Revisiting the Speed-versus-Sensitivity Tradeoff in Pairwise Sequence Search

Ashwin M. Aji and Wu-chun Feng
The Synergy Laboratory
Department of Computer Science
Virginia Tech
{aaji,feng}@cs.vt.edu

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. Hence, there exists a need to evaluate the tradeoffs between the different solutions. This motivation, coupled with the lack of an evaluation metric to quantify these tradeoffs leads us to formally define and quantify the *sensitivity* of homology search methods so that tradeoffs between sequence-search solutions can be evaluated in a quantitative manner. As an example, though the BLAST algorithm executes significantly faster than Smith-Waterman, we find that BLAST *misses* 80% of the significant sequence alignments.

This paper then presents a highly efficient parallelization of the Smith-Waterman algorithm on the Cell Broadband Engine, a novel hybrid multicore architecture that drives the PlayStation 3 (PS3) game consoles, and emulates BLAST by repeatedly executing the parallelized Smith-Waterman algorithm to search for a query in a given sequence database. Through an innovative mapping of the optimal Smith-Waterman algorithm onto a cluster of PlayStation 3 nodes, our implementation delivers a *10-fold speed-up* over a high-end multicore architecture and an *88-fold speed-up* over a non-accelerated PS3.

Finally, we compare the performance of our implementation of the Smith-Waterman algorithm to that of BLAST and the canonical Smith-Waterman implementation, based on a combination of three factors — execution time (speed), sensitivity, and the actual cost of deploying each solution. In the end, our parallelized Smith-Waterman algorithm approaches the speed of BLAST while maintaining ideal sensitivity and achieving low cost through the use of PlayStation 3 game consoles.

1 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 to identifying possible drugs to cure prevalent diseases. 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.

As a result, this led to innovations on the non-algorithmic front — specifically, special-purpose, but quite expensive, hardware solutions on FPGAs [11, 10, 5] and linear processor arrays [4]. Simultaneously, there were also developments on the algorithmic front that gave rise to heuristics such as FASTA [7] and the BLAST [1] family of algorithms that sacrificed sensitivity for speed. With the emergence of a wide range of homology search methods, the need to evaluate the trade-offs between these solutions has become quite important. The *speed* of algorithms can be easily compared by measuring their relative execution times. However, there exists no formal method in which the *sensitivity* of a sequence search algorithm is quantified.

Previous works that have attempted to measure sensitivity have done so in an informal and open-ended fashion [8, 6]. In this paper, we present a formal definition and method to measure the sensitivity of sequence-search algorithms by analyzing the similarities between homology search and web search methods and corresponding definitions. As an example, we use this quantifying technique to measure the sensitivity of the BLAST algorithm, relative to Smith-Waterman, and show it to be only 0.2 times as sensitive as Smith-Waterman, which means that the BLAST heuristic *misses* 80% of the significant alignments that are produced by the optimal Smith-Waterman algorithm. By quantifying both the speed and sensitivity of pairwise sequence-search algorithms like Smith-Waterman and BLAST, we can concretely articulate our goal of accelerating Smith-Waterman to approach the speed of BLAST while maintaining ideal sensitivity (as well as low cost via the Playstation 3)

In addition to providing the means to compare sequence-search methods, we attempt to achieve both high speed and ideal sensitivity by parallelizing the Smith-Waterman algorithm on the emerging (and arguably, commodity) Chip multi-processors like that Cell Broadband Engine (BE). The Cell processor is a combined effort from Sony, Toshiba, and IBM, and it contains heterogeneous cores and specialized accelerators on the same chip and also drives the Sony PlayStation 3 game console.[1] The Cell BE possesses an aggregate single-precision floating-point performance of 204.8 Gflops, thus providing the necessary computational horsepower for scientific computing such as the pairwise sequence searching found in Smith-Waterman.

We present an innovative scheme to implement Smith-Waterman on the Cell BE, that completely utilizes all the accelerator cores on the chip to the fullest capacity. We then emulate the BLAST algorithm, by executing the parallel pairwise alignment algorithm multiple times to search a query sequence through an entire sequence database. Further, we parallelize this scheme by utilizing a cluster of PS3 nodes, so that each node searches a fragment of the database simultaneously.

In summary, to address the ever-increasing need to more quickly search biological databases, computational scientists have proposed a plethora of faster homology sequence searches *but* at the expense of using heuristics that reduce the sensitivity of the searches or using expensive hardware to produce ideal sensitivity. This motivation has led us to define and quantify the *sensitivity* of homology search methods, so that trade-offs between solutions could be evaluated in a quantitative manner. Further, we attempt to achieve both high speed and ideal sensitivity on the emerging (and arguably, commodity) Cell BEs that drive the PlayStation game consoles. Through an innovative mapping of the optimal Smith-Waterman algorithm onto the cluster of PlayStation 3 nodes, our implementation delivers a *10-fold speed-up* over a high-end multi-core architecture and an *88-fold speed-up* over a non-accelerated PS3. Finally, we compare the performance of our implementation of the Smith-Waterman algorithm with that of BLAST and the naive Smith-Waterman implementation based on three factors – execution time (speed), sensitivity and the actual cost of deploying each solution. Our solution is found to approach the speed of BLAST, at ideal sensitivity and low costs, which is ideal to resolve the homology search problem.

The rest of this paper is organized as follows: Section 2 outlines the Cell Broadband Engine (BE) architecture. Section 3 presents the sequential Smith-Waterman algorithm and our design

[1]This computer is a game console that currently costs a mere $399.

to parallelize it for the Cell BE and architectures with asymmetric cores. Section 4 formally defines and quantifies the 'sensitivity' of any sequence-search algorithm. Section 5 presents our experimental set-up and methodology and discusses the results. Section 6 concludes the paper.

2 Experimental Platform

In this study, we used a cluster of 22 PlayStation 3 (PS3) nodes, out of which, 14 nodes were available to us at all times. The PS3 nodes are connected to a 1000BASE-T Gigabit Etherent switch, and they communicate via MPICH2-1.0.7 library calls. We used the IBM Cell SDK 2.1 to program the individual PS3 nodes.

The Cell Broadband Engine (BE) is the main processing workhorse within the PS3. The Cell is a hybrid muti-core processor combining a two-way SMT 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) [2]. 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 is a 4-ring structure, capable of transmitting 96 bytes per cycle, for a theoretical memory bandwidth of 204.8 gigabytes/second (GB/s).

The PPE is a 64-bit SMT processor running the PowerPC instruction set architecture (ISA) with vector/SIMD multimedia extensions. The PPE consists of two levels of on-chip cache, L1-I and L1-D with a capacity of 32 KB each and L2 with a capacity of 512 KB.

Each SPE has two main components, the Synergistic Processor Unit (SPU) and the Memory Flow Controller (MFC). The SPU has 128 registers, each of which is 128-bits wide, and 256 KB of software-managed Local Storage (LS). Each SPU can only access its own local storage with direct loads and stores. The MFC performs memory transactions between the local storage and main memory by issuing DMA requests. The ISA of the SPU is different from that of the PPE, using vector execution units that implement Cell-specific SIMD intrinsics on the registers local to the SPU. Varying degrees of computation-communication overlap are possible due to the lack of cohesion between the SPU and MFC.

3 Optimal Local Sequence Alignment on the Cell

This section describes the sequential Smith-Waterman algorithm, followed by the design and implementation details of parallelizing Smith-Waterman for the Cell.

3.1 The Sequential Algorithm

The Smith-Waterman algorithm [9] is an *optimal local sequence alignment* methodology that follows the dynamic programming paradigm, where the intermediate alignment scores are stored in a matrix before the maximum alignment score is calculated. Next, the matrix entries are inspected, and the highest-scoring local alignment is generated. The Smith-Waterman algorithm can thus 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 (or opening) a gap in the alignment, and the *gap-extension* penalty, e is imposed for extending a previously existing gap by one unit. The gap-extension penalty is usually smaller than the gap-open penalty.

Using this scoring scheme, the dynamic-programming matrix is filled out following a *wave-front* 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.

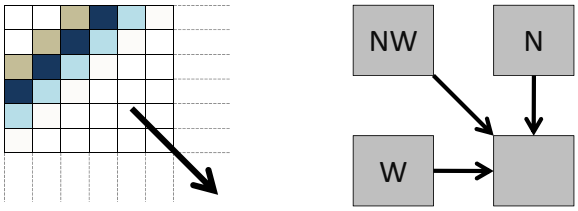


FIGURE 1: The Smith-Waterman wavefront algorithm and its dependencies

3.2 The Parallel Algorithm

In this section, we present the design and implementation of our parallelized Smith-Waterman algorithm to align a pair of sequences on the PlayStation 3.

3.2.1 Design

The wavefront computation pattern of the Smith-Waterman algorithm forces consecutive anti-diagonals to be computationally dependent, as shown in Figure 1. However, the elements on the same anti-diagonal are computationally independent and can be processed in parallel on separate SPE cores of the Cell BE. However, this scheme 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*. This strategy, however, does not change the wavefront pattern of the algorithm, i.e., the algorithm still advances through the matrix by computing anti-diagonals, but each anti-diagonal will be composed of multiple tiles, as shown in Figure 2. We call this pattern as the *tiled-wavefront*.

The most critical aspects of parallelizing the Smith-Waterman algorithm are (1) scheduling the execution of the tiles on the SPEs, (2) communication among the SPEs that process the tiles, and (3) the implementation optimizations required to execute a single tile on the SPE. These concepts are explained in more detail below.

Tiled-wavefront scheduling: 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 the sake of 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. Increasing numbers of SPEs are utilized 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 chip in the PlayStation 3 (i.e., 6), and all SPEs can be actively processing tiles.

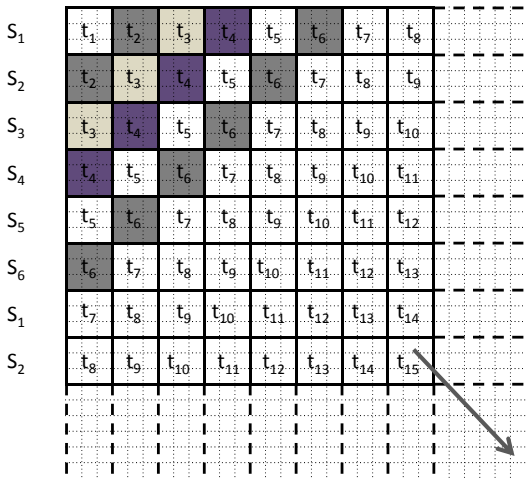


FIGURE 2: Tiled wavefront

Our scheduling scheme allows static assignment of tiles to the SPEs, as outlined in the Figure 2. 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. The SPEs choose the row of tiles in a cyclic fashion and process them, eventually filling the entire matrix. This static scheduling policy achieves perfect load balancing among the SPEs and also enables complete utilization of the Cell chip.

Communication-computation pattern: Figure 3 shows A detailed description of the communication mechanism used to obtain the data necessary for tile computation by each SPE. First, our algorithm fetches the elements on the north and west boundaries of the tile in order to compute the non-boundary elements within the tile. The northwest boundary element is fetched while collecting either the boundary elements from the north or west. Next, each SPE completes the processing of the non-boundary elements. Finally, the SPE moves the processed tile to the main memory for post-processing. The SPE simultaneously sends a notification to the SPE, which will process the south tile, that the boundary elements are ready for transfer. The above process repeats until all the non-boundary tiles of the matrix are processed. For the remaining tiles, the boundary conditions can be easily checked, and the redundant steps can be avoided.

Implementation optimizations: Each tile is physically stored in memory as a one-dimensional (1D) array, where consecutive anti-diagonals are stored as a continuous linear array. We call this representation as the *diagonal-major* format. This storage mechanism is different from the traditional row-major or column-major storage formats that are typically used to represent multi-dimensional arrays in physical memory. The diagonal-major format makes it easier to perform vector operations on each anti-diagonal to compute the elements of the tile. The SPE is a vector processor, where batches of four integers can be processed in a single clock cycle. If we do not arrange the tile as described, the matrix elements cannot be grouped efficiently into batches of

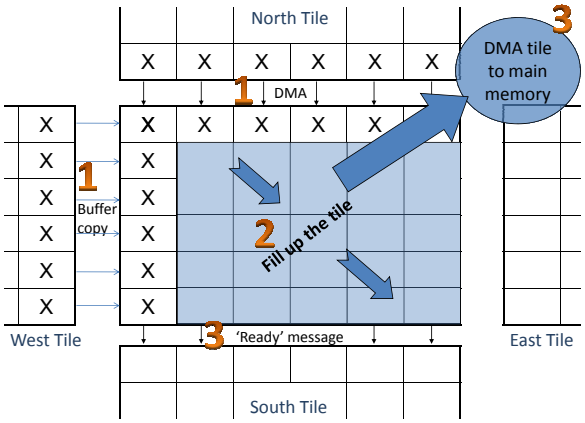


FIGURE 3: Communication-computation pattern.

four integers before processing, which may cause many clock cycles to be wasted and cause a performance hit over the sequential implementations.

4 Performance Metrics

In this section, we define and discuss the metrics that we use to estimate the performance of sequence-search algorithms in general. Later, we use these metrics to compare the performance of our parallel implementation of Smith-Waterman against that of BLAST. We also compare the performance of the parallel Smith-Waterman on different configurations of the PS3 against Smith-Waterman on a general-purpose CPU.

Sequence-search algorithms can be measured on many dimensions such as execution time (or speed), sensitivity, complexity of implementation, and cost of deployment. In this paper, we specifically consider execution time, sensitivity, and cost of deployment as the metrics of interest. While measuring the execution time of an algorithm and estimating the cost of deployment of the complete system are straightforward, there is no existing definition that clearly defines and quantifies the 'sensitivity' of a sequence-search algorithm, which we discuss below.

4.1 Sensitivity

Previous work defined and measured sensitivity in an unconvincing and informal fashion [8, 6]. 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 some of the many definitions and metrics that have been proposed to measure the performance of information retrieval systems. We then analyze their relevance to sequence search, and later, modify and adapt those definitions in order to quantify sensitivity. The information retrieval metrics that are of

interest are as follows:

- *Precision:* Among all the retrieved documents, the fraction of documents that is relevant to the user's information need is termed as *precision*. It gives an indication of the percentage of false-positives that are included in the final result set of the search.

- *Recall:* Among all the documents relevant to the 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.

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

Relevance to homology search: 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 positives are created by assigning a score that is greater than the optimum to a typically low-scoring (irrelevant) alignment, which may cause the irrelevant alignment to cross the threshold score and appear in the final result set. However, no sequence-search algorithm assigns a score that is higher than the optimum to any alignment. Therefore, false positives cannot be generated by this class of algorithms, thus eliminating 'precision' as a relevant metric to compare sequence-search algorithms.

False negatives, on the other hand, can be generated by those heuristic algorithms that 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. Therefore, we consider 'recall' as a relevant metric to compare homology search methods. With this background, we can now define and quantify the term *sensitivity*.

Definition Among all the sequence alignments that are 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[2] 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	$\in$	χ, the set of threshold scores
$sensitivity_i$	=	sensitivity at the threshold score i
S_i	=	result set generated by Smith-Waterman with alignment scores $\geq i$
T_i	=	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. $T_i \subseteq S_i$.

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

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 5.2 show that the sensitivity values for different threshold scores have very low variance, and therefore, their mean value gives a good estimate of the sensitivity of the algorithm.

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

Therefore, 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 value is less than 1, it means that the sequence-search algorithm has *missed* generating significant alignments.

5 Experiments

This section presents our experimental set-up and methodology, followed by our results.

5.1 Set-Up and Methodology

Our experiments were run on a cluster of 22 PlayStation 3 nodes, 14 of which were always available to us at all times. To test the performance of our implementation of Smith-Waterman to align a pair of sequences, we executed our code on a single, dedicated node.

To evaluate the speedup obtained by our parallelized version of Smith-Waterman on the Cell, we aligned sequences of realistic sizes as are currently present in the NCBI Genbank nucleotide (NT) database. There are approximately 3.5 million sequences in the NT database. Of those, 95.66% are 5 KB in size or less [3]. For the purpose of conducting the experiment, we randomly generated sequence pairs with sizes varying from 512 bytes to 3.6 KB, which spanned across most of the realistic spectrum of sequence sizes. Moreover, the running time of the Smith-Waterman algorithm is dependent only on the size of the sequence and not on its contents, and this justifies our choice of randomly generated input sequences.

To emulate the execution of BLAST, we executed the sequential Smith-Waterman algorithm on a PC, for each of the sequence pairs formed by a sample DNA query sequence and the sequences in the Drosophila nucleotide database (provided by NCBI). We executed our parallel implementation of Smith-Waterman on the PS3 cluster. We first partitioned the Drosophila database into as many fragments as the available nodes, and then distributed the fragments within the cluster such that each PS3 console was responsible for searching through equal-sized fractions of the database. Thus, we 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. We repeated the above experiment for input queries of 4 different sizes, ranging from 512 bytes to 3.6 KB.

The BLAST program that was used was blastall where the same query sequences were searched against the Drosophila database. The program with the parameter list used is as follows:

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 with filtering the query sequence being turned off, and only the top query strands are searched against the database. The threshold expectation value (E-value) is set at 100.

5.2 Results

Below we present our results in two parts. In the first part, we show how our parallel implementation of Smith-Waterman on the Cell achieves linear speedup for up to 14 PlayStation 3 nodes. In the second part, we compare the performance of Smith-Waterman on the Cell cluster to traditional implementations of sequence search, i.e. BLAST and Smith-Waterman on a PC, respectively.

5.2.1 Speedup

We first tested our implementation of parallel Smith-Waterman on a single PS3 node by measuring the execution time for different combinations of the input sequence sizes, number of SPEs, and different tile sizes. We achieved optimum performance when we used all the available cores of the PS3 Cell to perform a single pairwise sequence alignment for all sequence sizes up to 3.6 KB. We observed that our version of Smith-Waterman was 7.7 times faster than the sequential version that was executed on the PPE core of the PS3 and 1.75 times faster than the execution time measured on a 2.8-GHz dual-core Intel processor with 2-GB memory. On inspecting these speedup values, we note that the PPE has very limited computational capabilities, and thus, using the PPE core as a basis for speedup calculation artifically inflates our results without much benefit. These speedup values were consistent across the different input sequence sizes and chosen tile sizes. From the results, we conclude that the tiled-wavefront scheme works well with the Cell architecture and if more cores were available, then we might have seen a larger speedup for aligning a pair of sequences. Also, we could align larger sequences on the PlayStations if more memory is made available in them.

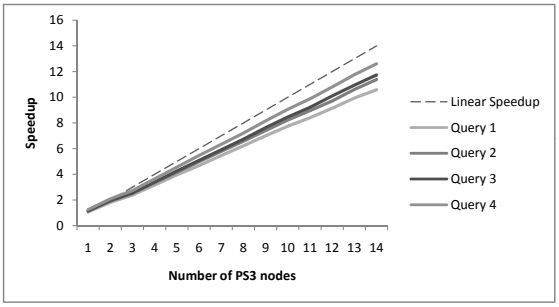


FIGURE 4: Speedup chart for the PS3 cluster.

To emulate BLAST, we developed a scheme as discussed in Section 5.1 where we executed our parallel Smith-Waterman implementation on the PS3 cluster comprising of up to 14 nodes. We conducted the above experiment for four different query sizes ranging from 512 bytes to 3.6 KB (named Query 1 through Query 4 respectively). Figure 4 (b) shows that we achieve a 10.5 times speedup for all the query sizes and the scalability of our implementation is close to linear, i.e. we

get better performance as we employ more PS3 nodes and our design is highly scalable for larger clusters. The above speedup values are measured by using the 2.8-GHz Intel processor as a basis. We measured an 88-fold speedup of our parallel algorithm over the sequential implementation when the PPE was the basis of calculation, which is an inflated value as discussed.

5.2.2 Smith-Waterman vs. BLAST

In this section, we compare our parallel Smith-Waterman implementation on the PS3 cluster to BLAST on a PC and Smith-Waterman on a PC. The metrics of comparison are execution time, cost of deployment and sensitivity of the algorithm as defined in Section 4.

Execution Times: In this section, the parallel execution times of Smith-Waterman from Section 5.2.1 are used in comparison to the other two implementations. The sequential version of Smith-Waterman and the blastall program (from the NCBI BLAST suite) were executed with the same input query and database sequences. Figure 5 shows a 3-dimensional view of the execution times that were recorded for searching the four input query sequences against the given database using BLAST and the two flavors of Smith-Waterman.

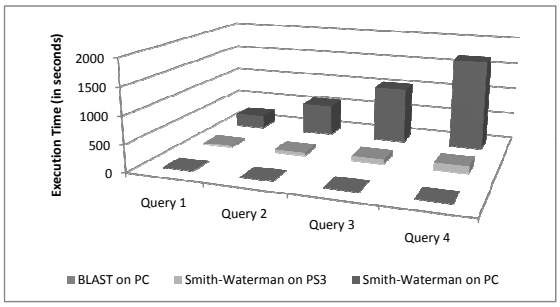


FIGURE 5: Execution times of BLAST and flavors of Smith-Waterman for different query sizes.

We first note that BLAST on a PC is three orders of magnitude faster than Smith-Waterman on a PC. However, by parallelizing Smith-Waterman on the PS3 cluster, we have moved an order of magnitude closer to the execution times of BLAST.

Cost of deployment: A high-end PC is necessary to efficiently execute computationally intensive applications such as BLAST or Smith-Waterman. On today's date, the market rates for such machines range from $2000 – $2500. On the other hand, a PlayStation game console is being sold at a mere $399. Using many PlayStations, we can set-up highly powerful clusters for about the same price as that of a single PC. Figure 6 depicts the relationship between the cost of deployment and the running times of the algorithms under consideration. A cluster of 7 PS3 nodes has about the same cost of deployment as a PC, and yet we achieve about a 6.3-fold speedup over the naive Smith-Waterman implementation. By bridging the gap between the execution times between Smith-Waterman and BLAST, together with effectively tapping the potential of inexpensive and fast hardware, we have provided a solution that tends towards the ideal – high speeds at affordable costs, together with ideal sensitivity, about which we discuss next.

Sensitivity: We measured the sensitivity of BLAST relative to the ideal Smith-Waterman algorithm by using the equations from Section 4. First, we used the output data from previous experiments to list the scores and E-values of each sequence alignment that was generated by BLAST and Smith-Waterman. In this experiment, we considered the scores of the alignments with E-value

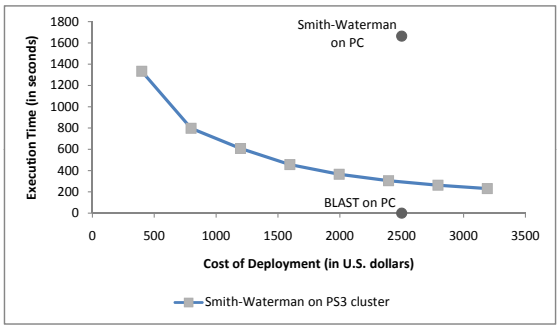
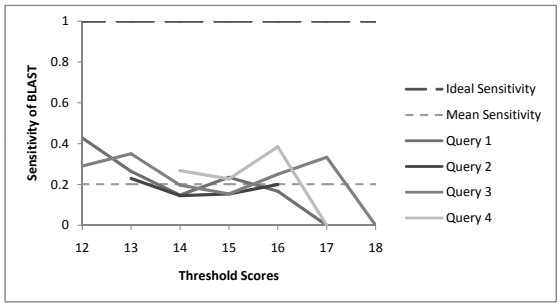


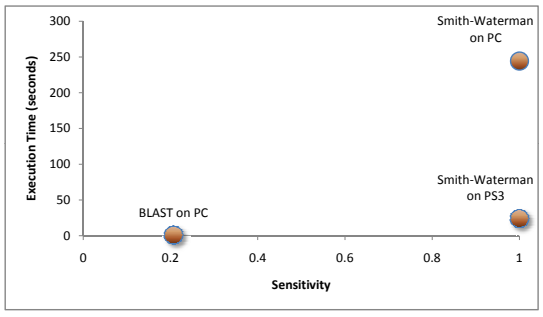
FIGURE 6: Execution time versus Cost of Deployment.

≤ 100 as *significant* threshold scores. By using Equation (3), we then calculated the sensitivity of BLAST for the different significant threshold scores and for the different query inputs, as shown in Figure 7 (a). From these sensitivity values, we found that the mean sensitivity value was 0.201 with a very low variance of 0.01 on an average, across all input queries. The consistent empirical results led us into formulating Equation (4) as a realistic estimate of the sensitivity for a given sequence search algorithm.

We used this sensitivity value of BLAST for further analysis. Figure 7 (b) explains the relationship between the sensitivity and execution time of a sequence search algorithm. It can be seen that although BLAST is faster than the Smith-Waterman implementations, its sensitivity value of 0.201 indicates that BLAST *misses* about 80% 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. The lower right part of the Figure 7 (b) is the *ideal* point for sequence search, i.e. high sensitivity and low execution time and should be the target for all future sequence search solutions.



(a)



(b)

FIGURE 7: (a) Sensitivity versus Threshold scores and (b) Sensitivity versus Execution Time.

6 Conclusions

This paper aims at solving the hardest computational problem that has plagued the Bio-Informatics community for many years – to execute the *optimal* sequence search algorithm *quickly* on *inexpensive* hardware. This motivation led us to define the term 'sensitivity' of homology search methods in a comprehensive manner. We used this definition to quantify the sensitivity of BLAST, and found that it misses 80% of the significant sequence alignments. It is now possible to evaluate the trade-offs between sequence search methods in a quantitative fashion.

Our efforts have then been directed to contribute novel schemes to design, implement and optimize the optimal local sequence alignment algorithm – Smith-Waterman to execute on the powerful Cell BE that drives the inexpensive PlayStation 3. We tested the scalability and efficiency of our design and optimization techniques by initially aligning individual sequence-pairs. We then imitated the execution of BLAST by performing a series of pairwise sequence alignments by comparing fixed sample query sequences against sequences that were fetched one-by-one from the Drosophila nucleotide database. We ran this algorithm across a cluster of 14 PlayStations, and compared the performances of the sequential and parallel flavors of Smith-Waterman against that of BLAST. Our solution is found to extract a ten-fold speed improvement over the naive Smith-Waterman implementation, at ideal sensitivity and low costs, which is ideal to resolve the homology search problem.

As future work, we intend to investigate the integration of the parallel Smith-Waterman for the Cell into sequence alignment tool-kits. Also, we intend to explore the challenges of mapping Smith-Waterman onto other powerful, yet inexpensive general purpose computing hardware like the GPGPU, thereby moving closer to providing the ideal solution in the sequence search domain.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[2] J. A. Kahle et al. Introduction to the Cell multiprocessor. In *IBM Journal of Research and Development*, pages 589–604, Jul-Sep 2005.

[3] J.; Heshan Lin; Xiaosong Ma Gardner, M.K.; Wu-chun Feng; Archuleta. 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*, pages 22–22, 11-17 Nov. 2006.

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

[5] D. Lavenier. Dedicated hardware for biological sequence comparison. 2(2):77–86, 1996.

[6] M. Li, B. Ma, D. Kisman, and J. Tromp. Patternhunter ii: Highly sensitive and fast homology search. 2003.

[7] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, March 1985.

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

[9] Tf Smith and Ms Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147:195–197.

[10] 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)*.

[11] Yoshiki Yamaguchi, Tsutomu Maruyama, and Akihiko Konagaya. High speed homology search with fpgas.