An Adaptive Hybrid Multiprocessor Technique for Bioinformatics Sequence Alignment

Talal Bonny, M. Affan Zidan, and Khaled N. Salama

Abstract*— *Sequence alignment algorithms such as the Smith-Waterman algorithm are among the most important applications in the development of bioinformatics. Sequence alignment algorithms must process large amounts of data which may take a long time. Here, we introduce our Adaptive Hybrid Multiprocessor technique to accelerate the implementation of the Smith-Waterman algorithm. Our technique utilizes both the graphics processing unit (GPU) and the central processing unit (CPU). It adapts to the implementation according to the number of CPUs given as input by efficiently distributing the workload between the processing units. Using existing resources (GPU and CPU) in an efficient way is a novel approach. The peak performance achieved for the platforms GPU + CPU, GPU + 2CPUs, and GPU + 3CPUs is 10.4 GCUPS, 13.7 GCUPS, and 18.6 GCUPS, respectively (with the query length of 511 amino acid).

I. Introduction

DNA and protein database sequence alignment are among most important applications in bioinformatics. This application needs to process large amount of data. Heuristic algorithms, such as FASTA [2] and BLAST [3], are fast in finding approximate solutions. These algorithms have the problem of sensitivity since they trim the search and miss unexpected but important homologies. On the other hand, Smith-Waterman (SW) algorithm [4] is a well-known sequence alignment algorithms used in bioinformatics research. This algorithm is used to identify sequence alignments as well as sequence comparisons, sequence searches or even for plagiarism detection in text [5]. The Smith-Waterman algorithm guarantees the return of the optimal alignment of two sequences. The first one is called the *query* and the second one is called the *database*. It takes a long time to find the solution as the computing and memory requirements grow quadratically with the size of the database.
Previous attempts to accelerate the SW algorithm were implemented using the central processing unit (CPU) or the graphics processing unit (GPU). In the work reported here, we accelerate the SW algorithm by using our novel Adaptive Hybrid Multiprocessor technique, which uses both the GPU and the CPU. No attention has been paid in the past to exploit this opportunity, i.e. to use the existing resources (both the GPU and the CPU) in an efficient way. Furthermore, Our technique adapts the implementation of the SW algorithm depending on the number of CPUs given as the input to the technique. It distributes the workload (database) between the processing units (GPU and CPUs), such that each unit works on a portion of the database sequences according to its execution speed. This allows the resources available in the platform to be used efficiently and explicitly improves the execution performance.

Talal Bonny is with the Electrical Engineering Program, King Abdullah University of Science and Technology (KAUST), Thuwal, Kingdom of Saudi Arabia (phone: +966-598382799, e-mail: talal.bonny@kaust.edu.sa, web: http://sensors.kaust.edu.sa)

M. Affan Zidan is with the Electrical Engineering Program, King Abdullah University of Science and Technology (KAUST), Thuwal, Kingdom of Saudi Arabia (e-mail: mohammed.zidan@kaust.edu.sa)

Khaled N. Salama is with the Electrical Engineering Program, King Abdullah University of Science and Technology (KAUST), Thuwal, Kingdom of Saudi Arabia (e-mail: khaled.salama@kaust.edu.sa

II. The Smith-Waterman Algorithm

The Smith-Waterman algorithm [4] is a dynamic programming method for identifying similarity between nucleotide or protein sequences. It compares segments of all possible lengths between two sequences (query and database sequences) to identify the best local alignment. The algorithm starts by creating a similarity matrix, which computes scores for the comparisons of the two sequences. A score is based on the result of a comparison, which is either 'match' or 'mismatch'. The equations for computing the alignment scores are as follows:

$$E(i,j) = max \begin{cases} E(i,j-1) - G_{ext} \\ H(i,j-1) - G_{open} \end{cases} \tag{1}$$

$$F(i,j) = max \begin{cases} F(i-1,j) - G_{ext} \\ H(i-1,j) - G_{open} \end{cases} \tag{2}$$

$$H(i,j) = max \begin{cases} 0 \\ E(i,j) \\ F(i,j) \\ H(i-1,j-1) - W(q_i,d_j) \end{cases} \tag{3}$$

such that E(i,j) and F(i,j) are the maxima of the following two items: open a new gap and continue to extend an existing gap, respectively.
$W(q_i, d_j)$ stands for the score substitution matrix, which differs depending on the type of sequences required to find their alignment. In bioinformatics, there are two types of sequences, either DNA or protein. In our implementation, we do the alignment for protein sequences, and use two values in the score substitution matrix, $W(q_i, d_j)$, either +5 when q_i and d_i are matched, or -4 when q_i and d_i are mismatched. To open a new gap and extend an existing gap, we use the values -2 and -10, respectively.
From the previous formulas, we can find that the score of any cell H(i,j) in the matrix depends on the scores of the

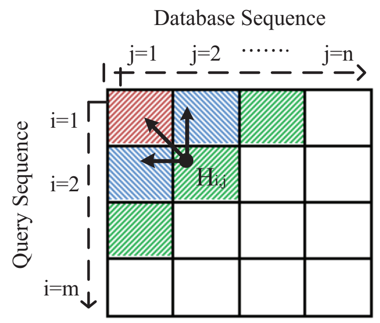

Fig. 1. Similarity Matrix. The cells located on the same anti-diagonal positions (have the same color) are computed together

three other elements (see Fig. 1): The left neighbor H(i,j-1), the up neighbor H(i-1,j), and the up-left neighbor H(i-1,j-1). That means that all cells located on the same anti-diagonal positions (have the same color) may be computed together using different threads because they are independent from each other. To measure the performance of the Smith-Waterman implementation, the Cell Updates per Second (CUPS) metric is commonly used [10], which represents the time required to complete the computation for one cell of the similarity matrix. The total number of cell updates gives the implementation performance of the sequence alignment algorithm:

$$\textit{Performance (CUPS)} = \frac{\textit{size(Query) x size(Database)}}{\textit{Time to complete the computation}} \tag{4}$$

Many previous attempts to accelerate the SW algorithm were implemented using CPU or GPU. Farrar [6] proposed a CPU-based SW implementation using Single Instructions Multiple Data (SIMD). The implementation delivered a peak performance of 3.0 GCUPS (giga cell update per second) on a 2.0 GHz Core 2 Duo processor. In [9], Farrar achieved a performance improvement of 11 GCUPS and 20 GCUPS by using four and eight CPU cores, respectively.
Munekawa et al. [8] accelerated the SW using the GeForce 8800 GTX card. They used the on-chip shared memory to reduce the amount of data being transferred between off-chip memory and the processing elements in the GPU. They achieved a peak performance of 5.65 GCUPS.
In [7], the authors used two GeForce 8800 GPU cards for CUDA-based implementation of the Smith-Waterman Algorithm. They compared their implementation with SSEARCH and BLAST running on a 3 GHz Intel Pentium IV processor and found that performed 2 to 30 times faster than other previous attempts on commodity hardware. A performance of 3.5 GCUPS was achieved using their implementation.
We accelerate the SW algorithm by using our novel Adaptive Hybrid Multiprocessor technique, which uses both the GPU and the CPU (as explained in the next section).

III. Our Adaptive Hybrid Multiprocessor Technique

The Smith-Waterman algorithm utilizes dynamic programming, which divides problem into sub-problems and solves the sub-problems separately. Computing the score of any cell can be considered as a sub-problem. Each sub-problem can be run on a different thread in parallel with another sub-problem running on another thread if there is no data dependency between them.
Previous implementations of the Smith-Waterman algorithm used platforms of homogeneous multiprocessor units (either CPUs or GPUs) to compute the score of the matrix cells. The available resources (CPU and GPU) in these platforms are not fully utilized in the implementation, although most users have ready access to PCs with modern graphics cards.

We introduce our Hybrid Multiprocessor Technique to implement the Smith-Waterman algorithm more efficiently. Our technique is based on using both the CPU and the GPU to improve the implementation performance.

In implementing the Smith-Waterman algorithm using heterogeneous multiprocessor units (GPU and CPU), two important issues need to be discussed: the type of workload (in terms of the length of the database sequence) and the amount of workload (in terms of the number of database sequences), which must be assigned to each processing unit (GPU and CPU). We have found that running the short sequences of the database on the GPU will not be as efficient [1] as running the long sequences on it. However, when long sequences of the database are run on the GPU and small ones run on the CPU simultaneously, the speed of the Smith-Waterman algorithm will increase explicitly. We sort the database sequences according to their length (from the longest to the shortest sequences). Then, our technique distributes the sequences between the GPU and CPU, such that the long sequences are sent to the GPU and the short sequences are sent to the CPU. The GPU and CPU will compute the similarity matrix for its sequence and will find the highest alignment score separately. The highest alignment score will be the final result for aligning the query sequence with the database sequences. To distribute the database between the GPU and CPU, we use 'Fixed Splitting' and 'Optimized Splitting' algorithms. The 'Fixed Splitting' algorithm distributes the database between the processing units equally. In the 'Optimized Splitting' algorithm, the number of database sequences assigned to the GPU and CPU is based on the speed of implementation for each one such that both of them will work in parallel and finish their tasks at the same time.

When the platform includes one GPU and a different number of CPUs, our technique adapts the work automatically based on the number of the CPUs given as input. Fig. 2 shows a flowchart of our database 'Optimized Splitting' algorithm. In the beginning, the algorithm finds the initial database splitter based on the number of CPUs (N) given as input. Then, it computes the ratio of the execution time (Tr) between the GPU and CPU. If this ratio is smaller than 1, then the splitting direction will be from the beginning of the database untill the end. Otherwise, it will be in the reverse direction. For any splitting direction, the algorithm splits the database between the GPU and the CPU parts based on the parameters 'Split'

[1]Efficiency means keeping the GPU cores as busy as possible in doing the computation.

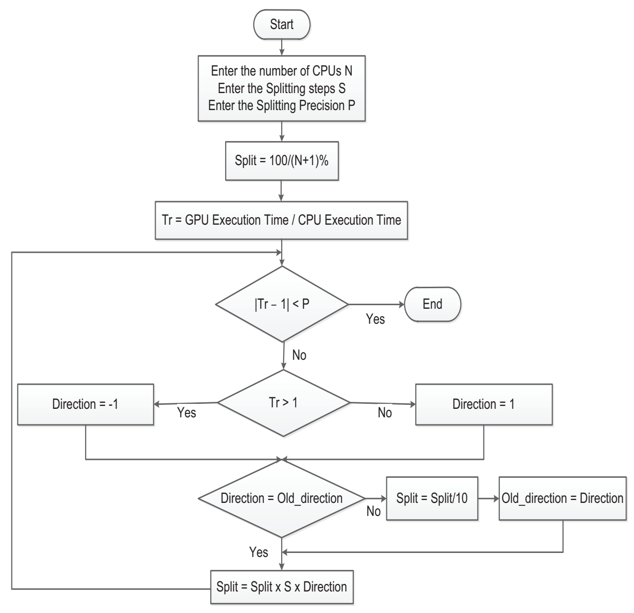
Start
Enter the number of CPUs N
Enter the Splitting steps S
Enter the Splitting Precision P
Split = 100/(N+1)%
Tr = GPU Execution Time / CPU Execution Time
|Tr – 1| < P
Yes
End
No
Direction = -1
Yes
Tr > 1
No
Direction = 1
Direction = Old_direction
No
Split = Split/10
Old_direction = Direction
Yes
Split = Split x S x Direction

Fig. 2. Flowchart of splitting the database automatically between the GPU and the CPUs based on the number of CPUs given in the input

and ‘S’ (splitting steps). The algorithm stops when it reaches the precision (P), which is defined as input parameter. In the end, the algorithm finds the optimal position for the database splitter between the GPU and the CPU parts. When more than one CPU is given in the input, the CPU part of the database will be distributed equally between all of them.

IV. Experimental Results

Our tests were conducted using an Intel Xeon X5550 CPU (2.676 GHz), and an nVidia Quadro FX 4800 GPU with 602 MHz core frequency and 76 GB/Sec memory bandwidth.
The ”SWISS-PROT” database (release 15.12, December 15, 2009) [1] was used to evaluate our adaptive technique. The query sequences were selected to cover different lengths (from 114 to 511 amino acid residues).
Our experimental results are presented in Figures 3 - 6 and Tables I and II.
The bar labeled ‘Average’ in any figure shows the average result for the full query sequence in that figure.

From the experimental results, we observe the following:
Figure 3 shows the performance results in giga cell update per second (GCUPS) for aligning each query sequence with the database using the implementation of Farrar [6] on a CPU, our implementation on a GPU, and our implementation on a hybrid GPU/CPU platform. The peak and average performance for all queries using our hybrid technique were 10.4 GCUPS and 8.5 GCUPS, respectively. Table I shows the Speed-up in the performance of our hybrid platform over Farrar’s implementation. Using the hybrid platform doubles the speed of the performance in comparison to Farrar’s implementation [6] which, was done on a CPU.

Our hybrid platform requires no additional costs to add the GPU or the CPU because they are commodity components. As noted above, most users have ready access to PCs with

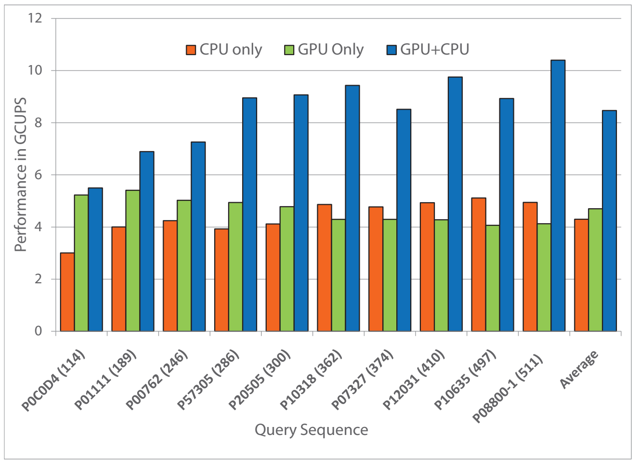

Fig. 3. performance results using CPU, GPU, and our hybrid GPU/CPU platform.

TABLE I
Speed-up in the performance of our Hybrid GPU/CPU over the implementation of Farrar

Sequence length	114	189	246	286	300
Speed-up	2.6x	2.4x	2.3x	2.5x	2.4x
Sequence length	362	374	410	497	511
Speed-up	1.9x	2.0x	1.9x	1.8x	2.1x

modern graphics cards. For these users, our implementation provides a zero-cost solution.
If the platform is equipped with a GPU and multiple CPUs, the workload may be distributed between them to exploit the available resources efficiently and to improve the execution performance. Fig. 4 shows the performance results for three types of hybrid platforms: ‘GPU + CPU’, ‘GPU + 2CPUs’, and ‘GPU + 3CPUs’, for different lengths of queries. The workload (database) is distributed between the GPU and the CPUs in each hybrid platform equally, i.e., using the ‘Fixed Splitting’ algorithm. For example, in the case of ‘GPU +3CPUs’ platform, each processor unit works on 25% of the database only. Fig. 4 shows (as expected) improved performance when the number of the CPUs is increased in the hybrid platform. The peak performance for the ‘GPU + CPU’, ‘GPU + 2CPUs’, and ‘GPU + 3CPUs’ platforms is 10.4 GCUPS, 13.7 GCUPS, and 18.6 GCUPS, respectively (which is achieved with the query length of 511 amino acid residues).
Our Adaptive Multiprocessor Technique distributes the workload between the processor units using the ‘Optimized Splitting’ algorithm (shown in Fig. 2). The workload is distributed between the GPU and the CPUs based on the number of CPUs given as input to the technique and based on the execution time of the processor units. Fig. 5 shows the percentage of the database assigned to the GPU for the three hybrid platforms. For example, in the ‘GPU + CPU’ platform and for the query length of 114 amino acid residues, 65% of the database is assigned to the GPU, while 35% is assigned to the CPU. But in the case of the ‘GPU + 3CPUs’ platform and for the same query length (i.e., 114), 40% of the database is assigned to the GPU, while 60% is assigned to the 3 CPUs (i.e., 20% of

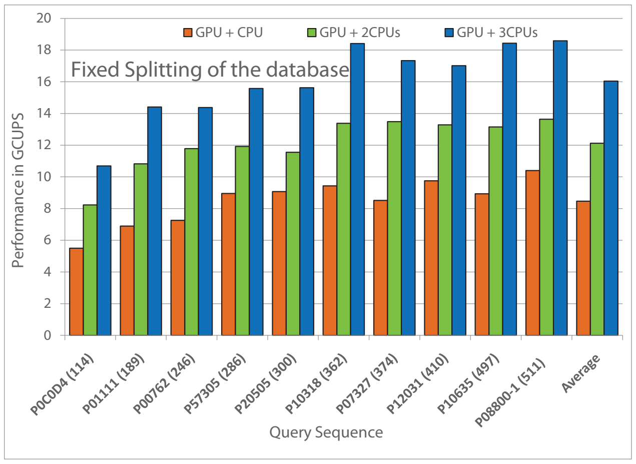

Fig. 4. Performance results for different types of hybrid platforms using the 'Fixed Splitting' Algorithm.

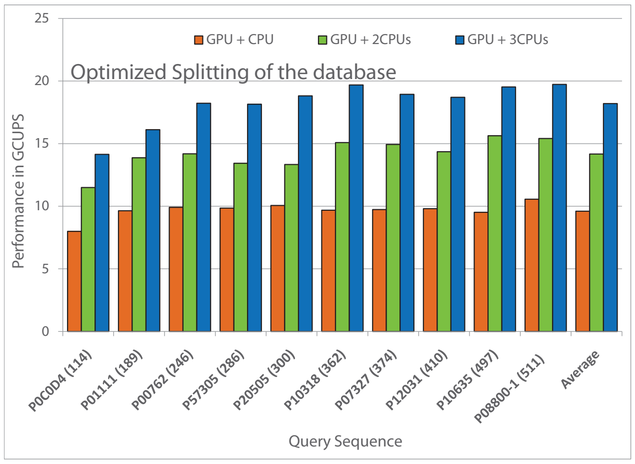

Fig. 6. Performance results for different hybrid platforms using the 'Optimized Splitting' Algorithm.

Percentage of Databse assigned for GPU (%)			
Query	**GPU + CPU**	**GPU + 2CPUs**	**GPU + 3CPUs**
P0C0D4 (114)	65.00	49.73	40.38
P01111 (189)	63.00	44.33	37.26
P00762 (246)	57.40	41.53	32.80
P57305 (286)	56.20	41.41	33.33
P20505 (300)	55.00	41.50	31.20
P10318 (362)	52.35	36.93	29.00
P07327 (374)	52.60	37.53	30.12
P12031 (410)	52.20	38.76	29.85
P10635 (497)	51.10	35.33	29.00
P08800 (511)	51.00	36.43	29.20

Fig. 5. Percentage of the database assigned to the GPU for the different hybrid platforms

the database is assigned to each CPU).
Using our 'Optimized Splitting' algorithm improves the performance results explicitly in comparison to the performance achieved by using the 'Fixed Splitting' algorithm. Fig. 6 shows the performance results after distributing the workload between the processor units and is based on the number of CPUs given as input to the technique. The peak performance is improved to 10.6 GCUPS, 15.5 GCUPS, and 19.8 GCUPS for the platforms 'GPU + CPU', 'GPU + 2CPUs', and 'GPU + 3CPUs', respectively.

The performance improvement from using the 'Optimized Splitting' algorithm instead of the 'Fixed Splitting' algorithm for the platform 'GPU + 2CPUs' is shown in Table II. In this table, the maximum improvement achieved in the performance is 40% (for the query sequence of 114 amino acid residues).

TABLE II

PERFORMANCE IMPROVEMENT OF THE 'OPTIMIZED SPLITTING' OVER THE 'FIXED SPLITTING' FOR THE PLATFORM 'GPU + 2CPUS'

Sequence length	114	189	246	286	300
Performance Improvement	40%	28%	20%	13%	15%
Sequence length	362	374	410	497	511
Performance Improvement	13%	11%	8%	19%	13%

V. Conclusion

We have presented a novel adaptive hybrid multiprocessor technique based on the GPU and CPU to Speed-up the Smith-Waterman sequence alignment algorithm. Our technique exploits the resources available in the platform efficiently to accelerate the implementation of the algorithm. In addition, our Speed-up technique using a hybrid GPU/CPU is not limited to the Smith-Waterman algorithm. It can be used generally with any sequence alignment algorithm. When our technique is used, a peak performance of 10.4 GCUPS is achieved, which is 2.6x faster than Farrar's implementation [6]. The code is available in our website:
http://sensors.kaust.edu.sa

References

[1] The UniProt/Swiss-Prot Database, http://www.ebi.ac.uk/swissprot/
[2] European Bioinformatics Institute Home Page, FASTA searching program, 2003. http://www.ebi.ac.uk/fasta33/.
[3] National Center for Biotechnology Information. NCBI BLAST home page, 2003. http://www.ncbi.nlm.nih.gov/blast.
[4] T. F. Smith and M. S. Watermann. Identification of common molecular subsequence. Journal of Molecular Biology, 147:196-197, 1981.
[5] Z. Su, B.-R. Ahn, K.-Y. Eom, M.-K. Kang, J.-P. Kim, and M.-K. Kim, Plagiarism detection using the Levenshtein distance and Smith-Waterman algorithm, in ICICIC 08, 2008, p. 569.
[6] M. Farrar, Striped Smith-Waterman speeds database searches six times over other SIMD implementations, Bioinformatics, vol. 23, no. 2, pp. 156-161, Jan. 2007.
[7] S. A. Manavski and G. Valle, CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment, BMC Bioinformatics, vol. 9, no. S10, Mar. 2008, 9 pages.
[8] Y. Munekawa, F. Ino, and K. Hagihara. Design and Implementation of the Smith-Waterman Algorithm on the CUDA-Compatible GPU. 8th IEEE International Conference on BioInformatics and BioEngineering, pages 1-6, Oct. 2008.
[9] Michael S. Farrar. "Optimizing smith-waterman for the cell broadband engine". http://sites.google.com/site/farrarmichael/smith-watermanfortheibmcellbe
[10] Lukasz Ligowski and Witold Rudnicki. An efficient implementation of Smith-Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In IEEE International Workshop on High Performance Computational Biology (HiCOMB 2009), 2009.