OPTIMIZING PARALLEL SCAN SMITH WATERMAN ALGORITHM ON GPU

[1]HARSH SHUKLA, [2]MONIKA SHAH

[1, 2] Department of Computer Science, Institute of Technology, Nirma University, Ahmedabad, India
E-mail: harsh.g.shukla@gmail.com, monikag.shah@gmail.com

Abstract- Smith-Waterman is a well-known local sequence alignment algorithm that is used for finding regions of maximum similarity between two biological sequences and is known to be a highly compute intensive task. As it is based on dynamic programming it guarantees optimal results. But Dynamic Programming has its own drawbacks such as heavy memory consumption and significant amount of computations. Many academicians and researchers have tried variety of methods to harness the large amount of computational capabilities provided by the GPU in order to make this algorithm run faster. This paper proposes a version of Parallel Scan Smith-Waterman algorithm to improve performance of its phase-2. Here, we have also compared and evaluated performance of proposed work with other approaches like anti-diagonal and blocked anti-diagonal for both constant gap model and affine gap model and have observed remarkable performance gain.

Keywords- Smith-Waterman, GPU, Parallel Scan, CUDA

I. INTRODUCTION

Sequence alignment is one of the most fundamental operations carried out in bioinformatics. A sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity. A particular alignment has a similarity score associated with it, which gives us information about how good is that alignment. This similarity score is obtained by using an appropriate scoring scheme that is conformed to underlying biological models. Fig.1 shows an example of sequence alignment and its similarity score. Highest score represents maximum similarity. There are two kinds of sequence alignments: Global and Local. Global alignment forces the alignment to span across the entire length of query sequence, whereas local alignment identifies regions of similarity within long sequences that are widely divergent overall.

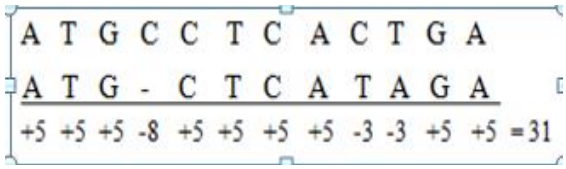

Figure 1 Sequence Alignment example and its score

One of the important issues in alignment is how the gaps are dealt. There are two kinds of gap penalty functions: (i) Constant Penalty (ii) Affine Gap Model. The constant gap penalty function treats each gap equally and assigns same penalty score to each gap. Affine gap model assigns different penalty scores to gap introducing position and consecutive gap positions.Keeping gaps together generates more significant results, in a biological perspective. Hence affine gap model is more preferred by biologists.

Smith-Waterman algorithm has a time complexity of O (mn) for sequences of size m and n. Since biological databases are huge in size, using Smith-Waterman algorithm is not a viable option for finding the alignment. Many efforts have been made in the direction of identifying an optimized solution for sequence alignment, which are time efficient but at the cost of accuracy. This paper considers time-efficiency without compromising accuracy like many other approaches.

The best option available to improve the time efficiency of Smith-Waterman algorithm is applying parallelism. In recent years, Graphical Processing Unit (GPU) cards have been introduced as a highly scalable parallel programming device. Easy availability and huge number of processors of GPU have attracted programmers to use it for performance improvement for data intensive computations. In 2007, NVIDIA released parallel programming model CUDA [9] with user-friendly and rich API to program GPU. This paper propose version of Parallel Scan [1], which is GPU-based parallel solution for Smith-Waterman algorithm, considering essential factors responsible for performance improvement on GPU [7]:

(i) Increase Byte: Flop ratio
(ii) Reduce Cache miss, as cache miss plays important role in over-all execution time
(iii) Synchronization free load distribution

The Smith-Waterman algorithm is discussed in Section II. Various methods that are used to parallelize the algorithm are explained in Section III.A detailed explanation of proposed work is presented in Section IV. Section V and Section VI consists of experimental results and conclusion respectively. And lastly future work is discussed in section VII.

II. SMITH-WATERMAN ALGORITHM

Smith-Waterman [3] is a very famous local alignment algorithm that has been in use since many years. In

order to incorporate the affine gap model, a modified version was proposed by Gotoh. The Smith-Waterman algorithm identifies highest scoring sequence alignment by computing a similarity matrix H as shown in Fig 3.

One of the sequences is placed on the top and the other sequence is placed on the left. The first row and first column are initialized to zero. The rest of the matrix is calculated using the formulae listed in Fig.2 based on defined score system. Fig.3 shows a similarity matrix for alignment computation between sequences "ATGCTCATAGA" and "ATGCCTCACTGA". Here we have applied a score system where, match=5, mismatch=-3, gap introduction penalty = -8, gap extension penalty =-1

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

$$E_{i,j} = \max \begin{cases} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{intro} \end{cases}$$

$$F_{i,j} = \max \begin{cases} F_{i-1,j} - G_{ext} \\ H_{i-1j} - G_{intro} \end{cases}$$

Figure 2 Formulae for calculating $H_{i,j}$

Where:

- G_{ext} is the gap extension penalty
- G_{intro} is the gap introduction penalty
- W(i,j) is the substitution matrix(match/mismatch)

	0	A	T	G	C	T	C	A	T	A	G	A
0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	5	0	0	0	0	0	5	0	5	0	5
T	0	0	10	2	1	5	0	0	10	2	2	0
G	0	0	2	15	7	6	5	4	3	7	7	0
C	0	0	1	7	20	12	11	10	9	8	7	6
C	0	0	0	6	12	17	17	9	8	7	6	5
T	0	0	5	5	11	17	14	14	14	6	5	4
C	0	0	0	4	10	9	22	14	13	12	11	10
A	0	5	0	3	9	8	14	27	19	18	17	16
C	0	0	2	2	8	7	13	19	24	16	15	14
T	0	0	5	1	7	13	12	18	24	21	15	14
G	0	0	0	10	6	5	11	17	16	21	26	18
A	0	5	0	2	7	4	10	16	15	21	18	31

Figure 3 Similarity Matrix between two sequences

The best alignment is obtained by finding a cell having the maximum value and then moving backwards depending on the direction used to construct the matrix.

The computation of similarity matrix cells is a compute intensive task and consumes most of the algorithm running time. Hence, this computation of matrix cells must be done in parallel in order to obtain faster results. Parallelizing data dependent computations is a highly challenging task. As shown in Fig.4, computation of each cell is dependent on the value of cells present on its left, top and diagonal.

	0	A	T	G	C	T	C	A	T	A	G	A
0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	5	0	0	0	0	0	5	0	5	0	5
T	0	0	10	2	1	5	0	0	10	2	2	0
G	0	0	2	15	7	6	5	4	3	7	7	0
C	0	0	1	7	20	12	11					
C												
T												
C												
A												
C												
T												
G												
A												

Figure 4 Data dependency present in the matrix

III. RELATED WORK

It has been observed by many researchers that the cells present in an anti-diagonal [8] are independent of each other as shown in Fig.5. Hence, executing one by one all the anti-diagonals, each one in parallel become a well-known solution for parallelizing Smith-Waterman like dynamic programming algorithm.

Issues in parallelizing using anti-diagonal approach are:

i) High amount of Cache-miss and Data Fetch operations for both row-major and column-major data organization.

ii)Complex computations for data reordering in anti-diagonal order.

	0	A	T	G	C	T	C	A	T	A	G	A
0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	5	0	0	0	0	0	5	0	5		
T	0	0	10	2	1	5	0	0	10			
G	0	0	2	15	7	6	5	4				
C	0	0	1	7	20	12	11					
C	0	0	0	6	12	17						
T	0	0	5	5	11							
C	0	0	0	4								
A	0	5	0									
C	0	0										
T	0											
G	0											
A	0											

Figure 5 Anti-diagonal Method

Sandes et al. Proposed a method known as blocked anti-diagonal in order to reduce the number of cache-misses that occurred in anti-diagonal method. Here, they divided the entire matrix into a number of blocks and these blocks were then executed in an anti-diagonal fashion. Anti-diagonal approach of a block reduced matrix reduces cache miss, but parallel execution of anti-diagonal blocks still causes higher fetch operations and smaller byte: Flop ratio.

Khajeh et al. Has recommended a new approach of parallelizing score computations for Smith-Waterman algorithm which reduces data fetch operations and cache miss at a higher level. Coalesced memory access also helps to reduce fetch operations, but the dependencies restrict coalesced access. Parallel Scan algorithm focuses on coalesced memory access and parallel access of row elements on row major data ordering to increase remarkable Byte: Flop ratio.

Parallel Scan algorithm applies 2 phases on each row and performs parallel calculations of elements in a row for each phase. The phase-1 ignores the influence of row-maximum ($E_{i,j}$).Therefore the value at any cell in the matrix is dependent only on the row above it and thus each cell in the current row can be calculated independently and in parallel. The phase-2 considers the previously ignored row-maximum. Khajeh et al. Uses a variation of parallel maximum scan to complete the phase 2.

IV. PROPOSED WORK

This paper proposes a version of phase-2 to reduce computations of Parallel Scan. Parallel Scan works in two stages up-shift and down-shift for parallel maximum scan computation. This paper proposes a version of phase-2, which reduces it to single stage for referring left-dependencies using parallel maximum scan.

Proposed phase-2 of parallel maximum scan is demonstrated in Fig.6 for computation of $E_{i,j}$. Like, Parallel Scan, it computes cells $E_{i,j}$ for j=1 to n-1 in parallel for i^{th} row. During first iteration 0, E_{ij} for each cell is initialized with gap introduction penalty (like 0 - g, 1- g, (n-1) - g) for j=1 to n-1. During (log n) subsequent iterations, it subtracts gap extension penalty ($e*2^{iteration-1}$) from a cell positioned at distance $2^{iteration-1}$ left from that cell for j= ($1+2^{iteration-1}$) to (n-1) and maximum between this subtracted value and the cell value is placed at $E_{i,j}$. For example, during iteration 1, e is subtracted from $E_{i,j}$ for j=1 to (n-1-1) and compared with cells 2 to n-1 respectively. During iteration 2, 2*e is subtracted from $E_{i,j}$ for j=1 to (n-1-2) and compared with cells 3 to n-1 respectively.

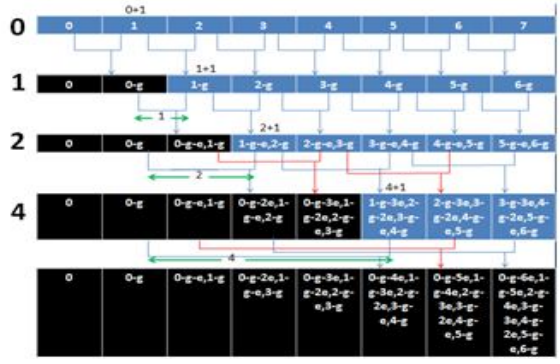

Figure 6 Single-stage Phase-2 of Parallel Scan Smith – Waterman algorithm

The proposed single stage phase-2 kernel of Parallel Scan is listed in Fig.7. Here, I specifies the iteration. d_E and d_H are E and H vectors present on the device (GPU). gap_intro represents the gap introduction penalty and gap_extend represents the gap extension penalty.

The proposed algorithm has several advantages over Parallel Scan Smith-Waterman algorithm.

(i) This proposed single stage phase-2 Parallel Scan takes runtime of O (logn), which is much optimized than phase-2 of Parallel Scan Smith-Waterman algorithm. In parallel maximum reduction each of the two stages in phase-2 requires O (logn) steps. Since the amount of work becomes half at each, the overall work is done in O (n).

(ii) Another time-inefficiency of parallel maximum reduction is in the case when the number of elements is more than the size of the single block. The entire array is divided into many blocks and partial modified scan results are used as input to the next recursive call. In this proposed algorithm, there is no need to divide the array and make recursive calls. The value of E can be calculated in one scan only as it works on global memory.

(iv) One more point is that the parallel maximum reduction algorithm works only on arrays having size of power of 2.So, if an array is not having a size of power of two it is required to convert it into power of two, whereas this proposed algorithm can work on data size which is not even a power of 2.

```
procedure PARALLEL_ALGORITHM()

for I=0 ; I < columns ; I=I*2
   if thread_id > I then
   j=thread_id
         if I=0 then
              d_E[j]=d_H[j-1] - gap_intro
              I=1/2;
         end if
         if I > 0 then
              temp=d_E[ j – I ] - ( I )*gap_extend
                   if temp > d_E [j] then
                   d_E[j]=temp
                   end if
          end if
   end if
     __syncthreads()
end for

 if d_E[j] > d_H[j] then
    d_H[j]=d_E[j]
 end if
```

Figure 7 Pseudo code for proposed single stage phase-2 of Parallel Scan [1]

V. EXPERIMENTAL RESULTS

We have tested our proposed algorithm on sequences of variable sizes and for both affined gap model and constant gap model. It is also compared with anti-diagonal and blocked anti-diagonal approaches. In blocked anti-diagonal method, we have not considered the optimizations mentioned in the Sandes et al.

A. Experiment Platform

We have performed our proposed algorithm on a system having Intel® Core™ i3 CPU 550 @ 3.2 GHz X 4 and NVIDIA GeForce GTX 480 graphic card.

B. Performance Comparison and Analysis

We have focused on reducing two-stage phase-2 to single stage phase-2 of Parallel Scan using parallel reduction.

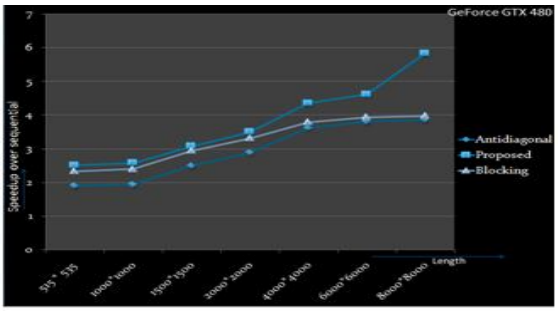

Figure 8 Speedup comparison for constant gap model

Initially, this algorithm was applied on constant gap model. Fig.8 shows speedup comparison chart between anti-diagonal, blocked anti-diagonal and the proposed Parallel Scan version for constant gap model. This chart shows that proposed algorithm has the highest speed up of 2-6 for constant gap model. It also represent that as size increases, speedup ratio also increases.

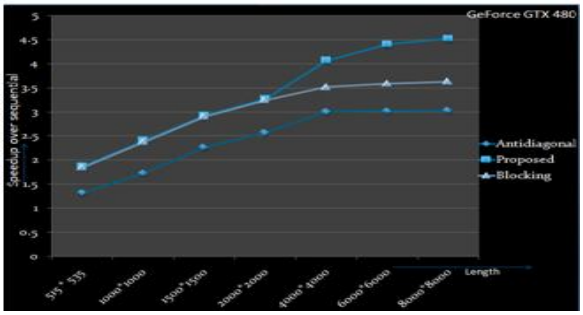

Figure 9 Speedup comparison for affine gap model

Affine gap model is highly preferred by biologists to find highest similarity and longer regions. Hence, we have also implemented proposed algorithm for affine gap model. Fig.9 shows speedup comparison chart between anti-diagonal, blocked anti-diagonal and the proposed Parallel Scan version for affine gap model. Here, it is observed higher speedup 1.5 – 4.5 using the proposed algorithm.

CONCLUSION

In this paper, we have discussed various approaches of parallelizing Smith-Waterman algorithm on GPU. We have shown that synchronization free load distribution, coalesced memory access on GPU reduces Fetch operations and improve overall performance. We have also demonstrated that optimization from multiple stage to single stage can help to improve overall performance. Our proposed algorithm gains 2-6 speedup over sequential among sequences with length varied from 512 to 8000 for constant gap model and 1.5 – 4.5 speedup for affine gap model.

FUTURE WORK

The implementation was done using global memory. Shared memory access can achieve higher performance gain as compared to global memory. Hence, we would like to apply shared memory access on proposed algorithm to reduce memory access time and improve overall performance.

Our work is limited to the extent that it can find the sequence alignment between two sequences only. We would like to extend it for database searches also.

REFERENCES

[1] Ali Khajeh-Saeed, Stephen Poole, J. Blair Perot, “Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors,” Journal of Computational Physics 229 (2010) 4247–4258.

[2] Sandes, E.F.O. and De Melo, A.C,”CUDA Align: using GPU to accelerate the comparison of megabase genomic sequences,” ACM SIGPLAN Notices. ACM (2010), pp. 137–146.

[3] T.F. Smith, M.S. Waterman, “Identification of common molecular subsequences,” J. Mol. Biol. 147 (1981) 195–197.

[4] O. Gotoh, “An improved algorithm for matching biological sequences,” J. Mol. Biol. 162 (1982) 705–708.

[5] S. Sengupta, M. Harris, Y. Zhang, J.D. Owens, “Scan primitives for GPU computing,” Graphics Hardware (2007) 97–106.

[6] G.E. Blelloch, “Prefix sums and their applications,” John H. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann, 1990.

[7] Shah M., Patel V.”An efficient sparse matrix multiplication for skewed matrix on gpu,” High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. (June 2012).

[8] P.Zhang, G.Tan, G.R.Gao,” Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform,” Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07 (A.C.M,NewYork,2007),pp. 39-48

[9] Nvidia CUDA [https://developer.nvidia.com/cuda-zone