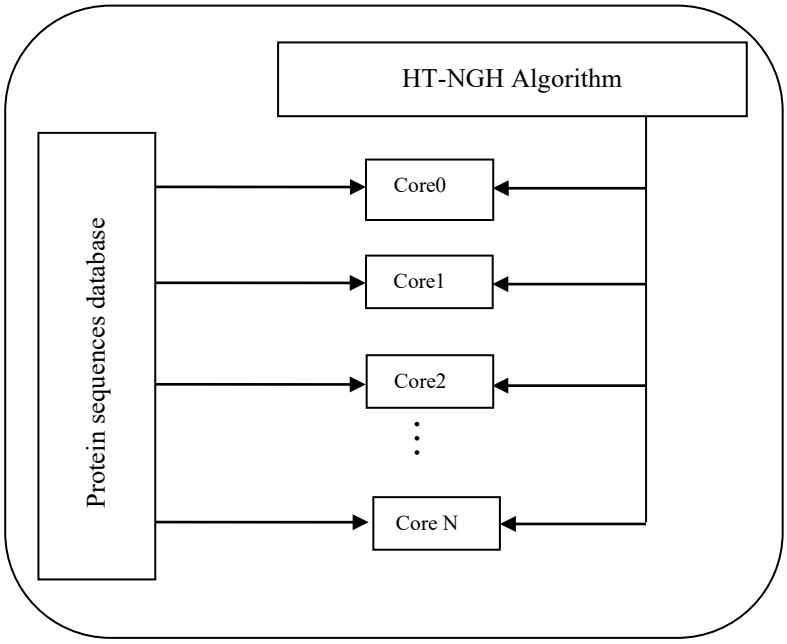

Figure 11. Data decomposition architecture

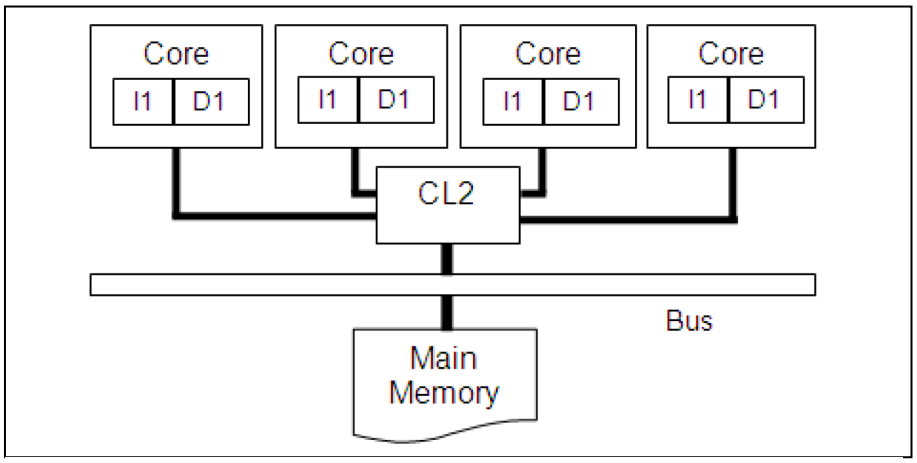

Figure 12. Schematic diagram of Intel quad-core architecture [36]

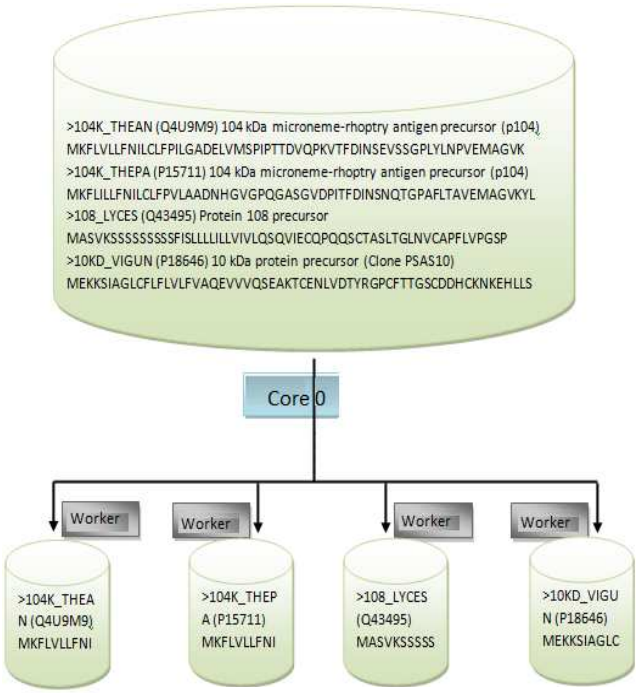

Figure 13. Dataset

Figures

>11S3_HELAN (P19084) 11S globulin seed storage protein G3 precursor (Helianthinin G3) [Contains: 11S globulin seed storage protein G3 acidic chain; 11S globulin seed storage protein G3 basic chain]

MASKATLLLAFTLLFATCIARHQQRQQQQNQCQLQNIEALEPIEVIQAEAGVTEIWDAYD

QQFQCAWSILFDTGFNLVAFSCLPTSTPLFWPSSREGVILPGCRRTYEYSQEQQFSGEGG

RRGGGEGTFRTVIRKLENLKEGDVVAIPTGTAHWLHNDGNTELVVVFLDTQNHENQLDENQRRFFLAGNP

QAQAQSQQQQQRQPRQQSPQRQRQRQRQGQGQNAGNIFNGFTPELIAQSFNVDQETAQKLQGQNDQRGHI

VNVGQDLQIVRPPQDRRSPRQQQEQATSPRQQQEQQQGRR.....

>11SB_CUCMA (P13744) 11S globulin beta subunit precursor [Contains: 11S globulin gamma chain (11S globulin acidic chain); 11S globulin delta chain (11S globuln basic chain)]

MARSSLFTFLCLAVFINGCLSQIEQQSPWEFQGSEVWQQHRYQSPRACRLENLRAQDPVR

Figure 1

Protein sequences in FASTA format

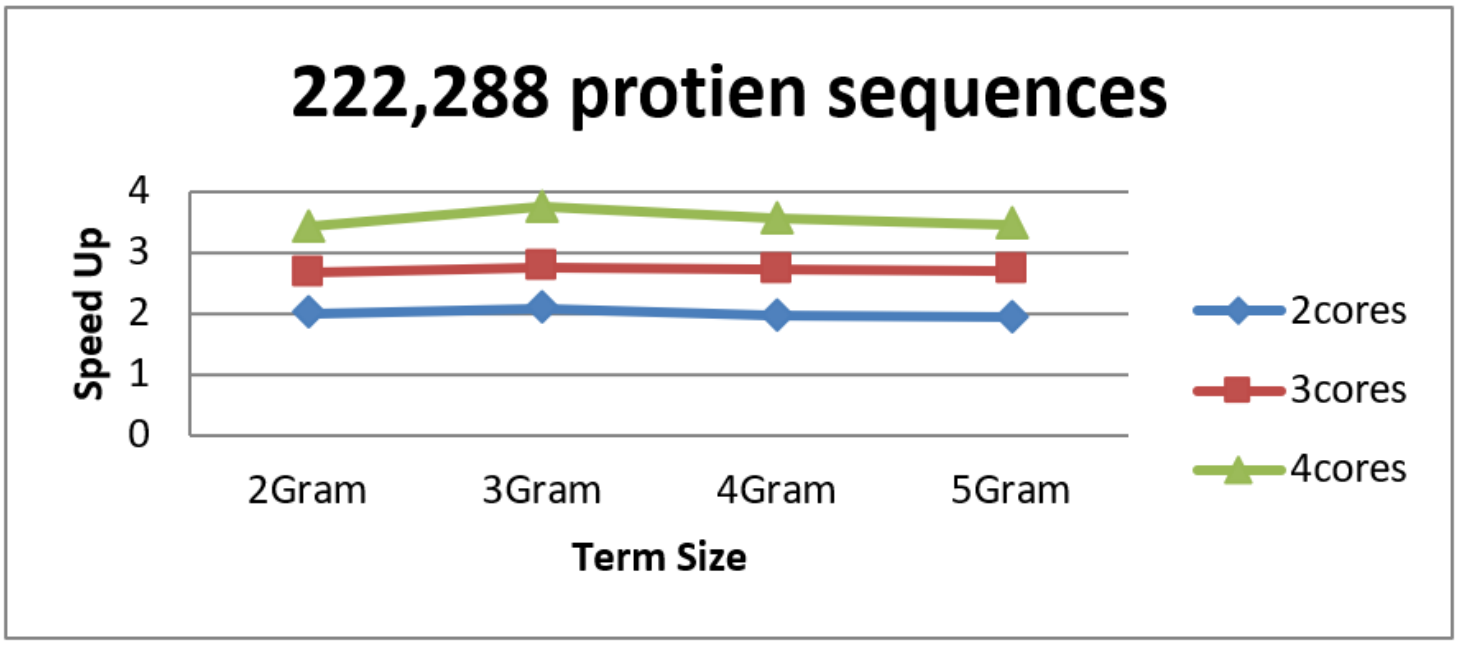

Figure 2

The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 97MB (200k Protein Sequences)

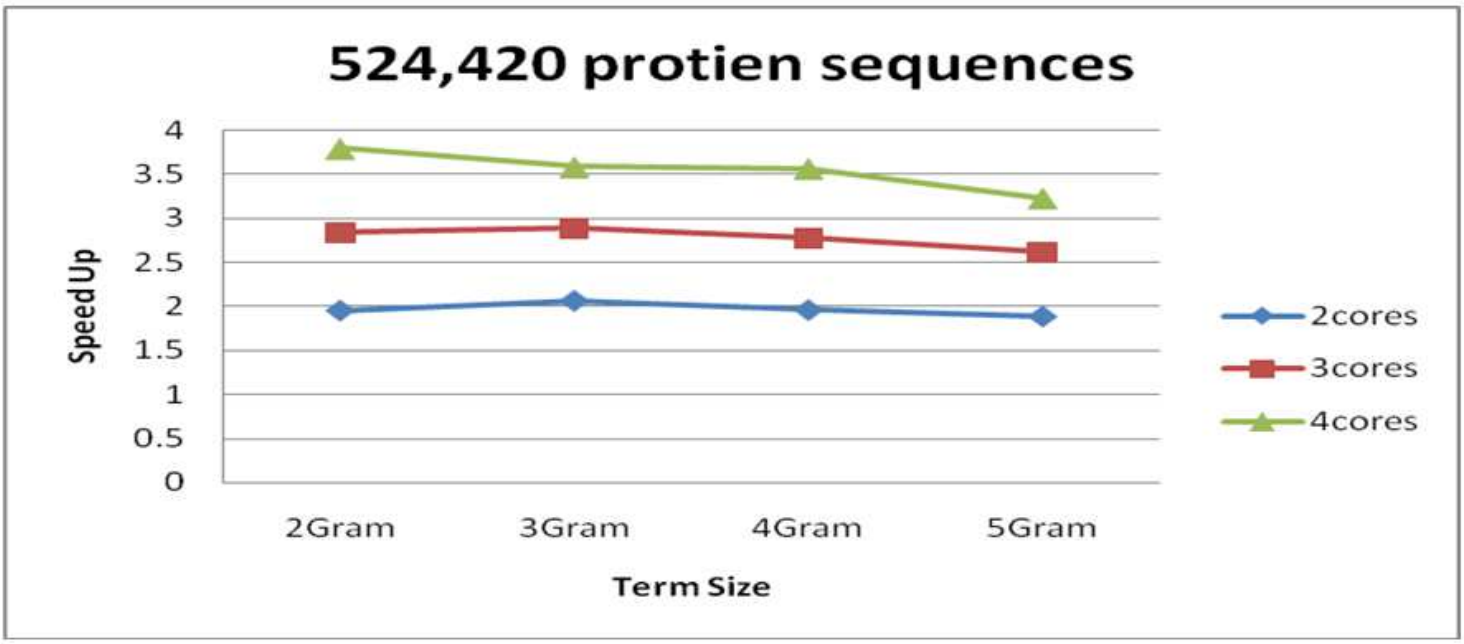

Figure 3

The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 234MB (500k Protein Sequences)

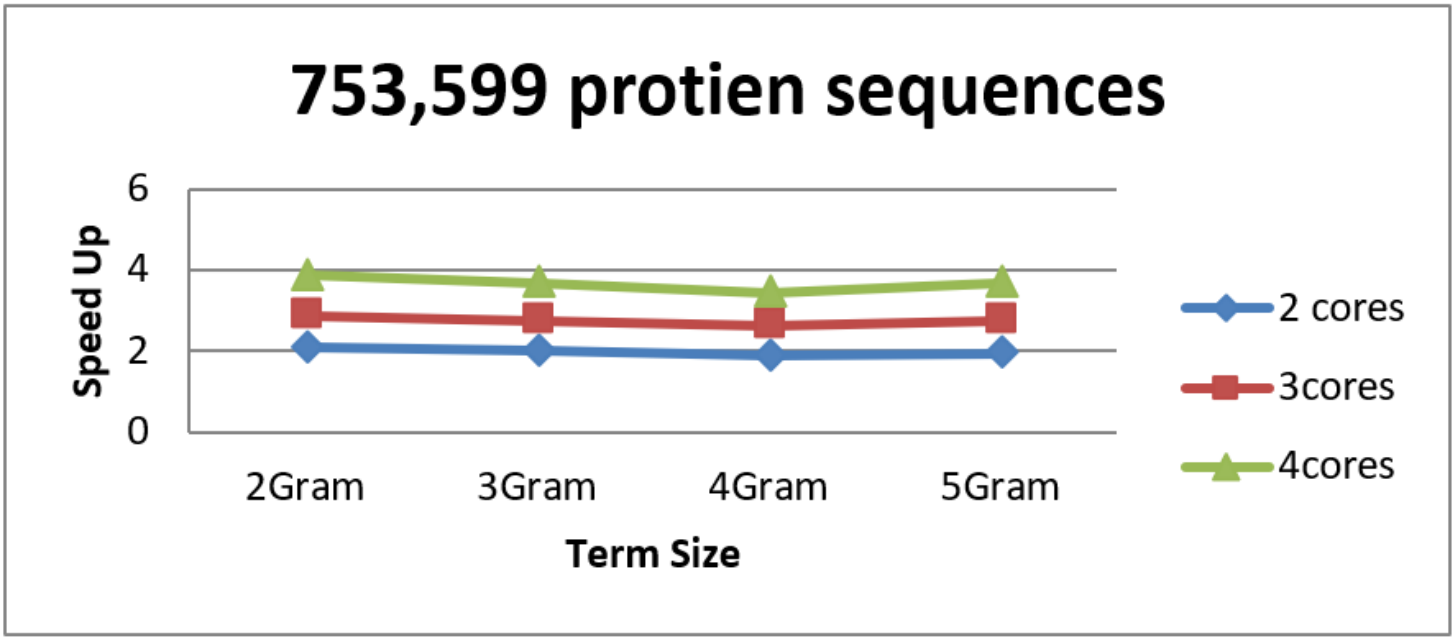

Figure 4

The speedup of the Parallel HT-NGH algorithm in terms of the dataset size is equal to 339MB (753k Protein Sequences)

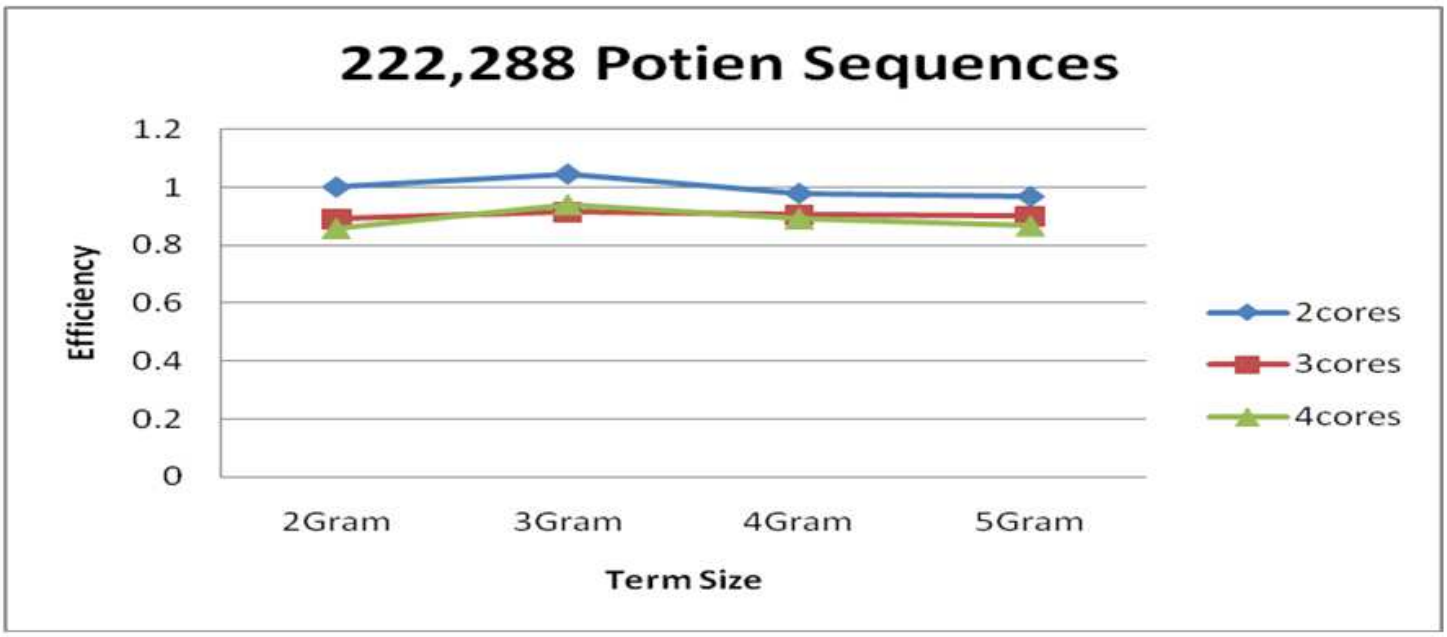

Figure 5

Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 97mb (200k protein sequences)

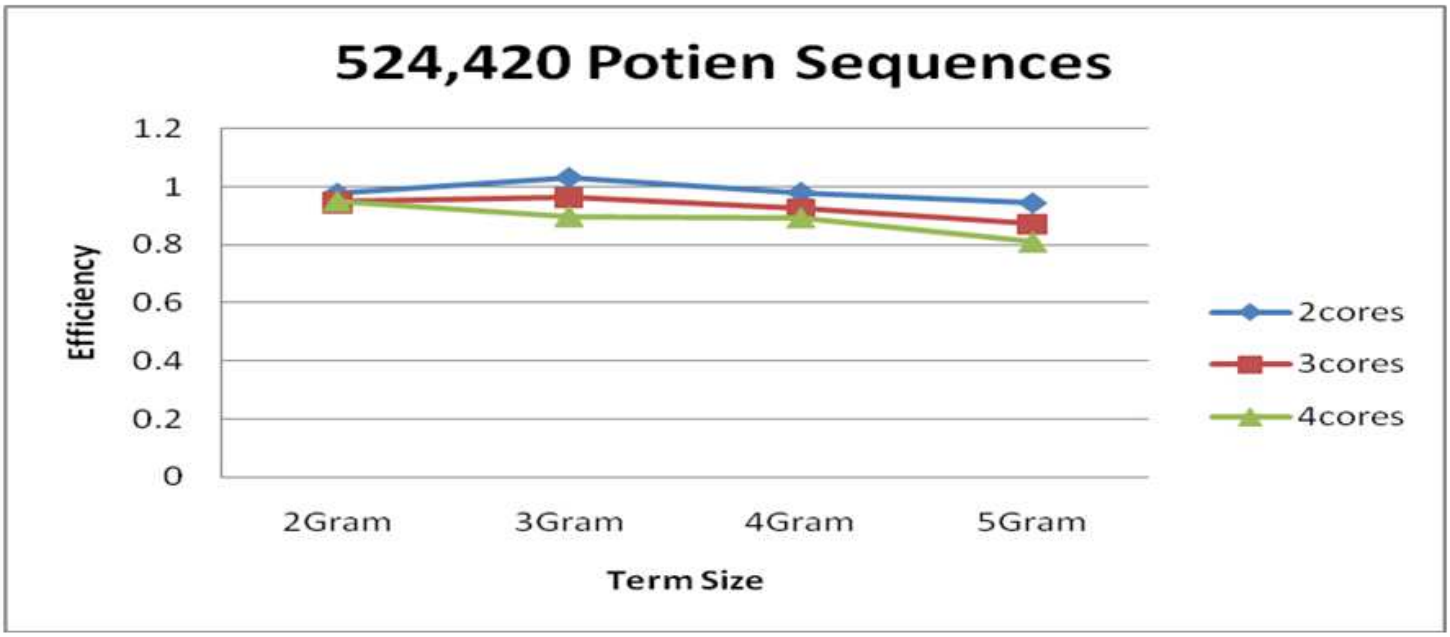

Figure 6

Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 234mb (500k protein sequences)

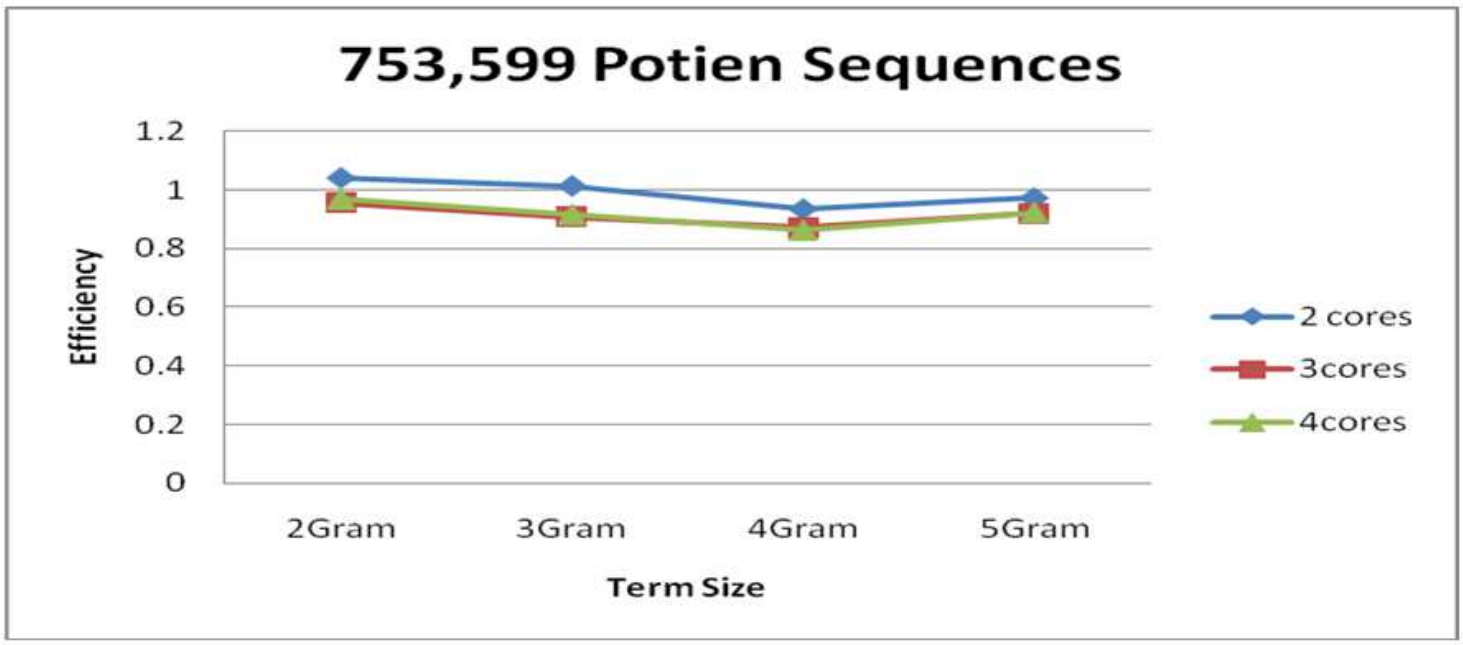

Figure 7

Parallel HT-NGH algorithm's efficiency on dataset's size that is equal to 339mb (753k protein sequences)

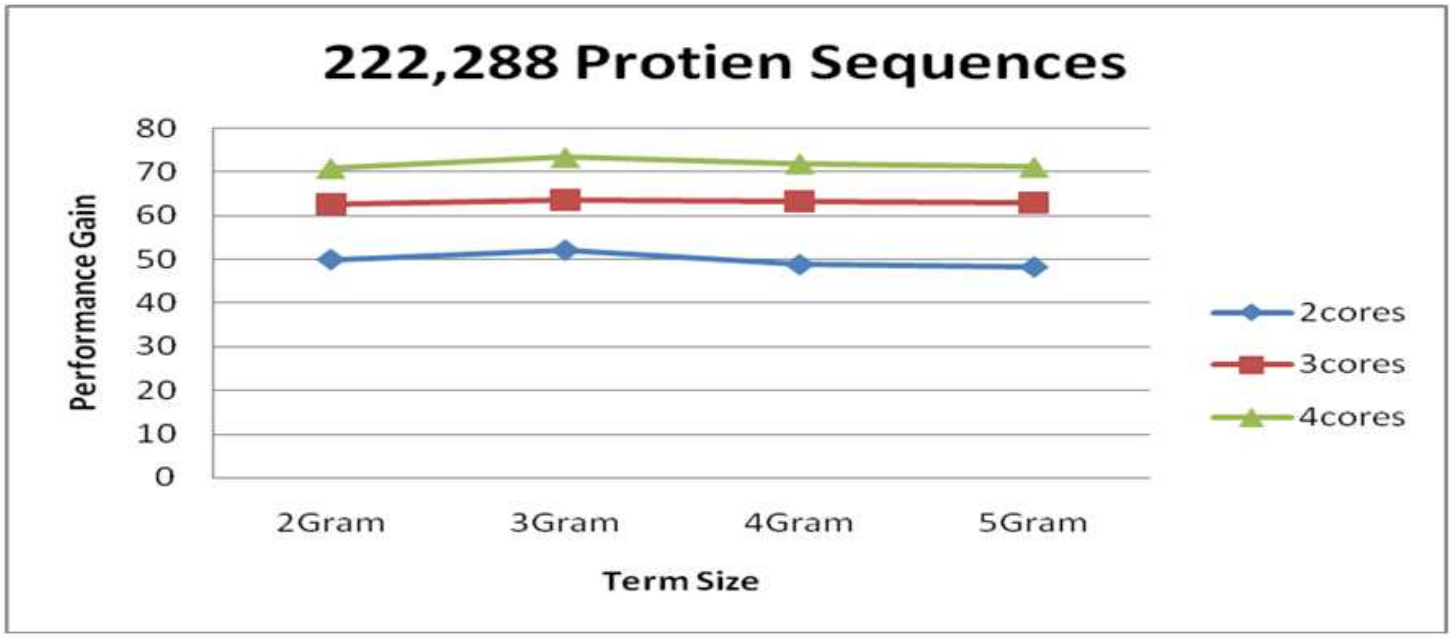

Figure 8

Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 97MB (200k Protein Sequences)

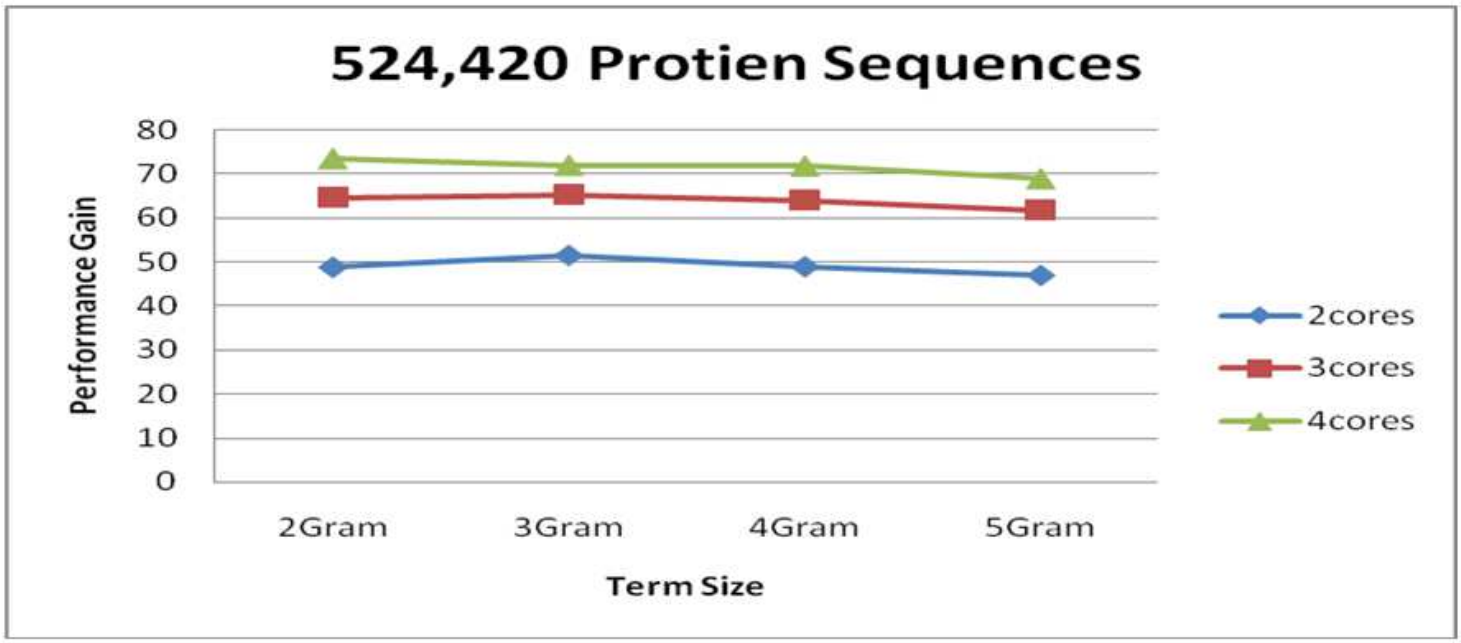

Figure 9

Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 234MB (500k Protein Sequences)

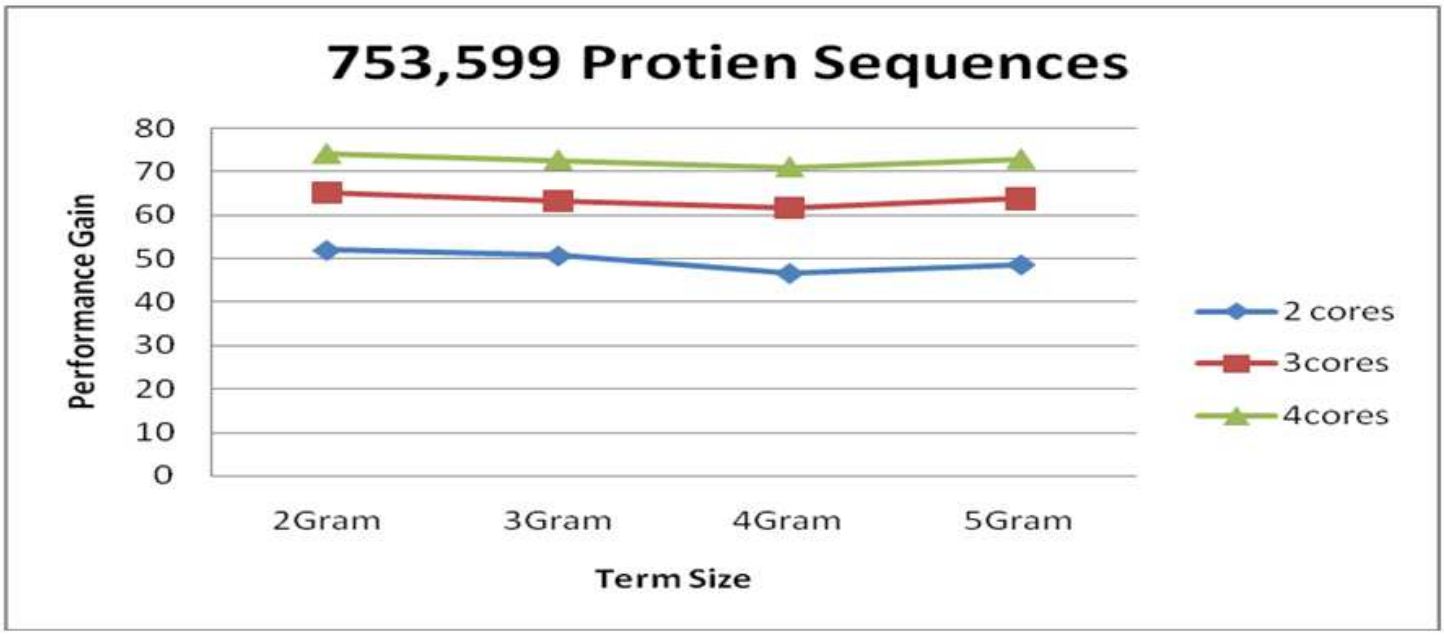

Figure 10

Parallel HT-NGH Algorithm's Performance Gain On Dataset's Size = 339MB (753k Protein Sequences)

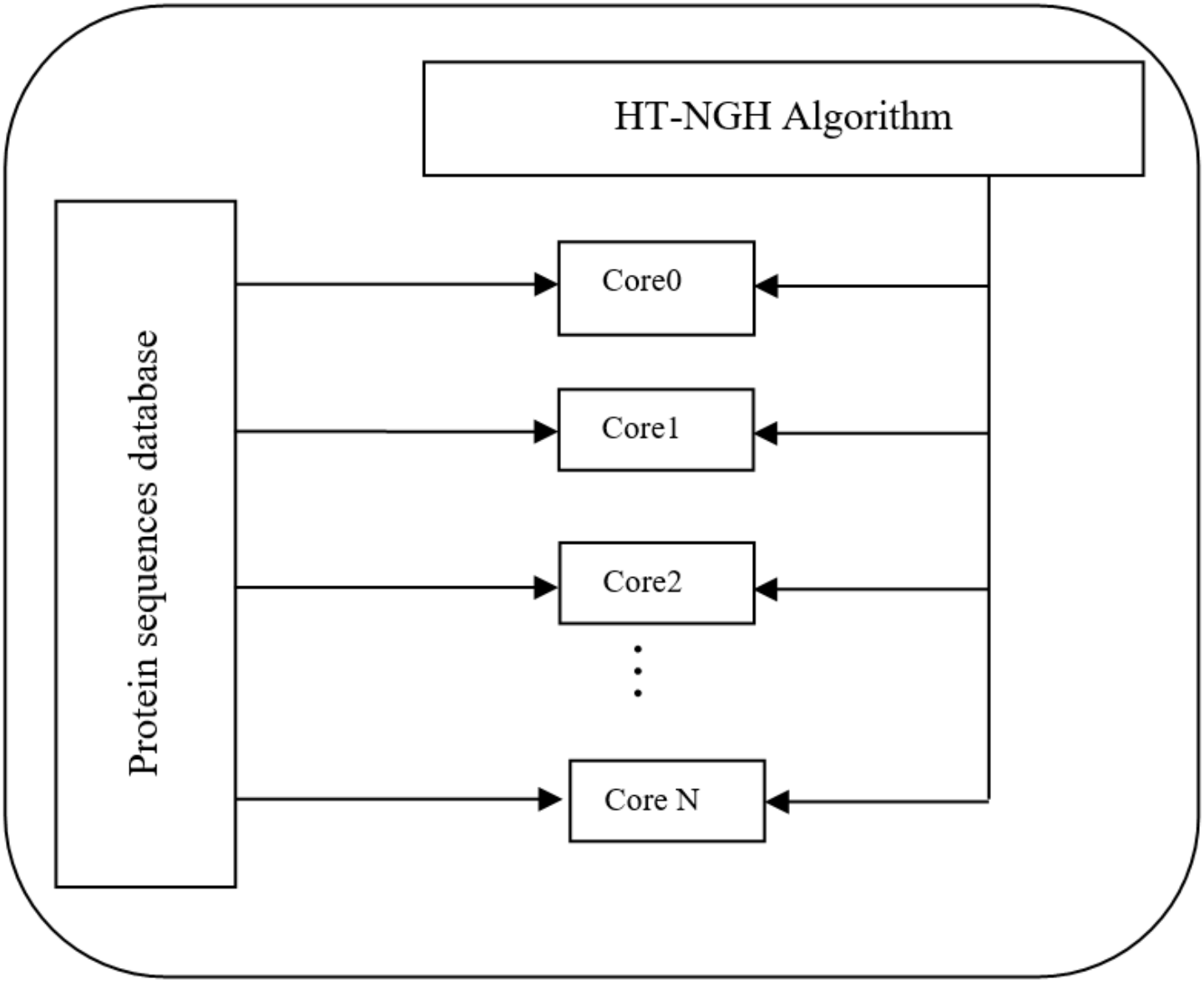

Figure 11

Data decomposition architecture

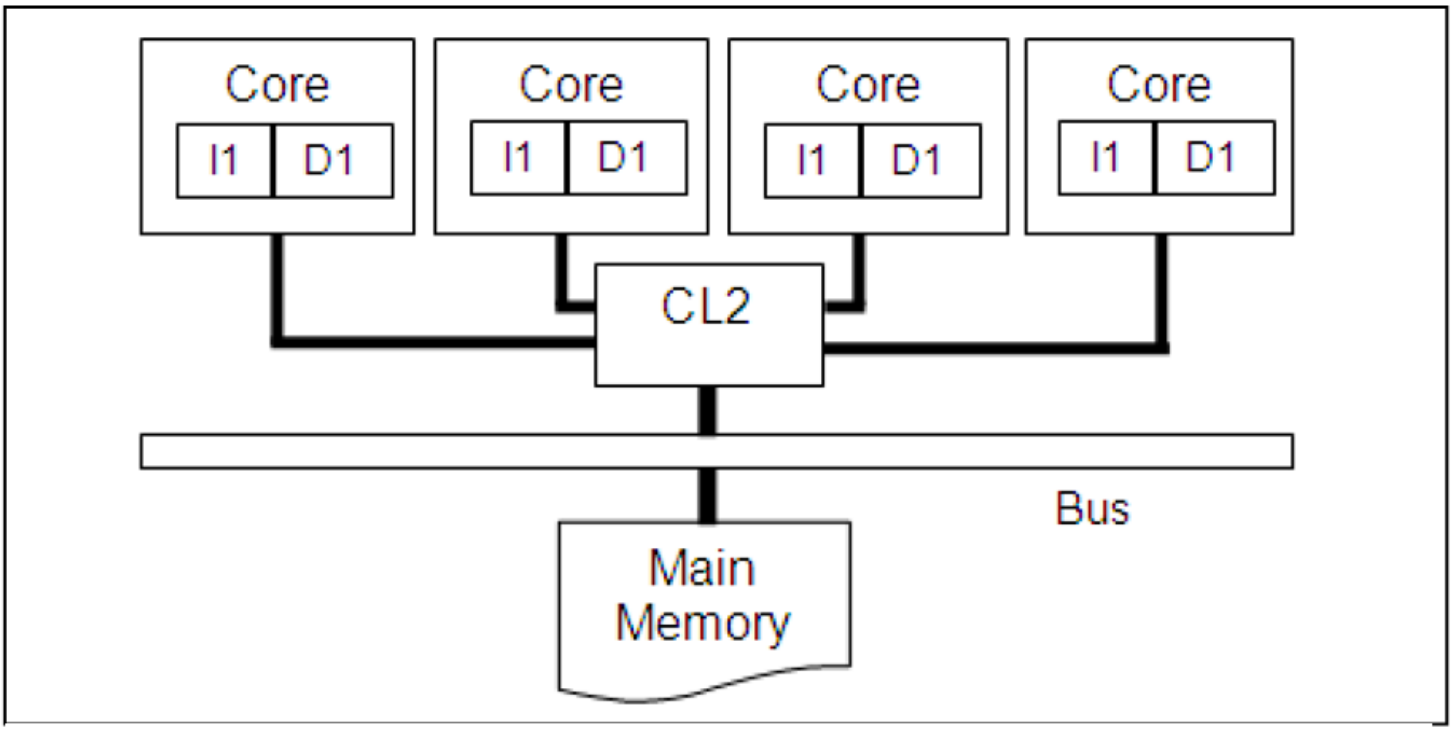

Figure 12

Schematic diagram of Intel quad-core architecture [36]

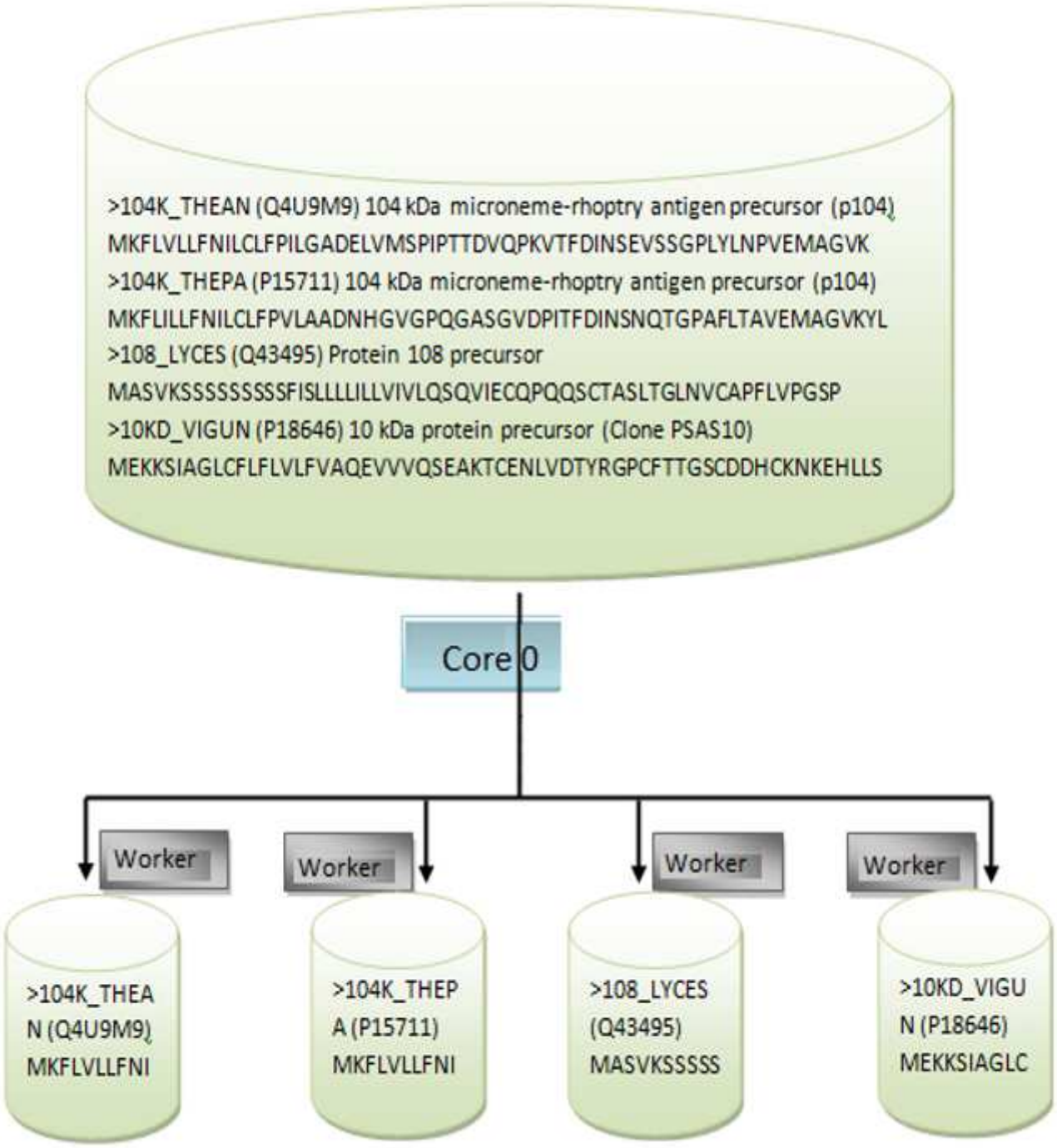

Figure 13

Dataset

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- AdditionalFile2.txt

- AdditionalFile5.cpp
- AdditionalFile8.cpp
- AdditionalFile7.cpp
- AdditionalFile6.cpp
- AdditionalFile11.cpp
- AdditionalFile9.cpp
- AdditionalFile10.cpp
- AdditionalFile4.cpp
- AdditionalFile1.fasta
- AdditionalFile3.txt