Acceleration of Biological Sequence Alignment using Recursive Variable Expansion

Zubair Nawaz[1], Mudassir Shabbir[2], Zaid Al-Ars[1], Koen Bertels[1]
[1]Computer Engineering, Delft University of Technology, The Netherlands
{Z.Nawaz, z.al-ars, K.L.M.Bertels}@tudelft.nl
[2]Mentor Graphics, Pakistan
mudassir_shabbir@mentor.com

***Abstract*—Biological sequence alignment is one of the most important problems in computational biology. Given two sequences of varying length are aligned such that the alignment score is maximum. The alignment score is calculated based on the number of matches, mismatches and gaps in the alignment suggested. The basic sequence alignment algorithms are Needleman-Wunsch (NW) algorithm and Smith-Waterman (SW) algorithm, which find the optimal global and local alignment, respectively. In this paper, we have accelerated the Needleman-Wunsch by applying the Recursive Variable Expansion partially, which can be easily implemented on FPGA. This method extract more parallelism than other prevalent parallel implementations and we are able to achieve a speed up of 1.55 times.**

***Index Terms*—Reconfigurable Computing, Parallelization, Pattern Matching, Computational Molecular Biology, Needlman-Wunsch**

I. Introduction

Biological sequence alignment is one of the most important problems in computational biology. A biological sequence with unknown functionalities is compared to a biological sequence with known functionalities to infer the functions of the former sequence. The objective is to find an optimal alignment between two sequences of different length such that the alignment score is maximum. The two sequences are aligned and an alignment score is calculated based on the number of matches and gaps in the alignment.

The sequence alignment problem is further divided into global and local alignment. In global alignment, the optimal score and match is found between the two entire sequences, whereas in local alignment, the optimal score and match is found between some longest subsequences of the two sequences. The basic sequence alignment algorithms are Needleman-Wunsch (NW) algorithm [1] and Smith-Waterman (SW) algorithm [2], which find the optimal global and local alignment, respectively. Both algorithms are based on dynamic programming, are very similar to each other and both have time and space complexity *O(mn)*, where *m* and *n* are lengths of the sequences being aligned. Although this complexity seems to be desirable, the exponential growth in bio-sequence databases of known sequences makes this complexity intolerable [3], [4]. Therefore as the database size grows larger, faster algorithms to quickly compare and align the sequences become important.

One way to avoid such expensive solution is to use heuristic techniques like FASTA [5] and BLAST [6]. Both compute the local alignment and are fast and less sensitive than Smith-Waterman, as the time complexity is reduced at the cost of accuracy. Therefore an optimal alignment may not always be found through these techniques. Other way to reduce the time complexity is to accelerate the Needleman-Wunsch and Smith-Waterman algorithms through parallel processing. Researchers have been able to parallelize both algorithms on parallel machines [7], [8], [9]. However, keeping in view the growing size of the database, prevalent methods require further acceleration to match the growth.

There is always a need for a fast method, which can extract more parallelism than before. A major hurdle in achieving this is the bound on the maximum parallelization, depending upon the available hardware or inherent degree of parallelism available in the algorithms. In this paper, we will accelerate Needleman-Wunsch by extending the degree of parallelism available in the algorithm and utilizing the hardware to its full capacity.

The rest of the paper is organized as follows. In the next section, we discusses the background and related work for the parallelization of Biological Sequence Alignment. Our approach is discussed in Section III, furthermore, the comparison of the time estimates for our approach and prevalent parallel techniques is discussed. Finally we conclude the paper in Section IV.

II. Background and Related Work

A. Background

Let *S[1..n]* and *T[1..m]* be two sequences of length *n* and *m* for sequence alignment. The *optimal alignment score F(i, j)* for two subsequences *S[1..i]* and *T[1..j]* is given by the following recurrence equation.

$$F(i,j) = \max \begin{cases} F(i,j-1) + g \\ F(i-1,j-1) + x(i,j) \\ F(i-1,j) + g \end{cases} \quad (1)$$

where $F(0,0) = 0, F(0,j) = g \times j$ and $F(i,0) = g \times i$, for $1 \leq i \leq n, 1 \leq j \leq m$. The g is the penalty for inserting a gap in any of the sequence and $x(i,j)$ is the score for match/mismatch, depending upon whether *S[i] = T[j]* or *S[i]* $\neq$ *T[j]*.

In the remainder of the paper, we will use $g = -2$ and $x(i,j) = +1$ when *S[i] = T[j]*, otherwise -1. The recurrence

		G	A	C	G	G	A
	0	-2	-4	-6	-8	-10	-12
G	-2	**1**	-1	-3	-5	-7	-9
A	-4	-1	**2**	0	-2	-4	-6
T	-6	-3	**0**	1	-1	-2	-4
C	-8	-5	-2	**1**	0	-2	-4
G	-10	-7	-4	-1	**2**	1	-2
G	-12	-9	-6	-3	0	**3**	1
A	-14	-11	-8	-5	-2	1	**4**

Figure 1. Scoring Matrix for an example of Needleman-Wunsch Algorithm

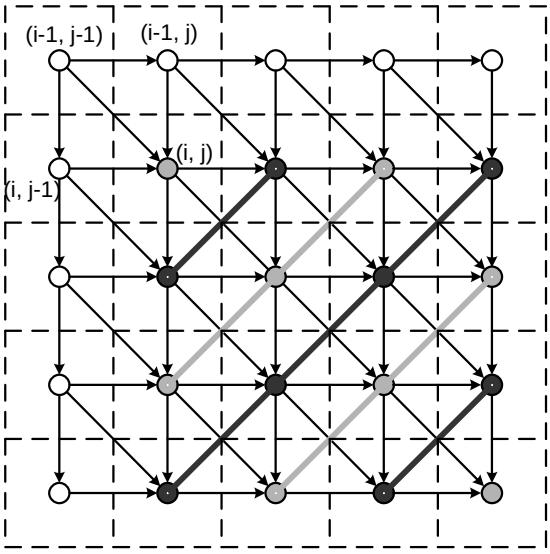

Figure 2. Data dependence graph for both equations, different shades of gray in circles show the elements which can be executed in parallel

Equation 1 shows the optimal substructure and overlapping subproblems in the problem, therefore dynamic programming [10] is applied to get an efficient solution. In dynamic programming, a bottom-up approach is used, in which initially the boundary conditions are computed and then *F* is computed from smaller subsequences to larger ones till it reaches the entire length of the sequences. An example of Needleman-Wunsch algorithm is shown in Figure 1, where a matrix is made and the two sequences are put along the row and column. The matrix is filled using Equation 1 from the top-left corner and elements are filled from left to right and from top to bottom. In the filled matrix the bottom-right element gives the maximum global alignment score for two sequences. Once the whole matrix is filled, we start a trace back from the bottom-right element to one of the three elements from which alignment score is calculated till it reaches the top-left corner. In the trace back, when an element is computed from the top element then there is a gap in the sequence along the row and similarly when an element is computed from the left element then there is a gap in the sequence along the column. The optimal alignment for the example is given below.

```
GA-CGGA
|||||||
GATCGGA
```

An operation is primitive, if the time taken by this function is independent of the operands. As all the operations given by the Equation 1 are primitive, the computation of optimal alignment score *F(i, j)* takes constant time, and since there are $m \times n$ elements to be computed, the time complexity for both the algorithms is *O(mn)*. Likewise, we need to keep the table of size $m \times n$ to compute the *F(i, j)* and for trace back, therefore the space complexity for the algorithms is also *O(mn)*.

To parallelize NW algorithm we need to look at their data dependence graphs as shown in Figure 2. Blank circles are the elements after the initialization with the boundary conditions. Any iteration *(i, j)* cannot be executed until iterations *(i-1, j)*, *(i-1, j-1)* and *(i, j-1)* are not executed first, due to data dependencies. Therefore we need to change the way the elements are traversed like starting from the top, elements with one shade of gray in a diagonal line can be executed in parallel followed by the next diagonal line with different shade of gray due to dependency constraint. Therefore the degree of parallelism is constrained to the length of the diagonal line and the maximum number of processing elements required will be equal to the length of longest diagonal. The number of steps required in such implementation is *O(n+m)* [8].

B. Recursive Variable Expansion

Recursive Variable Expansion (RVE) [11] is a kind of loop transformation which removes all the data dependencies from the program and then the program is prone to maximum parallelism. The basic ideas is that if any statement S_i is dependent on statement T_j for some iteration *i* and *j*, then instead we wait for T_j to complete and then execute S_i, we will replace all the occurrences of the variable in S_i that creates dependency with T_j with the computation of that variable in T_j. This way there is no need to wait for the statement T_j to complete and statement S_i can be executed independently of T_j. Similarly if T_j is dependent on some other statement, we will replace the computation of that statement with the variable to make it independent of that statement. This step is recursively repeated until the statement S_i is not dependent on any statement rather only inputs or known values, which essentially means that S_i can be computed with out waiting for the computation of any other statement. The technique is very beneficial when most of the operations are associative. In this way, the whole expanded statement can be computed in any order by computing the large number of operations in parallel and efficiently using binary tree structure. The major draw back of this technique is that the speed up is achieved at the cost of redundancy, which consumes very large resources.

C. Related Work

A lot of work has been done to accelerate the biological sequence alignment using different hardwares. In addition to

specific architectures designed for sequence alignment, many solution for special purpose hardware, SIMD and FPGAs have been devised.

A particular algorithm can be implement using special purpose hardware and it also provides result faster, however they lack the flexibility to run different algorithms required for biological sequence alignment. Some examples of such implementation are P-NAC, BISP and SAMBA [12], [13], [14].

Several implementations for SIMD have been proposed as MGAP, Kestrel and Fuzion [15], [16], [17]. SIMD contains general purpose processors (GPP) therefore it is programmable and is used for a wider range of applications like image processing and scientific computing. The drawback is that they are expensive.

Field Programmable Gate Arrays (FPGAs) are good candidates for accelerating biological sequence alignment algorithms. FPGAs are programmable using some hardware description languages like VHDL or Verilog and virtually any algorithm can be mapped on it. FPGAs can also be reconfigured during system operation, called Run-Time Reconfiguration (RTR), which makes it suitable if the algorithm or gap penalty is changed at runtime. Some of the solutions based on FPGAs are given in [18], [19], [20].

III. Our Approach and Time Estimation

When we apply recursive variable expansion partially on Equation 1, the recursion tree is shown in Figure 3. *F(n, m)* can be written in equation as shown by the leaf nodes in Figure 3.

$$F(n,m)=\max\begin{cases} i & F(n,m-2)-4 \\ ii & F(n-1,m-2)-2+x(n,m-1) \\ iii & F(n-1,m-2)-6 \\ iv & F(n-2,m-2)-4+x(n-1,m-1) \\ v & F(n-2,m-1)-6 \\ vi & F(n-1,m-2)-2+x(n,m) \\ vii & F(n-2,m-2)+x(n-1,m-1)+x(n,m) \\ viii & F(n-2,m-1)-2+x(n,m) \\ ix & F(n-1,m-2)-6 \\ x & F(n-2,m-2)-4+x(n-1,m-1) \\ xi & F(n-2,m-1)-6 \\ xii & F(n-2,m-1)-2+x(n-1,m) \\ xiii & F(n-2,m)-4 \end{cases} \quad (2)$$

Equation 2 is now written as the maximum of thirteen sub-equations. All the terms are independent of each other, therefore sub-equations can be computed in parallel. Since finding maximum is associative, then the best way to find is by making a binary tree from the result of thirteen sub-equation, which requires four levels as $\lceil \log_2 13 \rceil = 4$. Can we find *F(n,m)* better than this? Yes, if we look closely at Equation 2, unique *F(i, j)* terms are only *five*. If *F(i, j)* is present in more than one sub-equations, we can eliminate some sub-equations with out the loss of useful information based on the smallest

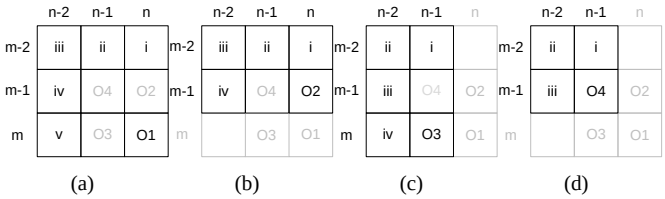

Figure 4. Matrices to show the elements from which *F(n,m)* is computed (a) O1 is calculated using i, ii, iii, iv and v as given in Equation 4 (b) O2 is calculated using i, ii, iii and iv as given in Equation 5 (c) O3 is calculated using i, ii, iii and iv as given in Equation 7 (d) O4 is calculated using i, ii and iii in Equation 6.

value of *x(i, j)*, which in this case is *-1*. For example, *F(n-1, m-2)* is present in equation *ii*, *iii*, *vi*, *ix*. so these sub-equations can be written as.

$$\begin{aligned} ii \quad & F(n-1,m-2)-2-1=F(n-1,m-2)-3 \\ iii \quad & F(n-1,m-2)-6 \\ vi \quad & F(n-1,m-2)-2-1=F(n-1,m-2)-3 \\ ix \quad & F(n-1,m-2)-6 \end{aligned} \quad (3)$$

So sub-equations *iii* and *ix* can be simply discarded as they can never be maximum. There is a tie between *ii* and *vi*, as we are not certain which may come as maximum. Using this reduction method for all the sub-equations, Equation 2 can be reduced to the following equation of seven sub-equations.

$$F(n,m)=\max\begin{cases} i & F(n,m-2)-4 \\ ii & F(n-1,m-2)-2+x(n,m-1) \\ iii & F(n-1,m-2)-2+x(n,m) \\ iv & F(n-2,m-2)+x(n-1,m-1)+x(n,m) \\ v & F(n-2,m-1)-2+x(n-1,m) \\ vi & F(n-2,m-1)-2+x(n,m) \\ vii & F(n-2,m)-4 \end{cases} \quad (4)$$

To find the maximum of seven sub-equations, we need $\lceil \log_2 7 \rceil = 3$ levels, which is better than 4 levels as before. Figure 4(a) show how *O1*(i.e. *F(n, m)* for a block of 3×3 elements) is calculated from Equation 4. But this is just one element in the 3×3 block of elements, we can find the rest of the unknowns *O2* (i.e. *F(n, m-1)* for a block of 3×3 elements), *O3* (i.e. *F(n-1, m)* for a block of 3×3 elements) and *O4* (i.e. *F(n-1, m-1)* for a block of 3×3 elements) using the similar methods for *O1*. The formulas for *O2*, *O3* and *O4* after applying partial Recursive Variable Expansion and elimination is given below.

$$F(n,m-1)=\max\begin{cases} i & F(n,m-2)-2 \\ ii & F(n-1,m-2)+x(n,m-1) \\ iii & F(n-2,m-2)-2+x(n-1,m-1) \\ iv & F(n-2,m-1)-4 \end{cases} \quad (5)$$

$$F(n-1,m-1)=\max\begin{cases} i & F(n-1,m-2)-2 \\ ii & F(n-2,m-2)+x(n-1,m-1) \\ iii & F(n-2,m-1)-2 \end{cases} \quad (6)$$

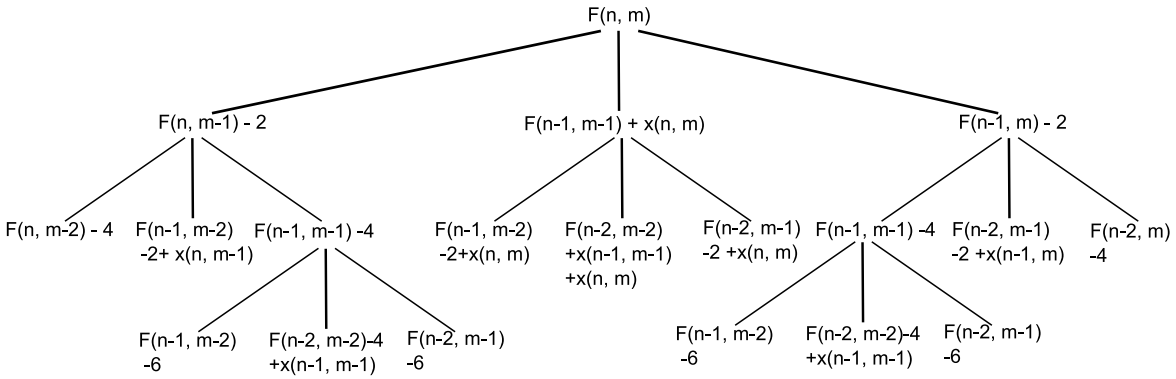

Figure 3. Recursion tree for Needleman-Wunsch Algorithm

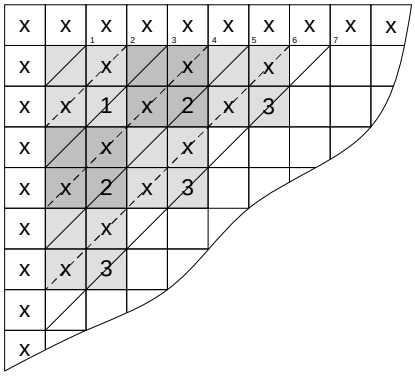

Figure 5. Sequence of fill, starting from the top left shaded square number 1 and moving down diagonally as showing by trailing numbers. All the squares with same number can be executed in parallel.

$$F(n-1,m)=\max\begin{cases} i & F(n-1,m-2)-4 \\ ii & F(n-2,m-2)-2+x(n-1,m-1) \\ iii & F(n-2,m-1)+x(n-1,m) \\ iv & F(n-2,m)-2 \end{cases} \quad (7)$$

The four unknowns (i.e. *F(n, m), F(n, m-1), F(n-1, m)* and *F(n-1, m-1)*) as shown in Figure 4 only depends on top and left elements of the 3×3 block, which we assume are already computed, therefore they can be computed in parallel. The sequence to fill a matrix in Needleman-Wunsch is shown in Figure 5. This sequence ensures that once a block is computed, all the terms in Equations 5, 5, 6 and 7 are already computed in the previous iteration.

A. Time Estimation

The four unknowns *F(n, m), F(n, m-1), F(n-1, m)* and *F(n-1, m-1)* in 3×3 block can be computed in parallel as there is no dependency among them, therefore the total time to compute all of them will be equal to the time taken by the unknown which has maximum number of sub-equations and which has maximum number of terms to be computed, which in this case is *F(n,m)*. A detailed time-estimation is given in Figure 6, which is the time to compute 3 blocks serially one after the other. If the matrix is filled to same extent with the prevalent parallel technique, then it will take seven diagonal lines as shown in Figure 5. The time estimate to compute seven lines serially one after the other is given in Figure 7, which is 28 levels. Therefore the speedup we got from our technique is $\frac{28}{18} = 1.55$.

IV. Conclusion

We have presented a new method to parallelize the Needleman-Wunsch algorithm which extracts more parallelism than the prevalent parallel techniques. By our technique, we are able to show that a speed up of 1.55 can be achieved, if a block of size 3×3 is used, this speed up can be increased if the RVE is applied further to achieve a bigger block size. Future work will involve the application of RVE on other Sequence alignment algorithm and with bigger block sizes.

References

[1] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol Biol.*, vol. 48, pp. 443–453, 1970.

[2] T. Smith and M. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, pp. 195–197, 1981.

[3] M. Y. Galperin, "The molecular biology database collection: 2007 update," *Nucleic Acids Research*, vol. 35, pp. D3–D4(1), January 2007.

[4] L. D. O. J. Benson DA, Karsch-Mizrachi I and W. DL, "Genbank," *Nucleic Acids Research*, vol. 35, pp. D21–5, January 2007.

[5] D. J. Lipman and W. R. Pearson, "Rapid and sensitive sequence comparison with fastp and fasta," *Methods Enzymol.*, vol. 183, pp. 63–98, 1990.

[6] W. M. E. W. M. S. F. Altschul, W. Gish and D. J. Lipman, "Basic local alignment search tool," *J. Mol. Biol.*, pp. 403–410, 1990.

[7] D. Brutlag, J. Dautricourt, R. Diaz, J. Fier, B. Moxon, and R. Stamm, "Blaze: an implementation of the smith-waterman sequence comparison algorithm on a massively parallel computer," 1993. [Online]. Available: citeseer.ist.psu.edu/brutlag93blaze.html

[8] Y. C. Hsien-Yu Liao, Meng-Lai Yin, "A parallel implementation of the smith-waterman algorithm for massive sequences searching," in *Proceedings of the 26th Annual International Conference of the IEEE EMBS San Francisco, CA, USA*, 2004.

[9] T. K. Yap, O. Frieder, and R. L. Martino, "Parallel computation in biological sequence analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 3, pp. 283–294, 1998. [Online]. Available: citeseer.ist.psu.edu/yap98parallel.html

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[11] T. M. E. M. P. K. B. S. V. Zubair Nawaz, Ozana Silvia Dragomir, "Recursive variable expansion: A loop transformation for reconfigurable systems," in *International Conference on Field-Programmable Technology 2007 (ICFPT'07) (to appear)*, 2007.

Time to compute four *x(i, j)* independently for different values of *i* and *j* in parallel	= 1 level
Time for three Add operations in parallel	= 2 level
Time to find max. of seven sub-equations	= 3 levels
Total time to compute one element	= 6 levels
Time to compute 3 blocks in diagonal serially	$= 6 \times 3 = 18$ levels

Figure 6. Time estimate for our technique

Time to compute *x(i, j)*	= 1 level
Time to Add or subtract in parallel	= 1 level
Time to find max. of three elements	= 2 levels
Total time to compute one element	= 4 levels
Time to compute 7 lines serially	$=4 \times 7 = 28$ levels

Figure 7. Time estimate for conventional parallel technique

[12] D. P. Lopresti, "P-nac: a systolic array for comparing nucleic acid sequences," *Computer*, vol. 20, no. 7, pp. 98–99, 1987.

[13] T. P. J. W. M. Chow, E. Hunkapiller, "Biological information signal processor," in *Proceedings of the International Conference on Application Specific Array Processors*, 1991.

[14] P. Guerdoux-Jamet and D. Lavenier, "Samba: hardware accelerator for biological sequence comparison," *Comput. Appl. Biosci.*, vol. 13, no. 6, pp. 609–615, 1997. [Online]. Available: http://bioinformatics.oxfordjournals.org/cgi/content/abstract/13/6/609

[15] R. H. S. I. M. Borah, M. Bajwa, "A simd solution to the sequence comparison problem on the mgap," in *Proceedings., International Conference on Application Specific Array Processors*, 1994.

[16] M. D. L. G. J. H. K. K. H. K. M. K. F. J. M.-M. D. P. E. R. A. S. D. S. R. H. Andrea Di Blas, David Dahle, "The kestrel parallel processor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16(1), pp. 80–92, 2005.

[17] B. Schmidt, H. Schröder, and M. Schimmler, "Massively parallel solutions for molecular sequence analysis," in *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 2002, p. 201.

[18] M. S. J. S. S. T. K. Chiang, J. Studniberg, "Hardware accelerator for genomic sequence alignment," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2006.

[19] Y. Yamaguchi, Y. Miyajima, T. Maruyama, and A. Konagaya, "High speed homology search using run-time reconfiguration," in *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2002, pp. 281–291.

[20] S. Margerm, "Reconfigurable computing in real-world applications," *FPGA and Structured ASIC Journal*, 2006.