Optimizing Performance, Cost, and Sensitivity in Pairwise Sequence Search on a Cluster of PlayStations

Ashwin M. Aji and Wu-chun Feng

Abstract— **The Smith-Waterman algorithm is a dynamic programming method for determining optimal local alignments between nucleotide or protein sequences. However, it suffers from quadratic time and space complexity. As a result, many algorithmic and architectural enhancements have been proposed to solve this problem, but at the cost of reduced sensitivity in the algorithms or significant expense in hardware, respectively.**

This paper presents a highly efficient parallelization of the Smith-Waterman algorithm on the Cell Broadband Engine platform, a novel hybrid multicore architecture that drives the low-cost PlayStation 3 (PS3) game consoles as well as the IBM BladeCenter Q22, which currently powers the fastest supercomputer in the world, Roadrunner at Los Alamos National Laboratory. Through an innovative mapping of the optimal Smith-Waterman algorithm onto a cluster of PlayStation 3 nodes, our implementation delivers *21 to 55-fold speed-up* over a high-end multicore architecture and up to *449-fold speed-up* over the PowerPC processor in the PS3.

Next, we evaluate the trade-offs between our Smith-Waterman implementation on the Cell with existing software and hardware implementations and show that our solution achieves the best performance-to-price ratio, when aligning realistic sequences sizes and generating the actual alignment.

Finally, we show that our low-cost solution on a PS3 cluster approaches the speed of BLAST while achieving *ideal* sensitivity. To quantify the relationship between the two algorithms in terms of speed and sensitivity, we formally define and quantify the *sensitivity* of homology search methods so that trade-offs between sequence-search solutions can be evaluated in a quantitative manner.

I. INTRODUCTION

The Smith-Waterman algorithm determines *optimal local alignments* between nucleotide or protein sequences and is therefore used in a wide range of areas from estimating evolutionary histories to predicting behaviors of newly found genes. However, the exponential growth in the nucleotide and protein databases has made the Smith-Waterman algorithm impractical to search on these databases because of its quadratic time and space complexity.[1]

As a result, this led to innovations on the non-algorithmic front — specifically, special-purpose hardware solutions on FPGAs [2], [3], [4] and linear processor arrays [5]. Simultaneously, there were also developments on the algorithmic front that gave rise to heuristics such as FASTA [6] and the BLAST [7] family of algorithms that sacrificed sensitivity for speed. The above solutions present a multitude of trade-offs in terms of speed, cost, and sensitivity of the sequence-search solutions. In contrast, we propose a solution that addresses all the above trade-offs simultaneously — delivering *high speed, low cost, and ideal sensitivity* via a novel parallelization of the Smith-Waterman algorithm on an emergent chip multiprocessing architecture called the Cell Broadband Engine (BE) from Sony, Toshiba, and IBM (STI). This parallelization requires explicitly managed parallelism at different levels of computational granularity in order to expose the full potential of the Cell BE, an architecture that drives both the commodity Sony PlayStation 3 game console and the high-end IBM BladeCenter QS22.[2]

Previously, we presented an innovative approach to parallelize a single pairwise sequence alignment on a single Cell BE using Smith-Waterman. Our implementation performed a *complete* optimal sequence alignment (i.e., calculating the score and generating the alignment) using the affine gap-penalty scoring scheme [8]. In this paper, we extend our design to exploit the computational power of a cluster of 14 PlayStation 3 nodes to dramatically speed-up sequence search. Our accelerated implementation delivers *21 to 55-fold speed-up* over a high-end multi-core architecture and up to *449-fold speed-up* over the PowerPC processor in the PS3.

Next, we evaluate the trade-offs between our Smith-Waterman implementation on the Cell BE platform with existing software and hardware Smith-Waterman implementations. The trade-offs are evaluated based on execution speed, deployment cost of each of the systems, completeness of the solution (i.e., calculating score *and* generating alignment), and maximum sequence size that can be aligned. Our solution for the PS3 cluster achieves the best performance-to-price ratio, when aligning realistic sequences sizes and generating the actual alignment.

Lastly, we present a formal definition and method to quantify the 'sensitivity' of sequence-search algorithms, by analyzing the similarities between homology search and web search methods and their corresponding performance metrics. As an example, we use this quantifying technique to measure the sensitivity of the BLAST algorithm relative to Smith-Waterman and show that although BLAST is much faster than the Smith-Waterman, it misses a majority of the significant alignments that are produced by the optimal Smith-Waterman algorithm. In summary, our low-cost solution approaches the speed of BLAST while achieving ideal sensitivity.

The rest of this paper is organized as follows: Section II describes our experimental platforms. Section III presents the sequential Smith-Waterman algorithm and our design

A. M. Aji is an M.S student from the Dept. of Computer Science, Virginia Tech, Blacksburg, Virginia 24060, USA `aaji@cs.vt.edu`

W. Feng is an Associate Professor in the Dept. of Computer Science, Virginia Tech, Blacksburg, Virginia 24060, USA `feng@cs.vt.edu`

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. IIP-0804155.

[1]Smith-Waterman can be implemented in linear space at the expense of using more time [1].

[2]As of June 2008, an IBM QS22-based system powered the fastest supercomputer in the world, Roadrunner from Los Alamos National Laboratory. It also held the distinction of being the first sustained petaflop supercomputer.

to parallelize it for the Cell cluster. Section IV discusses and defines the performance metrics for evaluating sequence-search algorithms. Section V presents our experimental set-up and methodology and discusses the results. Section VI concludes the paper.

II. EXPERIMENTAL PLATFORMS

In this paper, we compare the performance of our implementation of Smith-Waterman on the PlayStation 3, which hosts the Cell BE, to three other computing platforms: nVIDIA GeForce GPGPU (General-Purpose Computation on Graphics Processing Units), TimeLogic's DeCypher engine, and a standard PC.

The Cell BE is a hybrid multi-core processor that combines a PowerPC core (also known as the Power Processing Element or PPE), and eight SIMD-based processors (also known as the Synergistic Processing Elements or SPEs) [9]. However, the Linux kernel on the PS3 runs on top of a proprietary hypervisor that disallows the use of one of the SPE cores, while another SPE core is hardware-disabled. Thus, we can effectively use only 6 SPE cores for computational purposes. All the cores run at 3.2 GHz. The SPE cores are tightly coupled with the PPE via a high-bandwidth Element Interconnect Bus (EIB). The EIB can transmit 96 bytes/cycle for a theoretical memory bandwidth of 204.8 gigabytes/second (GB/s).

Our PS3 cluster platform consisted of 14 PS3 game consoles, interconnected via Gigabit Etherent and using MPI [10] for communication. We used the IBM Cell SDK 2.1 to program the individual PS3 nodes, where each node ran the Linux Fedora Core 5 OS.

We ran the CUDA-compatible Smith-Waterman implementation, authored by Manavski [11], on the nVIDIA GeForce 8800 GTS graphics card with 512 MB of memory.

We executed a hardware-accelerated Smith-Waterman on the TimeLogic-based DeCypher Engine G4 FPGA card with DeCypher software version 7.6.2. The FPGA card was hosted by a Sun SPARC V9 machine, running at 900 MHz with 16 GB of available physical memory.

Lastly, we ran the naïve software implementation of Smith-Waterman on a single core of an Intel Core 2 Duo E4500 processor, running at 2.2 GHz and having 4 GB of main memory.

III. OPTIMAL LOCAL SEQUENCE ALIGNMENT

A. The Sequential Algorithm

The Smith-Waterman algorithm [12] is an *optimal local sequence alignment* methodology that follows the dynamic-programming paradigm. It can be broadly classified into two phases: (1) matrix filling and (2) backtracing.

To fill out the dynamic-programming matrix (DP), the Smith-Waterman algorithm follows a scoring system that consists of a scoring matrix and a gap-penalty scheme. The scoring matrix, M, is a 2-dimensional matrix containing the scores for aligning individual amino acid or nucleotide residues. The gap-penalty scheme provides the option of gaps being introduced within the alignments, hoping that a better alignment score can be generated; but they incur some penalty or negative score. In our implementation, we consider an affine gap-penalty scheme that consists of two types of penalties. The *gap-open* penalty, o, is incurred for starting a gap in the alignment, and the *gap-extension* penalty, e, is imposed for extending a previously existing gap by one unit. Using this scoring scheme, the dynamic-programming matrix is filled out following a *wavefront* pattern, i.e., beginning from the northwest corner element and going towards the southeast corner, the current anti-diagonal is filled after the previous anti-diagonals are computed, as shown in Figure 1(a). Moreover, each element can be computed only after the calculation of its north, west and northwest neighbors are computed, as shown in Figure 1(b).

The backtracing phase of the algorithm generates the highest-scoring local alignment. The backtrace begins at the matrix cell that holds the optimal alignment score and proceeds in a direction opposite to that of the matrix filling, until a cell with score zero is encountered. The path, thus traced, yields the optimal local alignment.

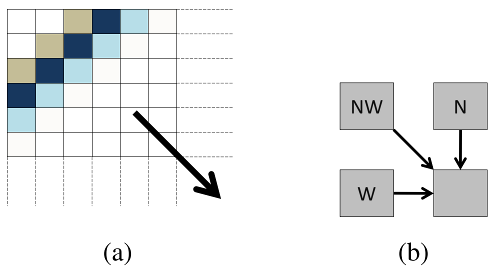


Fig. 1. The Smith-Waterman wavefront algorithm and its dependencies

B. The Parallel Algorithm

The wavefront pattern of the Smith-Waterman algorithm forces consecutive anti-diagonals to be computationally dependent, as shown in Figure 1. The elements on the same anti-diagonal are computationally independent, however, and can be processed in parallel on separate SPE cores of the Cell BE. This, in turn, creates high communication overhead between the cores, which can be avoided by providing more computation to the individual cores. We achieve this by grouping matrix elements into blocks of elements and call each block as a *tile*. While this strategy does not change the wavefront pattern of the algorithm, i.e., the algorithm still advances through the matrix by computing anti-diagonals, it reduces communication overhead since each anti-diagonal will be composed of multiple tiles, as shown in Figure 2. We call this pattern the *tiled-wavefront* pattern.

In mapping the tiled-wavefront pattern to the Cell, we process independent tiles on different SPEs because each core can perform independent asynchronous computations. For simplicity, we assume that the matrix is divided into square tiles. The execution starts by processing tile t_1, as noted in Figure 2. After the processing of tile t_1 completes, the two tiles lying on the anti-diagonal t_2 are processed in parallel on two SPEs. The number of SPEs used increases in the subsequent stages of the tiled wavefront. From the anti-diagonal t_6 onwards, the number of tiles available for parallel processing is equal to or exceeds the number of active SPEs on a single Cell BE in the PlayStation 3 (i.e., 6), and all SPEs can be actively processing tiles.

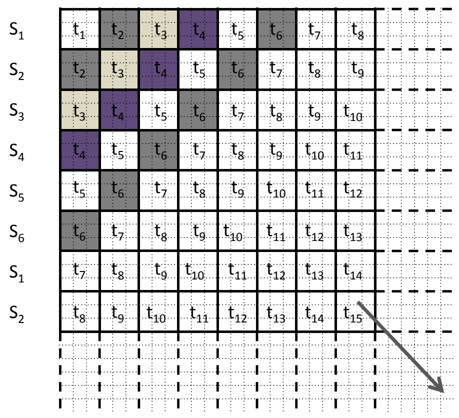

S_1
S_2
S_3
S_4
S_5
S_6
S_1
S_2
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11}
t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}
t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13}
t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14}
t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}

Fig. 2. Tiled wavefront

Our scheduling scheme allows static cyclic assignment of tiles to the SPEs, The SPE that is labeled S_1 computes the topmost row of tiles, S_2 computes the row of tiles below it, and so on. This static scheduling policy achieves perfect load balancing among the SPEs and enables complete utilization of the Cell BE.

We further leverage the above parallelization strategy to search for an unknown DNA sequence in a known DNA sequence database by using a cluster of Cell nodes (i.e., PS3 game consoles). We first partition the known DNA sequence database into as many fragments as the available PS3 consoles, and then distribute the fragments within the cluster such that each PS3 is responsible for searching roughly equal-sized and exclusive partitions of the database. Each Cell node then repeatedly aligns the unknown sequence with every sequence in its local database fragment. Thus, we simultaneously explore two levels of parallelism: coarse-grained parallelism, where different portions of the database are searched independently by different PlayStations, and fine-grained parallelism, where 6 SPE cores of a single PS3 are utilized to speedup the process of aligning a single sequence pair.

The resulting alignment set from each node in the cluster is then collected by the root node, and the collective alignment set is given to the user.

IV. PERFORMANCE METRICS

Sequence-search algorithms can be measured along many dimensions such as execution time (or speed), millions of cell updates per second (MCUPS), deployment cost, and sensitivity. While measuring the first three metrics is straightforward, there is no existing definition that clearly defines and quantifies the 'sensitivity' of a sequence-search algorithm.

A. Sensitivity

Previous work defined and measured sensitivity in an unconvincing and informal fashion [13], [14]. To address this, we propose a formal definition for sensitivity in the following way. Homology search methods are similar to web-search algorithms. In the web-search domain, an input query or keyword is searched against a large known document collection. The output will be a set of relevant web pages that are sorted by closeness or rank. Similarly, in the realm of homology search, an input query sequence is searched against a large known sequence database. The output will be a set of relevant sequences similar to the query, which are sorted by the alignment score or corresponding statistical quantifiers such as E-Value and P-Value of the alignment. Given the analogy between the homology search and web-search methods, we first explore one of the many definitions and metrics that have been proposed to measure the performance of information retrieval systems. We analyze its relevance to sequence search and then modify and adapt this definition in order to quantify sensitivity. The information retrieval metric that is of interest is as follows:

Among all the documents that are relevant to a query, the fraction of the documents that is successfully retrieved is termed as *recall* and can be represented by Equation (1). It gives an indication of the percentage of *false negatives* that are not included in the final result set of the search. In other words, recall denotes the power of the search algorithm to retrieve all the relevant documents.

$$recall = \frac{|all\ relevant\ docs.\ \bigcap\ retrieved\ docs.|}{|all\ relevant\ docs.|} \tag{1}$$

With respect to sequence-search algorithms, the universal relevant document group or the *absolute result set* corresponds to all the sequence alignments that are generated by the optimal Smith-Waterman algorithm, for a given threshold score. Different threshold scores generate different absolute result sets of sequence alignments. False negatives can be generated by those heuristic algorithms, which are willing not to output some high-scoring (relevant) sequences in order to obtain large speed improvements. This is typically the case with heuristics such as BLAST, FASTA, and PatternHunter, for example. Thus, we consider 'recall' as a relevant metric to compare homology search methods. With this background, we can now define and quantify the term *sensitivity*.

Among the sequence alignments generated by the Smith-Waterman algorithm for a given threshold score, the fraction of the alignments that is successfully generated for the same threshold score by the algorithm under test is denoted as the *sensitivity* of that algorithm for that threshold score.

Let χ represent the set of scores of all the statistically significant alignments[3] that are generated by the Smith-Waterman algorithm. If we consider each element of the set χ as a potential threshold score, then the sensitivity of the test algorithm at the different threshold scores in χ can be represented by the Equation (2):

$$sensitivity_i = \frac{|S_i \bigcap T_i|}{|S_i|} \tag{2}$$

where $i \in \chi$ are the set of threshold scores, $sensitivity_i$ is the sensitivity at the threshold score i, S_i is the result set generated by SW with alignment scores $\geq i$, and T_i is

[3]The statistical significance of an alignment can be inferred by examining the corresponding E-values and P-values.

the result set generated by the test algorithm with alignment scores $\geq i$.

Since no sequence-search algorithm generates false positives, the result set generated by the test algorithm is contained in the absolute result set generated by Smith-Waterman, i.e. $T_i \subseteq S_i$. Therefore, Equation (2) becomes

$$sensitivity_i = \frac{|T_i|}{|S_i|} \tag{3}$$

Sensitivity is therefore a function of the threshold score. To assign a unified sensitivity value to a sequence-search algorithm, we take the mean of sensitivity values at all the threshold scores in χ, as shown in Equation (4). Empirical results from Section V-B show that the sensitivity values for different threshold scores have very low variance, and thus, their mean value provides a good estimate of the sensitivity of the algorithm.

$$Sensitivity = \frac{\sum_{i \in \chi} sensitivity_i}{|\chi|} \tag{4}$$

Hence, the target for any sequence-search algorithm is to provide a result set that is identical to that of Smith-Waterman, thereby achieving a perfect sensitivity of 1. If the sensitivity is less than 1, it means that the sequence-search algorithm has *missed* generating significant alignments.

V. EXPERIMENTS

A. Set-Up and Methodology

To test our Smith-Waterman design, we conduct experiments to search for a single (potentially unknown) DNA sequence (query) against a known DNA sequence database.

The performance of Smith-Waterman depends only on the size of the sequences that are being aligned and not on their contents. So, we can conveniently choose synthetic or real biological sequence data in our experiments, without any change in the execution times of the application. Hence, each character in the query sequence was generated independently and uniformly at random from the alphabet A, C, G, T. We choose 6 different DNA sequences as 6 independent queries, their sizes ranging from 256 to 3584 characters, respectively. Given that 95.66% of the sequences in the NCBI NT database are 5000 characters in size or less [15], we evaluate our parallel design by using realistic sequence sizes.

We search the 6 chosen unknown sequences against the Drosophila NT database (downloaded from the NCBI site), which contains 1170 known DNA sequences that include approximately 3.5 million symbols.

To assess the scalability of our design, we executed it on our 14-node PS3 cluster and varied the number of nodes in the cluster from 1 to 14.

Using the performance metrics from Section IV, we compare Smith-Waterman performance on different platforms: our implementation on the Cell BE (PS3) cluster, Manavski's solution on the nVIDIA GeForce GPGPU, Smith-Waterman on the TimeLogic card, and the canonical Smith-Waterman on a general-purpose PC. We define two types of Smith-Waterman implementations: partial and complete. We define *partial* solutions as those that only calculate the alignment scores. These implementations do *not* perform the back-trace operation to generate the optimal alignments. On the other hand, *complete* Smith-Waterman solutions calculate the alignment scores *and* generate the optimal alignments.

Manavski's GPGPU implementation delivers a partial solution. TimeLogic provides a partial solution on their FPGA accelerators (referred to as *TimeLogic-only*) but then can use these results to perform a complete alignment on the host servers (referred to as *TimeLogic + CPU server*). We provide *complete* Smith-Waterman solutions on the PS3 and PC platforms.

We then compare the performance of our PS3 implementation of Smith-Waterman against BLAST executed on the PC. The blastall program was executed with the following parameter list:

```
blastall –p blastn –d <database_file> –i <query_file>
          –o <output_file> –F F –S 1 –e 100
```

The parameter list denotes that the blastn program is being executed by switching off the query sequence filtering, and only the top query strands are being searched against the database. The threshold expectation value (E-value) of the output alignments is set at 100.

For the experiments in this section, we chose a gap-open penalty of -5 and gap-extension penalty of -1 and allotted 3 points for a character match and -1 for a mismatch.

B. Results

Below we present our results in three parts. First, we discuss the speedup of parallel Smith-Waterman on the Cell platform. Second, we compare the performance of parallel Smith-Waterman on the Cell cluster to other implementations discussed in previous sections. Finally, we show that our solution approaches the speed of BLAST but maintains *ideal* sensitivity.

1) Speedup of Smith-Waterman on the PS3 Cluster: In [8], we concluded that the tiled-wavefront scheme, which was also discussed briefly in Section III-B, scales linearly within the Cell architecture, i.e., if more cores were available on the Cell chip, then we might have seen a larger speedup for aligning a single pair of sequences. Here we execute our extended parallel Smith-Waterman implementation on a PS3 cluster having 14 nodes. The sizes of the 6 query sequences range from 256 to 3584 characters and are named as Query_256 through Query_3584, respectively. Figure 3 shows that we achieve a 21 to 55 times maximum speedup over the canonical Smith-Waterman implementation on the PC. Also, the scalability of our implementation is close to linear for all the query sizes, i.e., we get better performance as we employ more PS3 nodes and our design is highly scalable for larger clusters.

We measure a 449-fold speedup of our parallel algorithm over the sequential implementation when the PPE was the basis of calculation. But we note that the PPE has very limited computational capabilities, and thus, using the PPE core as a basis for speedup calculation artificially inflates our results without much benefit.

2) Smith-Waterman on PS3 vs. Others: We compare the various Smith-Waterman solutions based on their performance per dollar or MCUPS/\$, as shown in Figure 4.

Platform of Execution	Cost of Deployment (U.S. Dollars)	Sequence sizes													
		256		512		1024		2048		3072		3584		> 3584 (5120)	
		MCUPS	MCUPS per $ (scaled)	MCUPS	MCUPS per $ (scaled)	MCUPS	MCUPS per $ (scaled)	MCUPS	MCUPS per $ (scaled)	MCUPS	MCUPS per $ (scaled)	MCUPS	MCUPS per $ (scaled)	MCUPS	MCUPS per $ (scaled)
SW on PC	**800**	8.07	1	8.03	1	7.81	1	7.75	1	7.77	1	7.69	1	5.43	1
SW on CUDA	*1170*	*833*	*70.54*	*901*	*76.69*	*940*	*82.27*	*960*	*84.66*	*N/A*		*N/A*		*N/A*	
SW on TimeLogic	*20000*	*3613*	*17.90*	*5657*	*28.17*	*7257*	*37.15*	*8798*	*45.39*	*8928*	*45.98*	*9390*	*48.82*	*9676*	*71.32*
SW on TimeLogic + CPU	**20000**	194	0.96	209	1.04	219	1.12	137	0.71	224	1.15	227	1.18	148	1.09
SW on PS3	**5586**	167.33	**2.97**	272.92	**4.87**	342.97	**6.29**	399.89	**7.39**	415.41	**7.66**	421.74	**7.85**	N/A	

Fig. 4. Table comparing the performances of the various existing Smith-Waterman (SW) software and hardware solutions. The italicized grey fonts in the table mean that the solutions are *partial*, i.e., they only calculate the scores without generating the alignment. The other entries in the table indicate *complete* SW solutions, i.e. they calculate the scores as well as generate the alignments. Performances are measured in MCUPS and MCUPS/$. For the purpose of insightful performance comparison, all the MCUPS/$ values are scaled such that the value of the naïve solution of SW on PC is 1.

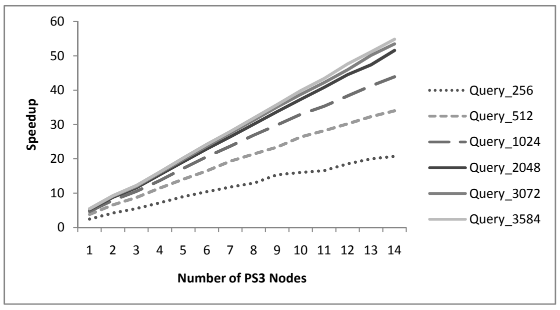


Fig. 3. Speedup chart for the PS3 cluster.

Figure 4 presents the multitude of trade-offs, based on the completeness of the Smith-Waterman implementation (i.e., partial or complete), the maximum sequence size that can be aligned, MCUPS and the cost of deployment (represented as MCUPS per dollar) of each of the systems.

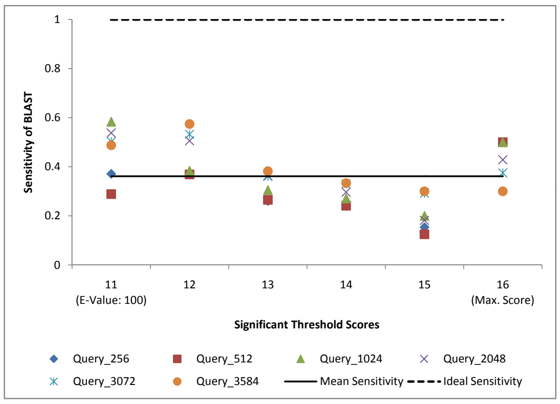
The TimeLogic-only solution clearly offers the highest MCUPS, but the hardware is expensive. The CUDA solution on the GPGPU is the best in terms of MCUPS/$, but it cannot align sequences whose lengths are 2048 characters, and thus, it does not align a significant amount of realistic sequences (refer to Section V-A). Moreover, the previous two solutions provide only a *partial* Smith-Waterman solution (i.e., no backtrace). While the TimeLogic + CPU Server and the PC implementations can align all the realistic sequence sizes, they have very low MCUPS/$.

Among all the complete Smith-Waterman solutions, our design for the PS3 cluster is arguably the best – we have the highest MCUPS/$, and we align realistic sequences sizes.

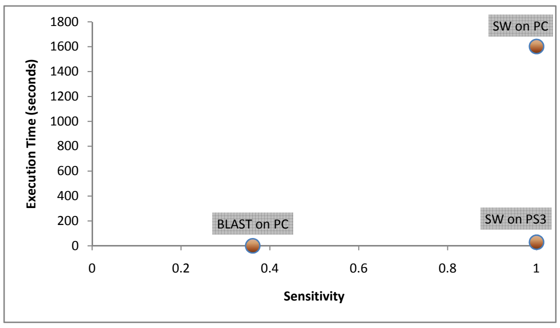
Why are the partial solutions much faster than the complete ones? The backtrace operation requires the entire matrix to be present in the physical memory beforehand. If the application is not performing the backtrace, many clever heuristics can be performed to pipeline the matrix-filling phase without having to store the entire matrix. Also, the smaller memory requirement for the partial solutions means that multiple pairwise sequence alignments can simultaneously fit into the limited memory resources, thereby increasing the MCUPS.

3) Speed vs. Sensitivity: Here we show that our solution on the PS3 cluster approaches the speed of BLAST but at *ideal* sensitivity. To measure the sensitivity values of Smith-Waterman and BLAST, we first assume the sensitivity of Smith-Waterman to be 1 and measured the sensitivity of BLAST relative to that, using the equations from Section IV. We first list the scores and E-values of each sequence alignment that was generated by BLAST and Smith-Waterman. All the alignments generated by Smith-Waterman had E-values ≤ 100, and we chose their corresponding alignment scores as the set of *significant threshold scores*, χ, as shown in Figure 5(a). Using Equation (3), we calculated the sensitivity of BLAST for each threshold score in χ. We repeated the above procedure for each of the input query sequences, and from all the sensitivity values, we found that the mean sensitivity value was 0.36 with a very low variance of 0.01 on an average, across all input query sequences. The consistent empirical results across a variety of input query sequences and threshold scores led us to formulate Equation (4), and we therefore chose 0.36 as the sensitivity of the BLAST algorithm.

Figure 5(b) shows that BLAST is faster than all the Smith-Waterman implementations, but the sensitivity value of 0.36 relative to Smith-Waterman indicates that BLAST *misses* about 64% of the significant alignments on average. On the other hand, the parallel Smith-Waterman implementation on the PS3 cluster is an order-of-magnitude faster than the conventional sequential version, and its performance approaches that of BLAST but with perfect (ideal) sensitivity of 1. The lower right part of the Figure 5(b) is the *ideal* point for sequence search, i.e., high sensitivity and low execution time should be the target for all future sequence-search solutions.



(a)



(b)

Fig. 5. (a) Sensitivity versus threshold scores and (b) sensitivity versus execution time.

VI. CONCLUSIONS

This paper addresses one of the hardest computational problems in the bioinformatics community — executing *optimal* sequence search algorithm *quickly* on *inexpensive* hardware. Our efforts have been directed to contribute novel schemes to design, implement, and optimize an optimal local sequence-alignment algorithm, Smith-Waterman, to execute on the powerful Cell BE that drives the inexpensive Sony PlayStation 3. Through an innovative mapping of the optimal Smith-Waterman algorithm onto a cluster of 14 PlayStation 3 nodes, our implementation delivers *21 to 55-fold speed-up* over a high-end multicore architecture and up to *449-fold speed-up* over a non-accelerated PS3. Also, the scalability of our implementation is nearly linear, and thus, our design is highly scalable for larger clusters. We compare our solution to the other existing software and hardware solutions and show that we achieve the highest execution speed per U.S dollar. Moreover, we align realistic sequences sizes and generate the actual alignment. Finally, we formally define the term 'sensitivity' of homology search methods. We used this definition to quantify the sensitivity of BLAST and found that it missed many significant sequence alignments. Having quantified the relationship between Smith-Waterman and BLAST in terms of speed and sensitivity, we show that our inexpensive solution on the PS3 cluster approaches the speed of BLAST but at ideal sensitivity.

As future work, we intend to investigate the integration of the parallel Smith-Waterman for the Cell into sequence alignment toolkits. Also, we intend to explore the challenges of mapping other sequence-search algorithms, such as the popular BLAST onto powerful, yet inexpensive, general-purpose computing hardware like the Cell BE and the GPGPU.

REFERENCES

[1] Myers, Eugene W. and Miller, Webb, "Optimal alignments in linear space," *Bioinformatics*, vol. 4, no. 1, pp. 11–17, 1988. 10.1093/bioinformatics/4.1.11.

[2] D. Lavenier, "Dedicated hardware for biological sequence comparison," vol. 2, no. 2, pp. 77–86, 1996.

[3] TimeLogic Biocomputing Solutions, "DeCypherSW," *URL: http://www.timelogic.com/downloads/decyphersw.pdf. Accessed: 2008-02-02.(Archived by WebCite at http://www.webcitation.org/5VK0AyWiI).*

[4] Y. Yamaguchi, T. Maruyama, and A. Konagaya, "High Speed Homology Search with FPGAs.."

[5] R. Hughey, "Parallel hardware for sequence comparison and alignment," 1996.

[6] D. J. Lipman and W. R. Pearson, "Rapid and sensitive protein similarity searches.," *Science*, vol. 227, pp. 1435–1441, March 1985.

[7] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool.," *J Mol Biol*, vol. 215, pp. 403–410, October 1990.

[8] A. M. Aji, W. Feng, F. Blagojevic, and D. S. Nikolopoulos, "Cell-SWat: Modeling and Scheduling Wavefront Computations on the Cell Broadband Engine," in *Proc. of the ACM International Conference on Computing Frontiers*, May 2008.

[9] J. A. Kahle and M. N. Day and H. P. Hofstee and C. R. Johns and T. R. Maeurer and D. Shippy, "Introduction to the Cell multiprocessor," in *IBM Journal of Research and Development*, pp. 589–604, Jul-Sep 2005.

[10] Dan Nagle, "MPI – The Complete Reference, Vol. 1, The MPI Core, 2nd ed., Scientific and Engineering Computation Series, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongarra," *Sci. Program.*, vol. 13, no. 1, pp. 57–63, 2005.

[11] Svetlin A Manavski and Giorgio Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, 2008.

[12] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, pp. 195–197.

[13] B. Ma, J. Tromp, and M. Li, "Patternhunter: faster and more sensitive homology search," 2002.

[14] M. Li, B. Ma, D. Kisman, and J. Tromp, "PatternHunter II: Highly sensitive and fast homology search," 2003.

[15] M. K. Gardner, W. Feng, J. Archuleta, H. Lin, and X. Ma, "Parallel genomic sequence-searching on an ad-hoc grid: Experiences, lessons learned, and implications," *Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference*, p. 22, 11-17 Nov. 2006.