An Efficient Solution for Aligning Huge DNA Sequences

Ahmad M Hosny
Faculty of Computer and Information Sciences, Ain Shams University

Howida A Shedeed
Faculty of Computer and Information Sciences, Ain Shams University

Ashraf S Hussein
Faculty of Computer and Information Sciences, Ain Shams University

Mohamed F Tolba
Faculty of Computer and Information Sciences, Ain Shams University

ABSTRACT
Recently, many parallel solutions were proposed in order to accelerate the exact methods of aligning huge DNA sequences. However, most of these solutions restrict the sequence's sizes to be in kilobytes, in such a way that megabyte-scale genome comparison cannot be achieved. In addition, these solutions calculate only the alignment similarity score without finding the actual alignment. This paper presents an efficient solution to find the optimal alignment of the huge DNA sequences. This solution releases the condition of the sequence size to be in megabyte-scale instead of few kilobytes. The fundamental innovation in this work is developing efficient, linear space complexity, parallel solution to achieve the optimum alignment with relatively good performance. The shared memory parallel architecture is the focus of this work and therefore we have considered off-the-shelf systems like multi-core CPUs as well as advanced shared memory platforms. Experimental results show that, the proposed solution achieved high records compared to other solutions that targeted the same goal with less hardware requirements.

General Terms
Bioinformatics, High Performance Computing

Keywords
sequence alignment; space-efficient; parallel computing; multi-core.

1. INTRODUCTION
Sequence alignment is a fundamental operation in bioinformatics. It is a way of arranging DNA, RNA, or protein sequences to identify regions of similarity or difference. From a biological point of view, matches may turn out to be similar functions, e.g. homology pairs or conserved regions, while mismatches may detect functional differences e.g. Single Nucleotide Polymorphism (SNP). The alignment itself can be a global alignment (in which the complete sequences take part in the alignment) or a local alignment (in which only certain regions of each sequence that optimally align – are considered).

Local alignment is often preferable, but can be more difficult to find because of an additional challenge of identifying the regions of similarity.

The complexity due to the sheer number of possible combinations and searches makes the sequence alignment a very compute-intensive problem. Exact algorithms are based on dynamic programming. Needleman and Wunsch (NW) [1] presented the first global alignment algorithm. Smith and Waterman (SW) [2] improved this algorithm for local alignment to find the optimum common alignment according to a scoring function. These exact algorithms have a quadratic space and computational complexities with respect to the length of the two sequences. These quadratic complexities forbid their use for large-scale biological sequences. For example, aligning two sequences with one megabyte length each requires several terabytes of memory, which cannot be provided by most of the commodity computational resources. Therefore, most of the commercial applications use other algorithms based on heuristic approaches like Fasta [3] and Blast [4]. These heuristic approaches generally reduce the search space and make comparison of large genomic banks faster, but at the expense of a non-negligible reduction of algorithmic accuracy.

The challenge of quadratic space and time of the exact algorithms was addressed with many research groups along with the advent of High Performance Computing (HPC) revolution [5] [6]. There has been a plethora of new solutions that attempt to solve this problem. To comprehensively evaluate these contributions, we defined the problem challenges by the following metrics.

1.1 Functionality
Sequence alignment means calculating the maximum similarity score, then finding the actual alignment between sequences to detect the functional similarity or difference. It requires massive storage to be calculated for huge sequences. So, many solutions ignore finding the actual alignment to maximize the performance gain.

1.2 Performance
The quadratic complexity of the exact algorithms makes it a must to use parallelization to support larger sequences' sizes with reasonable computational time. The speed of computation is measured in Mega Cell Updates per Seconds (MCUPS). MCUPS= $\frac{m*n}{t} * 10^6$ Where m and n are the sequences sizes and t is the execution time.

1.3 Storage
Exact methods require quadratic space. Thus, for megabyte-scale sequences, terabytes are needed. Supporting huge sequences enforces space complexity to be linear.

1.4 Hardware Cost
The cost of the parallel computing solution and its availability is also one of the main metrics, as sequence alignment is a fundamental problem that needs available, cheap and commodity hardware. Solutions based on supercomputers, large scale computing clusters, or specially designed hardware are quite expensive.

The above four metrics can be considered as the main dimensions of the sequence alignment problem. The compromise between these metrics may lead to efficient solutions for the sequence alignment problem.

The remainder of this paper is organized as follows: Section-II discusses the recent related work. Section-III reviews SW algorithm and its parallel formulation(s). Section IV describes the purposed solution. Section V presents the experimental results. Finally, Section VI provides the conclusions of this work and future work directions.

2. RELATED WORK

The problem of obtaining an efficient implementation of SW and NW algorithms has been pursued by many research groups [5] [6].

A parallel framework with several multi-core implementations is proposed in [7]. The maximum supported sequence size was 1.25M * 0.2M. This framework adopted an intermediate-grained parallelism by dividing the query and database sequences among the cores. It calculates both score and alignment using a heuristic approach, which limits the number of processed cells to calculate the trace-back. Certainly, this limitation affects the solution's optimality.

Several implementations take advantage of the SIMD technologies like SSE2, SSE3, instructions available on Intel processors. Farrar [8] exploited the SSE2 instruction set to compute the SW algorithm in a striped pattern, outperforming the previous SIMD based SW implementations by Wozniak [9] and Rognes [10]. The striped pattern follows fine-grained parallelism in which, computations carried out in parallel in different stripes to reduce the impact of some of the computational dependencies. Farrar's implementation was then optimized by Rognes [11] to further enhancing the performance. Rognes implemented the stripped algorithm on SSE3 and Linux 64 bit. The experiments were scaled to include different databases which enhanced the overall performance. The different SIMD implementations achieved massive performance enhancement using the off the self processor. However, the maximum supported sequence size did not exceed a few kilobytes. Also this excellent performance is degrading with increasing the query size. Finally none of these SIMD implementations could calculate the trace-back, due to the followed stripped pattern.

Several parallel implementations using computer clusters were developed in [12], [13], [14] and [15]. These solutions divided the DP matrices into sets of columns or rows, which are assigned on a per-node basis. A set of multiple clusters is used in [12]; however, a maximum of 800K sequence is supported. A maximum of 1.1M sequence is supported in [13], using a cluster of 60 nodes with poor performance. A heuristic solution to align two 400K sequences is introduced in [14], but without any guarantee for the optimality and with somewhat weak performance. A parallel exact solution to produce local alignment is proposed in [15]. It can align up to 3 MB sequences using a cluster of 16 processors.

Hirschberg reported the first global alignment algorithm with linear space complexity [16]. Hirschberg's algorithm is improved in [17] by recording some rows and columns to reduce the re-computations. Improved implementation is proposed in [18] by recording a limited number of cached columns or anti diagonals (a maximum of 16). Due to this limitation, the recomputed areas were huge and the maximum query supported was only 300K

Recently, the main trend is to use hardware accelerators to implement SW algorithm like Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs). FPGAs have been used to implement SW in many solutions (such as [19] and [20]). They presented impressive speedups over software implementations. However, they are still not considered to be commodity hardware and their programming interface is rather complex. Due to the limited storage, FPGAs cannot produce the alignment for huge sequences.

GPUs have a massively parallel architecture. With GPUs, impressive speedups can be achieved using a programming model that is simpler than the one required for FPGAs. The on-chip memory of the GPU is limited. The main memory or the hard disk cannot be used as an alternative because the communication with the CPU is too expensive in terms of the communication time. These limitations make GPUs impractical to be used in case of large scale sequences alignment. Thus, most of the recent works based on them (such as [21] and [22]) were enforced to use coarse-grained parallelization with small query sizes and without finding the alignment.

From the above survey we can conclude that, most of the relevant research work contributes only to a subset of the four metrics, defined in Section I, at the expense of the remaining ones. This paper presents an optimum solution for aligning huge DNA sequences, which compromises between all the four metrics. Thus, an efficient, linear space complexity, parallel solution is developed to achieve the optimum alignment for huge DNA sequences with a relatively good performance.

3. SMITH–WATERMAN ALGORITHM

The algorithm used to calculate the optimal local alignment is the Smith–Waterman (SW) algorithm with the Gotoh (1982) improvements for handling multiple sized gap penalties. SW is an exact method based on dynamic programming to obtain the best local alignment between two sequences in quadratic time and space.

3.1 The Algorithm

Consider two sequences Q and D of length m and n. The individual residues for Q and D are $q_1, q_2 \ldots q_m$ and $d_1, d_2 \ldots d_n$, where $1 \le i \le m$ and $1 \le j \le n$. A scoring matrix P (q_i, d_j) is defined for all residue pairs. A constant value may be assigned to gaps. The penalties for opening and extending a gap are defined as: G_{init} and G_{ext}. The algorithm is divided into two phases: Calculating the dynamic programming matrices and finding the best local alignment.

3.1.1 Phase 1

Calculating the Dynamic Programming (DP) Matrices, at the beginning, the first row and column are filled with zeroes. The remaining elements of H are obtained from equations (1), (2) and (3) The values for Hi, j, Ei, j and Fi, j are defined as 0 where $i < 1$ or $j < 1$.The similarity score between sequences Q and D is the highest value in H and the position (i, j) of its occurrence represents the end of the alignment. In order to calculate the trace-back, only the arrows' directions need to be stored in the matrix cells. A left arrow in Hi, j indicates the alignment of Q[i] with a gap in D. An up arrow represents the alignment of D[j] with a gap in Q. Finally, an oblique arrow indicates that Q[i] is aligned with D[j].

3.1.2 Phase 2

Finding the best alignment, In order to find the best local alignment, the algorithm starts from the cell that contains the highest score value and follows the arrows until a zero-valued cell is reached.

$$H_{i,j} = max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} - P(i,j) \end{cases} \quad (1)$$

$$E_{i,j} = max \begin{cases} E_{\text{i,j-1}} - G_{ext} \\ H_{i,j-1} - G_{first} \end{cases} \quad (2)$$

$$F_{i,j} = max \begin{cases} E_{\text{i-1,j}} - G_{ext} \\ H_{i-1,j} - G_{first} \end{cases} \quad (3)$$

3.2 The Algorithm Data Dependency and Parallelization

The challenge in implementing parallelism at the similarity matrix is the data dependency. Any cell of the alignment matrix can be computed only after computing the values of the Northern, Western, and North-Western cells. The access pattern presented by the matrix calculation is non-uniform. So, the traditionally used parallelization strategy, in this kind of problems, is the wave front method [6]. In this manner, cells can be only processed in parallel if they are on the same anti-diagonal in a wave front pattern as depicted in Figure 1.

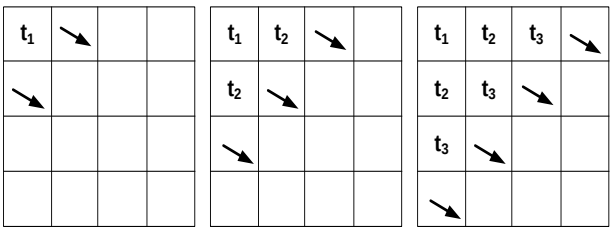

Fig 1: Wave front execution. Each step of these three steps calculates a diagonal

4. THE PROPOSED SOLUTION

This research work is concerned with the development of a solution that can produce the optimum local score and alignment between two megabyte-scale sequences with a relatively good performance. Shared memory parallel architecture is the focus of this work, and therefore we have considered off-the-shelf systems like multi-core CPUs as well as advanced shared memory platforms.

To put the challenges in perspective, consider producing the optimal alignment between a pair of 5M sequences using SW algorithm. Assume that each matrix cell is two bits; it holds a direction (that can be up, left, oblique or none). Thus, 25 terabytes of memory are required, which are not normally available in any commodity hardware. Therefore, linear memory solution is a must, in addition to the huge runtime that should be parallelized.

In the proposed solution, the computation adopts an affine gap. So, we need to compute the three matrices E, F and H. These three matrices are logically grouped into a single matrix M. Each cell Mi, j contains the three values Hi, j, Ei, j and Fi, j, where each is declared as an unsigned integer (4 bytes), to support huge sequence sizes. Two additional bits are added to store the directions then the total cell's size is 12.25 bytes.

In order to reduce this huge memory requirement, it is important to point out that calculating any anti-diagonal in the logical matrix M is dependent only on the values of the previous two anti-diagonals. We define a Key Anti-Diagonal (KAD) as a pair of consecutive anti-diagonals in M whose cells values are saved. Let x be the number of such KADs. The proposed solution is based on saving x KADs that are normally distributed all over the M matrix. These x KADs can be used to recalculate any needed cell. Recalculation starts from the cell with maximum score to the nearest KAD then continues towards the alignment start cell. Increasing x reduces the recalculated area till reaching the target cell. In order to increase x, we can store KADs on the HDD (instead of main memory). The phases of the proposed solution can be divided into forward and backward phases that will be explained in the next subsections.

4.1 Forward Phase

In this phase, the wavefront method is applied to iterate in parallel over all cells in every anti-diagonal at M. The pseudo-code of this phase is shown in Algorithm 1. The x KADs are calculated and saved to the HDD. All anti-diagonals' indices that contribute to one of these x KADs are saved in the main memory. For each anti-diagonal, all cores compute all cells' values in parallel. If the anti-diagonal index is one of the x KADs, then its M values and directions will be saved to the HDD (see Fig. 2). During the iteration process, the cell with maximum H score and its coordinates will be updated.

4.2 Backward Phase

This phase executes a trace-back function to find the actual alignment. The pseudo-code of this phase is shown in Algorithm 2. It starts from the end of the forward phase, from the cell with the maximum score at H and moves backwards to find the alignment's starting point. The areas containing the trace-back points are to be recalculated using KADs saved to the HDD before. Because we can move only left, up or left-up, the recalculated area always forms a triangle (or maybe a trapezoid, if cut by a border) whose apex is the maximum point, and whose base lies on the nearest next KAD. This triangle is recomputed in parallel, starting from its base towards its apex, using the wavefront method. After that, the trace-back continues from the maximum score, found on the base of the previous triangle, till it intersects with a point on the nearest saved KAD (The base of the current triangle). This process is repeated until the trace-back reaches the endpoint (The first encountered cell with a zero value). See Fig. 3 for a depiction of this process.

ALGORITHM 1 FORWARD PHASE

```
Procedure Produce KADs (Q, D)

1:   define AD1 and AD2 buffers for latest pair of anti-diagonals
2:   For each anti-diagonal AD in Matrix H
3:     Start dynamic balanced parallel for each
4:     For each cell C(i,j) in AD do
5:       Get Previous Values from AD1 and AD2;
6:       Calculate values H(i,j) ,E(i,j) , F(i,j) and directions;
7:       Update max score H(i,j)
8:     end parallel for each
9:     update AD1 and AD2;
10:    if AD1 and AD2 are one of x KADs
11:      start asynchronous task
12:        Save AD1 and AD2 to HDD;
13:    end for each
14:    return max score coordinates
END Procedure
```

ALGORITHM 2 BACKWARD PHASE

```
Procedure Trace-back ()

1:   define Current Cell with the max score cell
2:   start loop till Current Cell equals null
3:     define triangle buffer t;
```

```
4:      t apex is the CurrentCell and base is on the nearest KAD
5:      THREADS:= Get All workers
6:      Start dynamic balanced parallel foreach with THREADS
7:      For each anti-diagonal AD in t
8:          Start dynamic balanced parallel for each with threads
9:          For each cell C(i,j) in AD do
10:             get previous values from t;
11:             calculate the direction for C(i,j);
12:         end parallel for each
13:     end for each
14:     start trace-back from Current Cell to the base of t
15:     Current Cell equals the intersected cell with t base.
16: end loop
END Procedure
```

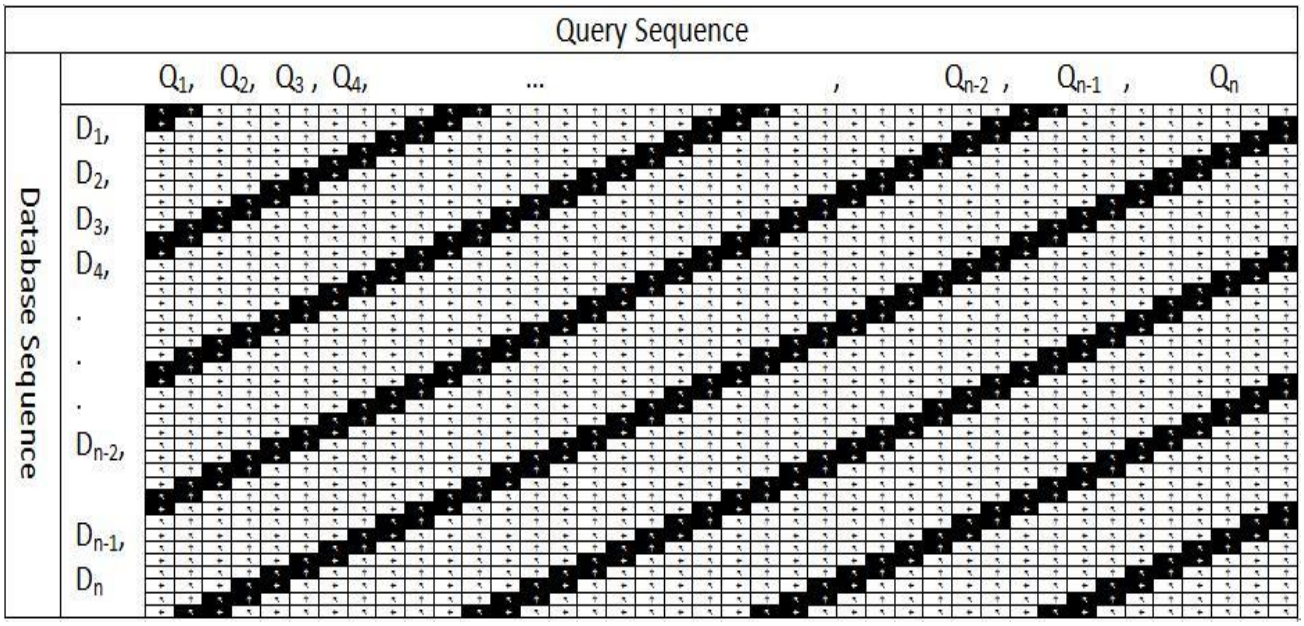

Fig 2: Forward Phase. Cells are processed in a wavefront pattern. x = 8 KADs. Black anti-diagonals represent KADs that are saved to the HDD. White anti-diagonals are discarded during iterating

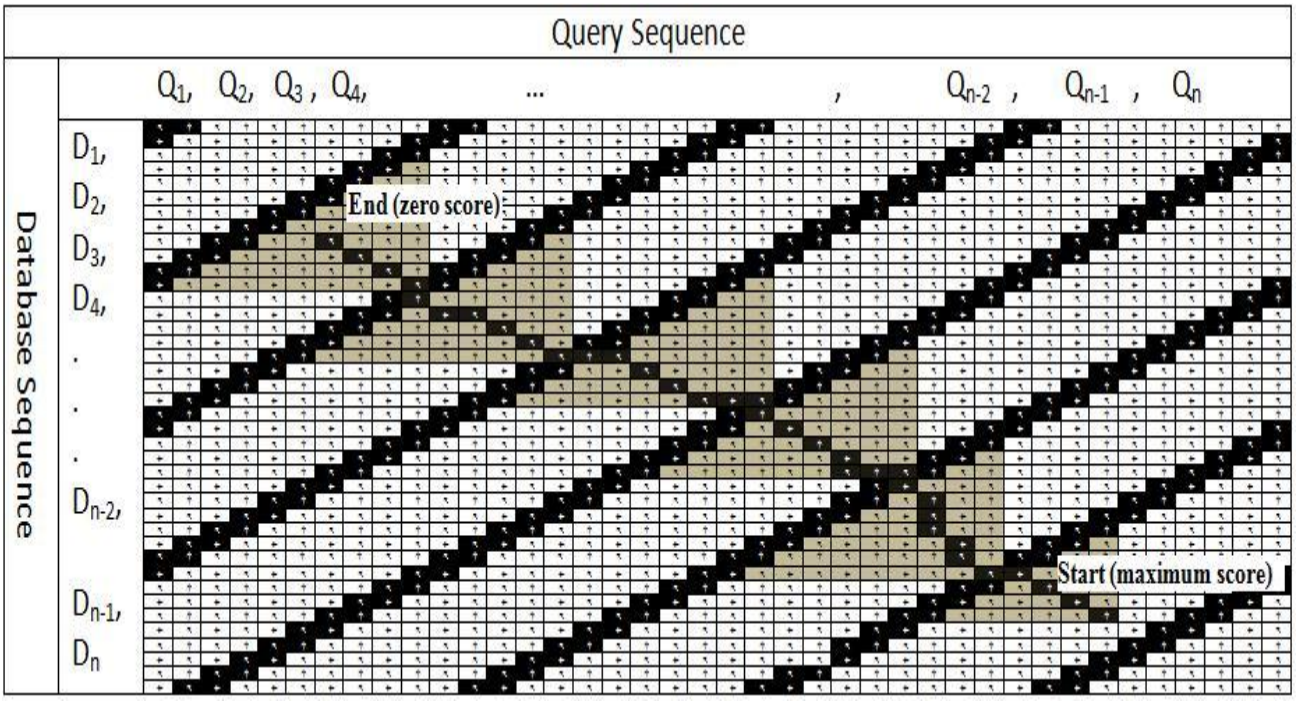

Fig 3: Backward phase. It starts from the cell with the maximum score and recalculates the triangle (dark-gray cells), Then the trace-back continue till the intersection with the next KAD (black cells represent the actual trace-back). These two steps are repeated till finding an empty cell, which is taken to be the end point

5. RESULTS AND DISCUSSION

The proposed solution was implemented using C++ and OpenMP and tested using an 8-core CPU of 1.6 MHz and 4MB cache, with 4 GB RAMS running on Windows Server 2007. Visual studio 2010 was used as a development environment. In order to evaluate the scalability of the present solution, nucleotides of exact sizes ranging from 32K to 5M long were generated for measuring the scalability using

precise figures. The SW score parameters used in the tests were: +1 for match; –1 for mismatch; –2 for first gap; and –1 for gap extension. The used user-configured HDD storage is 50 GB. Query and database sequences with the best MCUPS obtained for different numbers of CPU cores are shown in Fig. 4.The performance raised with increasing the sequence size and the number of cores because the huge number of parallel items allowed a better load balancing between the cores and reduced the communication time.

We measured the average load for each working processor during aligning 5M x 5M sequences on 8 cores the results are shown in Fig. 5. Dynamic load balancing is used in allocating the parallel iterations to the processors. However, the loads may not be symmetric because there is a parallel task that writes the KADs to the HDD and the number of items varies from iteration to the other. Fig.6 shows the variations of the speedup achieved when the number of working processors increases from 1 to 8. The results show that the proposed solution, with the proposed parallel design, scales linearly with the number of working processors.

The proposed solution achieved excellent MCUPS records which surpassed other exact solutions that targeted the same goal with less hardware requirements. This can be inferred by comparing our results to the results in table I. We can align pairs of up to 5M sequences and can support more with upgrading the hardware.

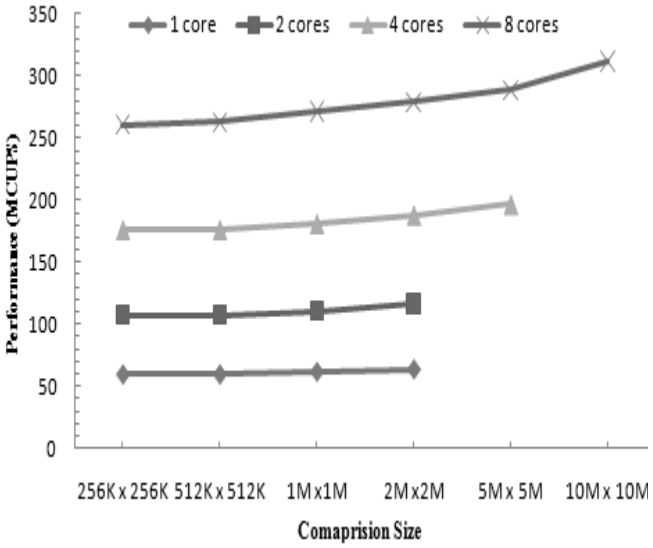

Fig 4: Resulting performance in MCUPS obtained from the experiment with different sequence sizes

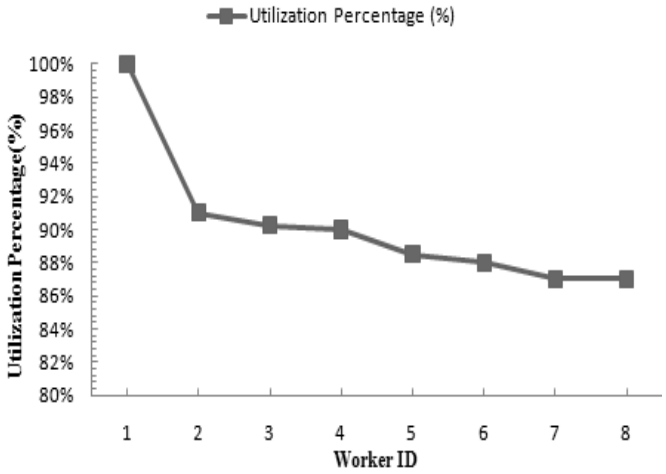

Fig 5: The load balance for aligning 5M x 5M sequences on 8 cores

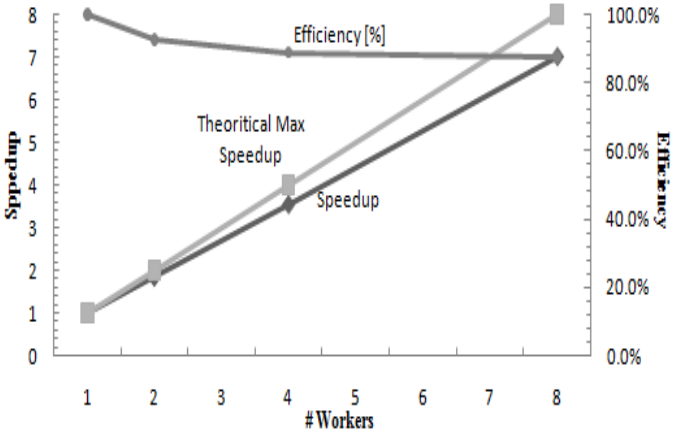

Fig 6: Resulting speedup and efficiency obtained for the parallel execution with different number of workers

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed and evaluated a multi-core-accelerated implementation of the Smith-Waterman (SW) algorithm with affine gap that compares two megabyte-scale genomic sequences. As opposed to the previous solutions based on SW optimum algorithm, the proposed solution does not impose severe restrictions on the size of the largest sequence. Thus experimental results show that we can support up to 5M sequence using our simple hardware configuration, with a relatively very good performance. Larger sequences can also be supported by a small upgrading of the hardware.

7. REFERENCES

[1] C.D. Wunsch S.B. Needleman, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," Journal of Molecular Biology, vol. 48, no. 3, pp. 443-453 , March 1970.

[2] M.S. Waterman T.F. Smith, "Identification of common molecular subsequences," Journal of Molecular Biology, vol. 147, no. 1, pp. 195-197, March 1981.

[3] William R. Pearson, "Searching protein sequence libraries: Comparison of the sensitivity and selectivity of

the Smith-Waterman and FASTA algorithms," Genomics, vol. 11, no. 3, pp. 635-650, November 1991.

[4] Altschul SF, "Gapped BLAST and PSI-BLAST: "A new generation of protein database search programs"," Nucleic Acids Res, vol. 25, no. 17, pp. 3389-3402, 1997.

[5] T. Majumder, A. Kalyanaraman, P. Pande S. Sarkar, "Hardware accelerators for biocomputing: A survey ," Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, pp. 3789-3792, 2010.

[6] Sanjay Kumar Pandey ,Digvijay Pandey Binay Kumar Pandey, "A Survey of Bioinformatics Applications on Parallel Architectures," International Journal of Computer Applications, vol. 23, no. 4, pp. 21-25, 2011.

[7] Nuno Filipe Valentim Roma Tiago José Barreiros Martins de Almeida, "A Parallel Programming Framework for Multi-core DNA Sequence Alignment," in International Conference on Complex, Krakow, Poland , 2010, pp. 907-912.

[8] Michael Farrar, "Striped Smith–Waterman speeds database searches six times over other SIMD implementations," Bioinformatics, vol. 23, no. 2, pp. 156–161, 2007.

[9] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," Comput Appl Biosci, vol. 13, no. 2, pp. 145–150, 1997.

[10] Erling Seeberg Torbjørn Rognes, "Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors," Bioinformatics, vol. 16, no. 8, pp. 699–706, 2000.

[11] Rognes Torbjørn, "Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation," BMC Bioinformatics, vol. 12, no. 1, p. 221, 2011.

[12] B. Schmidt C. Chen, "Computing Large-Scale Alignments on a Multi-Cluster," Fifth IEEE International Conference on Cluster Computing (CLUSTER'03), pp. 38 - 45, 2003.

[13] S. Aluru S. Rajko, "Space and time optimal parallel sequence alignments," vol. 15, no. 12, pp. 1070-1081, 2004.

[14] Alba Cristina Magalhaes Alves de Melo, Mauricio Ayala-Rincon , Thomas M. Santana Azzedine Boukerche, "Parallel Smith-Waterman Algorithm for Local DNA Comparison in a Cluster of Workstations," in Experimental and Efficient Algorithms. Berlin : Springer, 2005, vol. 3503, pp. 131-144.

[15] A. Boukerche, , A. C. Melo R. B. Batista, "A parallel strategy for biological sequence alignment in restricted memory space," Journal of Parallel and Distribituted Computing, vol. 68, no. 4, pp. 548-561, April 2008.

[16] D.S. Hirschberg, "A linear space algorithm for computing longest common subsequences," Communications of the ACM, vol. 18, no. 6, pp. 341-343, June 341- 43.

[17] C. Xu, T. Wang, L. Jin and Y. Zhang E. Li, "Parallel Linear Space Algorithm for Large-Scale Sequence Alignment," in Euro-Par 2005 Parallel Processing. Berlin: Springer, 2005, vol. 3648, pp. 644-644.

[18] D. Fan, W. Lin X. Ye, "A fast linear-space sequence alignment algorithm with dynamic parallelization framework," in IEEE Ninth International Conference on Computer and Information, 2009, pp. 274-279.

[19] H. K. Tsoi, W. Luk Y. Yamaguchi, "FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design," in Reconfigurable Computing: Architectures, Tools and Applications. Berlin: Springer, 2011, vol. 6578, pp. 181-192.

[20] P. Zhang, N. Sun, X. Jiang, X. Liu. L. Xu, "A reconfigurable accelerator for smith-waterman algorithm," Circuits and Systems II: Express Briefs, IEEE Transactions on, vol. 54, no. 12, pp. 1077 - 1081 , December 2007.

[21] Valle Giorgio Manavski Svetlin, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," BMC Bioinformatics, vol. 9, no. 2, pp. 1-9, March 2008.

[22] Schmidt Bertil, Maskell Douglas Liu Yongchao, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," BMC Research Notes, vol. 3, no. 1, pp. 1-12, 2010.