High Performance Technique for Database Applications Using a Hybrid GPU/CPU Platform

M. Affan Zidan, Talal Bonny, and Khaled N. Salama
Electrical Engineering Program
King Abdullah University of Science and Technology (KAUST)
Thuwal, Kingdom of Saudi Arabia
{mohammed.zidan, talal.bonny, khaled.salama}@kaust.edu.sa

ABSTRACT

Many database applications, such as sequence comparing, sequence searching, and sequence matching, etc, process large database sequences. we introduce a novel and efficient technique to improve the performance of database applications by using a Hybrid GPU/CPU platform. In particular, our technique solves the problem of the low efficiency resulting from running short-length sequences in a database on a GPU. To verify our technique, we applied it to the widely used Smith-Waterman algorithm. The experimental results show that our Hybrid GPU/CPU technique improves the average performance by a factor of 2.2, and improves the peak performance by a factor of 2.8 when compared to earlier implementations.

Categories and Subject Descriptors

C.1 [**Computer Systems Organization**]: Processor Architectures

General Terms

Performance, Design

Keywords

Performance, Hybrid Platforms, Sequence Alignment, GPU

1. INTRODUCTION

Sequence database applications, such as sequence comparing, sequence searching and sequence matching, etc., are used in different research areas for different purposes such as Video, audio, or image Copy Detection [2], text plagiarism detection [3], DNA sequence matching detection [4], etc. . These applications are considered to be high consumers of computation time because they are based on searching or comparing a given pattern (sequence) with large numbers of pattern (sequences) stored in a database to find matching or near matching between them. These searches process large amounts of data. For instance, protein database [5] may reach more than 11 GB in size. Other domains that produce sequential data have also reached similar data scales [10]. As these applications may take hours of mainframe time to get the optimum solution, an efficient techniques and powerful platforms are required to produce solutions in a reasonable time. Researchers have utilized various techniques and general-purpose platforms to improve the execution performance of sequence database applications. Their work on these platforms is based either on a CPU with many cores or on a Graphical Processing Unit (GPU). GPUs are increasingly being deployed for applications reserved for CPUs. For example, China Nebulae supercomputer claims a high spot on the top 500 supercomputer list while relying on NVIDIA Tesla C2050 GPUs. Competition between CPUs and GPUs is likely to be intense, and hybrid CPU/GPU systems are on the near-term horizon. For example, Intel developed processor micro-architecture, called Sandy Bridge, to be launched on January, 2011. The new processor contains a GPU and multi-core CPUs on one chip [15]. Another example for a future next generation microprocessor combining a GPU with CPUs in a single package is AMD Fusion [16]. The new microprocessor generation era will be suitable to implement our first hybrid GPU/CPU technique to improve the performance of sequence database applications.

Usually, a database contains different sequence lengths. We have found that running the short sequences of the database is not as efficient [1] as running the long sequences on it (details presented later in this paper). However, when long sequences of the database are running on a GPU and the short ones on a CPU simultaneously, the speed of the database application will be efficiently increased. To verify our technique, we apply it to the Smith-Waterman algorithm, a well-known sequence alignment algorithm [6]. The Smith-Waterman (SW) algorithm guarantees to return the optimal alignment of two sequences. The first one is called the *Query* and the second one is called the *Database*. On the other hand, it requires a long time to return the solution as its computing and memory requirements grow quadratically with the size of the database.

We explicitly improve the performance of the Smith-Waterman algorithm by using a novel implementation technique based on GPU and CPU. This new technique distributes the data-base sequences between the GPU and CPU such that the GPU cores are efficiently engaged by keeping them busy throughout the entire computing cycle. Interestingly, our *Hybrid GPU/CPU* platform requires no additional costs to add a GPU or a CPU as they are commodity components. In particular, most users have easy access to PCs with modern graphics cards. For these users, our implementation provides a zero-cost solution.

Our novel contributions are as follows:

1) The GPU cores are efficiently exploited during the entire processing time by running the long sequences on it and the short ones on the CPU.

2) The performance of any previous implementation attempt

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

to speed up the sequence database applications can be further improved if our *hybrid GPU/CPU* technique is used in conjunction with the original implementation. No attention has been paid in the past to exploit this opportunity of combining a CPU with a GPU, i.e., to use existing resources in an efficient way.

3) The platform that is used to implement our speed-up technique, *Hybrid GPU/CPU*, requires no additional cost to add a GPU or a CPU as they are commodity components. In particular, most users have access to PCs with modern graphics cards. For these users, our implementation provides a zero-cost solution. In addition to that, the new microprocessor generation from Intel [15] or AMD [16] will be suitable to implement our technique.

4) When we apply our technique to the implementation of the Smith-Waterman algorithm, which was done by Farrar [7, 8], there is an improvement in the average performance by factor of 2.2 and a peak performance improvement by factor of 2.8 can be achieved.

The rest of this paper is organized as follows. In Section II, we introduce the Smith-Waterman (SW) algorithm and its traditional implementation. Section III describes the challenges of implementing the Smith-Waterman algorithm on a GPU. Section IV presents our novel technique for implementing the SW algorithm in detail using the Hybrid GPU/CPU platform. Experimental results are presented in Section V. We conclude in Section VI.

2. THE SMITH-WATERMAN ALGORITHM

The Smith-Waterman algorithm [6] is a dynamic programming method for determining similarities between nucleotide or protein sequences. This algorithm is based on the idea of comparing segments of all possible lengths between two sequences to identify the best local alignment. The SW algorithm begins by computing a similarity matrix score. This matrix has two dimensions, one for the query sequence and the other for the database sequence.

If the query sequence and the database sequence have m and n amino acid residues, respectively, then the number of the similarity matrix cells will be "m X n". The algorithm assigns a score to each cell comparison between the two sequences. The score is based on the result of the comparison, which is either "match", or "mismatch".

If the sequences are mismatched, then one of three operations may be done: insertion, deletion, or substitution.

Each of these operations has a previously defined score.

The equations for computing the alignment score, H(i,j), are as follows:

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

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

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

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

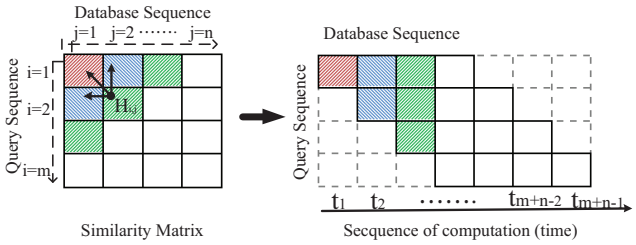

Figure 1: Computation sequence of the similarity matrix. the score of the cells which have the same color are computed together

substitution matrix, which differs depending on the type of sequences required to find their alignment. From the previous formulas, we can determine that the score of any cell, H(i,j), in the matrix depends on the scores of the 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). This means that the score of any cell can not be computed before computing the scores of cells located on the anti-diagonal positions. Fig. 1 (left side) shows the sequence of the cell computation in the similarity matrix. All the cells located on the same anti-diagonal positions (have the same color) are computed together using different threads because they are independent of each other.

To measure the performance of the Smith-Waterman implementation, the Cell Updates per Second (CUPS) metric is commonly used [17], 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:

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

Many previous attempts to accelerate the SW algorithm were implemented using a CPU or a GPU. Farrar [7] proposed a CPU-based SW implementation using Single Instructions Multiple Data (SIMD). The implementation delivered a peak performance of 3.0 GCUPS (giga cell updates 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. [11] accelerated the SW using a 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 [12], 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 their implementation performed 2 to 30 times faster than other previous attempts on commodity hardware. A performance of 3.5 GCUPS was achieved using their implementation.

Schatz et al. [13] also presented a CUDA-based implementation using a GPU in a common workstation. They achieved a 3.5-fold speedup over a CPU implementation.

In the next section, we show how we accelerate the SW algo-

rithm by using our novel hybrid technique, that utilizes both the GPU and the CPU (as explained in the next section).

3. CHALLENGES OF IMPLEMENTING THE SMITH-WATERMAN ALGORITHM ON A GPU

The Smith-Waterman algorithm needs to build and compute a similarity matrix of the number of times equal to the number of sequences in the database. When the "Similarity Matrix" is computed using a GPU, not all the GPU cores will be used to compute the matrix at the beginning and the end of the computation process (see Figure 1-right side). In other words, in the computation of each similarity matrix, there are always two triangles of cells (one at the beginning of the matrix and one at the end) in which the GPU cores are idle [2].

For example, in Fig. 1 (right side):

- At moment 't_1', only one GPU core is involved in computing the score of one cell and 'm-1' cores, which are responsible for the computation of the remaining cells in the first column, are idle.
- At moment 't_2', two GPU cores are busy computing the score of two cells and "m-2" cores, which are responsible for the computation of the remaining cells in the second column, are idle.

And so on. The same scenario happens at the end of the matrix computation. Clearly, when the sequence of the database is long, the two triangles of the similarity matrix (which are gray colored) can be ignored in comparison with the size of the matrix. But, when the sequence of the database is short, the two triangles can not be ignored because they will negatively affect the performance of the matrix computation as measured in CUPS.

Fig. 2 shows an example of the computation sequence of the similarity matrix when the length of the database sequence is changed and the length of the query is fixed.

Considering that the length of the query is 'm' and the length of the database sequence is 'n', then the efficiency, 'E', of using the GPU cores of any multiprocessor to compute the similarity matrix is computed as follows:

$$Efficiency = \frac{\sum_{t=1}^{t=m+n-1} Used\_Cores(t)}{\sum_{t=1}^{t=m+n-1} Available\_Cores(t)} \quad (5)$$

such that,

Used_Cores(t): are the cores used for computing the similarity matrix at the moment (t). In Fig. 2, the used cores are white colored, and their sum is equal to $(m \times n)$.

Available_Cores(t): are the cores used in the computation plus the idle cores (which are not used in the computation) at the moment (t). In Fig. 2, the available cores are white and gray colored, and their sum is equal to $(n + m - 1)$.

By substituting in Equation 5:

$$E = \frac{m \times n}{m \times (n + m - 1)} = \frac{n}{(n + m - 1)} \times 100\% \quad (6)$$

Such that E→[0%,100%]

If 'm' is fixed, the GPU cores efficiency, 'E', may have different values based on the value of 'n':

If n → 0 ⇒ E → 0
If n = m - 1 ⇒ E = 50%
If n → ∞ ⇒ E → 100%

In the first formula, if the length of the database sequence is very small, then the efficiency of the GPU cores that are used to compute the similarity matrix will be very low. This is because the two triangles of the cells (one at the beginning of the matrix and one at the end) in which the GPU cores are idle, will be very big. Fig. 2.a shows an example for the computation sequence when the database sequence length 'n' is less than the query length 'm'. In this figure, n = 1 and m = 4. For that, the efficiency will be E = 25%.

In the second formula, if the length of the database sequence is one residue less than the length of the query, then the efficiency will be improved by 50%, which is not enough. Fig. 2.b shows an example for this case when n = 3 and m = 4. This will result in an efficiency of E = 50%.

In the third formula, the length of the database sequence is longer than the length of the query. In this case, the efficiency will be very high because the two triangles of the matrix cells will be small and almost all the GPU cores will be involved in computing the similarity matrix cells during the whole computation time. Fig. 2.c shows an example for this case when n = 14 and m = 4. The efficiency in this case will be 82%. Fig. 3 shows how the efficiency is increased when the database sequence length is increased for different query sequence lengths (for brevity, only five different query sequence lengths are shown in this figure). The efficiency starts from 1% when the length of the database sequence is very short, and it increases to 99% when the length of the database sequence is very long. However, when the database sequence length is fixed, the efficiency and the query sequence length are inversely proportional (see Eq. 6). For that, the efficiency when the query length m = 114 (in Fig. 3) is higher than the efficiency when the query length is m = 511 amino acid residues.

The 'SWISS-PROT' database has a lot of short length sequences in comparison to the length of the query. If the GPU cores are used to compute the similarity matrix of these sequences, the efficiency will be very low and the execution performance will be negatively affected.

4. OUR HYBRID GPU/CPU TECHNIQUE

To solve the problem of the low efficiency resulting from

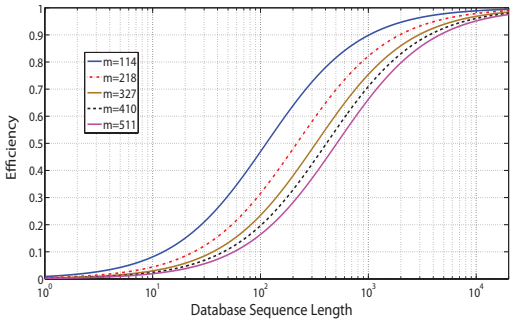

Figure 3: The Efficiency of the GPU cores for different query sequences. When the database sequence length is increased, the Efficiency will be improved

[2]Idle core in our case means that the core is not involved in computing the similarity matrix. It may be busy with coherence control

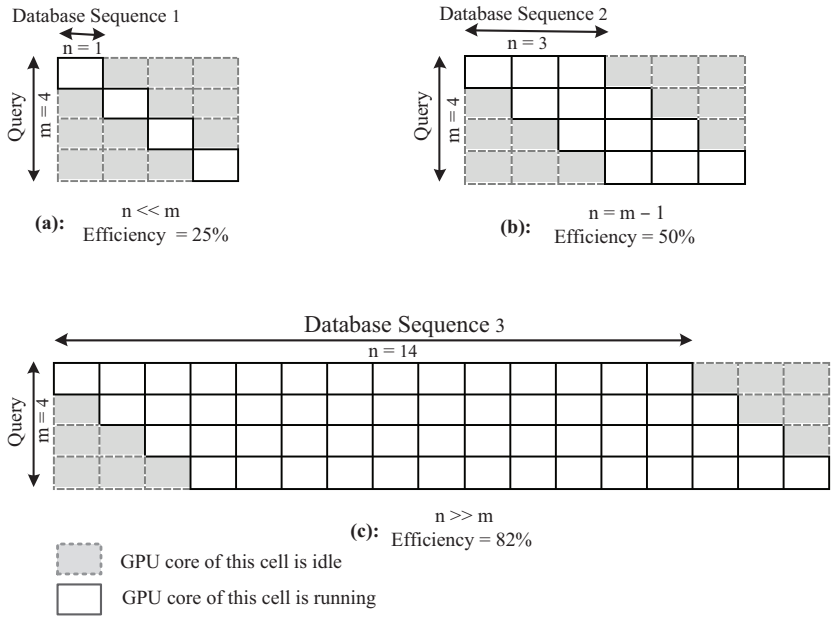

Figure 2: Example shows the Efficiency of the GPU cores which are involved in the computation of the similarity matrix cells. (a): the database sequence length 'n' is less than the query length 'm'. The efficiency is very low. (b): database sequence 2 is now longer and the Efficiency is improved. (c) 'n' is much longer than 'm' and the Efficiency is high

running the short length sequences of the database on a GPU (as explained in Sec 3), we introduce our novel solution which is called the *Hybrid GPU/CPU Technique*. In this technique, both the GPU and the CPU are used to compute the similarity matrix of the database sequences such that the similarity matrix of the long database sequences is computed using the GPU cores, while the CPU is used to compute the similarity matrix of the short sequences. Both GPU and CPU are run simultaneously and finish their tasks together.

If sequence $\rightarrow$ short $\Rightarrow$ suitable for CPU
If sequence $\rightarrow$ long $\Rightarrow$ suitable for GPU

To implement this technique, we sort the database sequences according to their length from the longest to the shortest sequences. This sorting is done off-line and only one time. After sorting the database sequences, we distribute the sequences between the GPU and the CPU, such that the long sequences are sent to the GPU and the short ones to the CPU. The query sequence is also sent to the GPU and the CPU as well (see Figure 4). The GPU and CPU compute the similarity matrices for their respective sequences and determine the highest alignment scores separately. The highest alignment score is the final result for aligning the query sequence with the database sequences. An off-line splitting technique is done to find the optimal splitting ratio for the database. Algorithm 1 shows the pseudo code for our splitting technique. The algorithm starts with two splitting points, one at the 0% ratio of the database "split 1" and the other one "split 2" at the 100% ratio (lines 1 and 2, respectively). After each splitting, the performance in CUPS is computed, 'P1' after split 1 and 'P2' after split 2 (lines 3 and 4, respectively). If the difference between the two splits is less than the required precision, then a new split "split 3" is done at the middle of the previous two splits (line 6). The performance 'P3' in CUPS is computed at the new split (line 7). If this performance is better than the performance done by "split 1" (i.e., 'P1'), then "split 1" will be at the new place (line 9). Otherwise, "split 2" will have the new place, which is at "split 3" (line 12). The performance in CUPS is computed each time until the difference between the two splits ("split 1" and "split 2") reaches the required precision. For faster splitting of the database sequences, we select the initial splitting point to be based on the performance ratios of the GPU and the CPU. This may be done by assigning the entire database sequences to the GPU and computing the performance (in CUPS). Then assigning the entire database sequences to the CPU and computing the

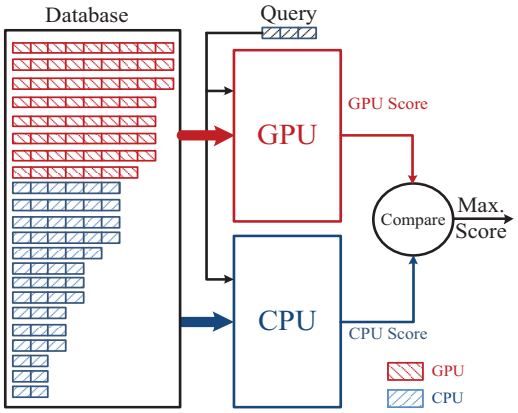

Figure 4: Our Technique: Hybrid GPU/CPU. The database sequences are sorted according to their length and then distributed to the GPU (long sequences) and CPU (short sequences)

new performance. The ratio of these performances implies the initial ratio of splitting the database sequences between the GPU and the CPU. This splitting ratio may be used later with any new query for aligning it with the same database. Farrar [7] delivered very high performance (3.0 GCUPS) in implementing the SW algorithm using a 2.0 GHz Core 2 Duo processor. Therefore, we selected his implementation to run the short sequences of the database on the CPU. Using our implementation of the SW algorithm on the GPU in conjunction with the implementation of Farrar on the CPU (the Hybrid GPU/CPU technique) improves the performance significantly (as presented in Section 5).

Algorithm 1 Our Splitting Algorithm

```
1: Split1 ← 0%                          ▷ Initial splitting point 1
2: Split2 ← 100%                        ▷ Initial splitting point 2
3: P1 = CUPS (Split1)                   ▷ Performance at Split 1
4: P2 = CUPS (Split2)                   ▷ Performance at Split 2
5: while (Split1 − Split2) < Required Precision do
6:     Split3 ← (Split2 + Split1) /2
7:     P3 = CUPS (Split3)
8:     if CUPS (Split3) > P1 then
9:         Split1 ← (Split3)
10:        P1 ← P3
11:    else
12:        Split2 ← (Split3)
13:        P2 ← P3
14:    end if
15: end while
```

5. EXPERIMENTAL RESULTS

In this section, we present the performance results of our Hybrid GPU/CPU technique. Our implementation was run on 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. We used a Microsoft Visual Studio 2008 compiler to compile the C++ part of the program. The 'optimize-for-speed' option is set during the compilation. All the other options for the compiler and linker were set at the default settings. However, the OpenCL compiler was used to compile the OpenCL code.

The evaluations were conducted using the well-known protein sequences from the "SWISS-PROT" database (release 15.12, December 15, 2009) [14]. The query sequences were selected to cover a large set of different lengths. Our code is available on our website [1].

Fig. 5 shows the performance of using the GPU and the CPU for different query sequence lengths considering two cases:

1) when the sequences of the database are distributed between the GPU and the CPU correctly, i.e., the long sequences are assigned to the GPU and the shorter ones to the CPU (second bar).

2) when the sequences of the database are distributed between the GPU and the CPU in a reverse fashion, i.e., the short sequences are assigned to the GPU and the longer ones to the CPU (first bar). Fig. 5 shows that improvement in the total performance is 1.35 times greater when the database sequences are assigned correctly. For example, at the query length 374, the performance when the database is assigned in a reverse fashion is 7.15 GCUPS, but when the

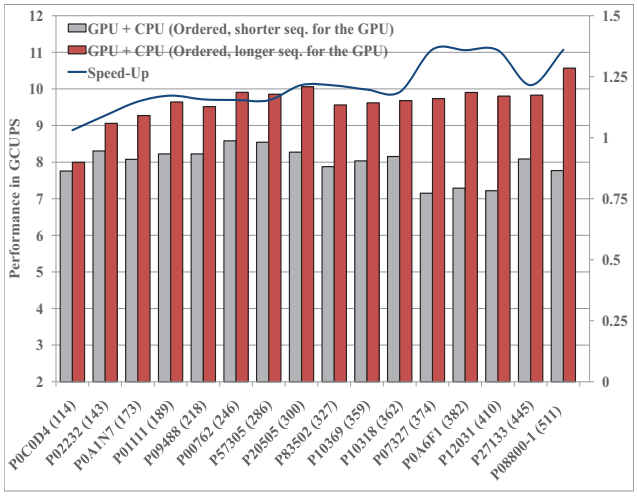

Figure 5: The total performance (GPU + CPU) when the database sequences are assigned correctly, i.e long sequences to the GPU and sort one to the CPU (second bar). And when the database sequences are assigned in a reverse fashion (first bar)

database is correctly assigned, the performance is improved to 9.7 GCUPS.

Fig. 6 shows the performance when only the CPU is used for the alignment (first bar). This performance is obtained by utilizing Farrar's implementation of the SW algorithm [7] on our CPU (Intel Xeon X5550 CPU with 2.676 GHz). The performance using only the CPU ranges from 3 GCUPS (for a query length of 114 amino acid residues) to 4.9 GCUPS (for a query length of 511 amino acid residues). The average performance for all queries is 4.2 GCUPS.

The second bar shows the performance when our implementation of the SW algorithm is run only with the GPU. The

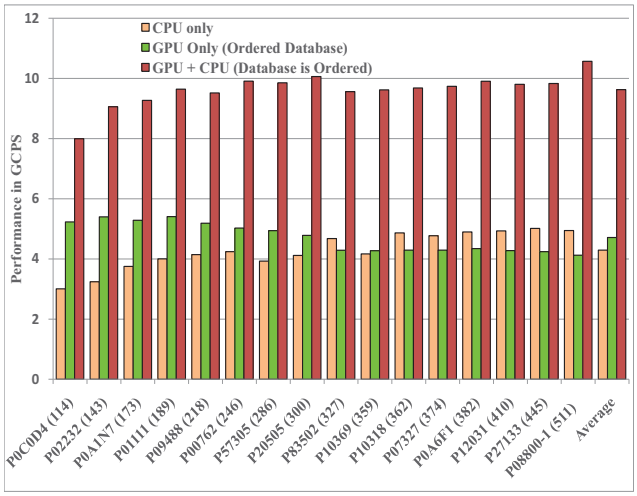

Figure 6: The performance of implementing the Smith-Waterman algorithm using: only CPU (first bar), only GPU (second bar), and our implementation using our Hybrid GPU/CPU technique (third-bar)

performance ranges from 5.2 GCUPS (for a query length of 114 amino acid residues) to 4.1 GCUPS (for a query length of 511 amino acid residues). The average performance for all queries using only the GPU is 4.7 GCUPS.
The third bar shows the performance of implementing the SW algorithm using our Hybrid GPU/CPU technique. This bar shows an improvement in the performance of between 7.9 and 10.5 GCUPS. The average performance for all queries using our Hybrid GPU/CPU technique is 9.6 GCUPS.
Fig. 7 shows the performance improvement of our Hybrid GPU/CPU implementation over Farrar's implementation [7]. This figure shows an average performance improvement by a factor of 2.2, and a peak performance improvement by a factor of 2.8 at the query length of 143 amino acid residues.

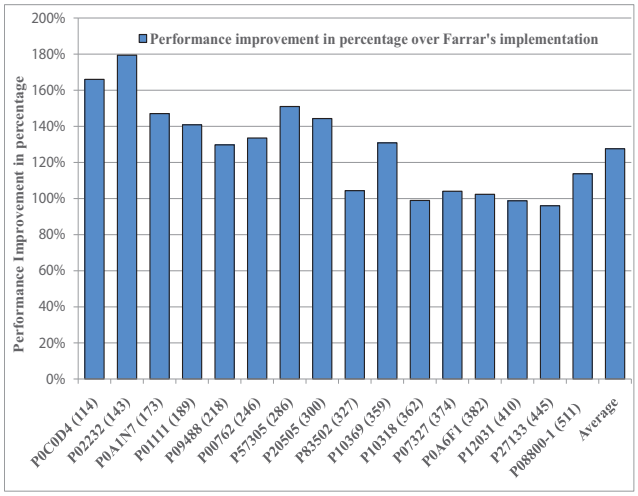

Figure 7: The performance improvement in percentage when our Hybrid GPU/CPU is used in comparison to Farrar's implementation. The peak performance improvement is 2.8x

6. CONCLUSION

We have presented the first hybrid technique based on both a GPU and a CPU to improve the performance of sequence database applications. Our technique efficiently exploits the GPU cores during the entire processing time by running the long sequences on the GPU and the short ones on the CPU. As a case study, we selected the Smith-Waterman algorithm, which was implemented by Farrar [7, 8], to measure the performance of our technique. If the platform contains multiple CPUs then the database can be distributed between all the cores to provide better efficiency as we did in [18].

7. REFERENCES

[1] http://ee.kaust.edu.sa/smith_waterman.html

[2] A. Hampapur and Ki-Ho Hyun and R. Bolle. Comparison of Sequence Matching Techniques for Video Copy Detection. In Proceedings of the International Conference on Storage and Retrieval for Media Databases, pp. 194-201, 2002.

[3] Sven Meyer zu Eissen and Benno Stein, Intrinsic Plagiarism Detection. Lecture Notes in Computer Science, Volume 3936, pp. 565-569, 2006.

[4] Khan, F.A. and Aurangzeb and Khan, Z.A. DNA sequence matching system based on hardware accelerators utilized efficiently in a multithreaded environment. International Conference on Electrical Engineering, pp. 1-6, 2009.

[5] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung, BioBench: A Benchmark Suite of Bioinformatics Applications, In Proceedings of the International Symposium on Performance Analysis of Software and Systems, pp. 2-9. 2005.

[6] T. F. Smith and M. S. Watermann. Identifcation of common molecular subsequence. Journal of Molecular Biology, 147:196-197, 1981.

[7] M. Farrar, Striped Smith-Waterman speeds database searches six times over other SIMD implementations, in Bioinformatics, vol. 23, no. 2, pp. 156-161, Jan. 2007.

[8] Michael S. Farrar. "Patent application title: OPTIMIZED SMITH-WATERMAN SEARCH". IPC8 Class: AG06F1730FI, USPC Class: 707, September 2009.

[9] Michael S. Farrar. "Optimizing smith-waterman for the cell broadband engine". http://sites.google.com/site/farrarmichael/smith-watermanfortheibmcellbe

[10] D. Yankov, E. Keogh, and U. Rebbapragada. Disk-aware discord discovery: Finding unusual time series in tera-byte sized datasets . In Proceedings of the IEEE International Conference on Data Mining, pp. 381-390. 2007.

[11] 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, pp. 1-6, Oct. 2008.

[12] S. A. Manavski and G. Valle, CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment, in BMC Bioinformatics, 9(Suppl 2):S10. Mar. 2008.

[13] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney, High-throughput sequence alignment using graphics processing units, in BMC Bioinformatics, vol. 8, no. 1, pp. 474-483, Dec. 2007,.

[14] The UniProt/Swiss-Prot Database, http://www.ebi.ac.uk/swissprot/

[15] http://en.wikipedia.org/wiki/Sandy_Bridge_(microarchitecture)

[16] http://en.wikipedia.org/wiki/AMD_Fusion

[17] 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 Symposium on Parallel & Distributed Processing. pp. 1-8. 2009.

[18] Talal Bonny, M. Affan Zidan, and Khaled N. Salama. An Adaptive Hybrid Multiprocessor Technique for Bioinformatics Sequence Alignment. In the 5th Cairo International Conference on Biomedical Engineering (CIBEC 2010). pp. 112-115. December 2010