Fundamenta Informaticae 96 (2009) 181–194
DOI 10.3233/FI-2009-173
IOS Press

The new SIMD Implementation of the Smith-Waterman Algorithm on Cell Microprocessor

Witold R. Rudnicki*
Interdisciplinary Centre for Mathematical and Computational Modelling
University of Warsaw, Pawińskiego 5A, 02-106 Warszawa, Poland
W.Rudnicki@icm.edu.pl

Aleksander Jankowski
ajank@students.mimuw.edu.pl

Aleksander Modzelewski
aleander@shirk.pl

Aleksander Piotrowski
ap219542@students.mimuw.edu.pl

Adam Zadrożny
Adam.Zadrozny@gmail.com

Abstract. Algorithms for estimating similarity between two macromolecular sequences are of profound importance for molecular biology. The standard methods utilize so-called primary structure, that is a string of characters denoting the sequence of monomers in hetero-polymer. These methods find the substrings of maximal similarity, as defined by the so-called similarity matrix, for a pair of two molecules. The problem is solved either by the exact dynamic programming method, or by approximate heuristic methods.

The approximate algorithms are almost two orders of magnitude faster in comparison with the standard version of the exact Smith-Waterman algorithm, when executed on the same hardware, hence the exact algorithm is relatively rarely used. Recently a very efficient implementation of Smith-Waterman algorithm utilizing SIMD extensions to the standard instruction set reduced the speed

*Address for correspondence: Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw, Pawińskiego 5A, 02-106 Warszawa, Poland

advantage of heuristic algorithms to factor of three. Here we present an improved implementation of the Smith-Waterman algorithm on the Cell processor.

Implementation presented here achieves execution speed of approximately 9 GCUPS. The performance is independent on the scoring system. It is 4 to 10 times faster than best Smith-Waterman implementation running on a PC and 1.5 to 3 times faster than the same implementation running on Sony PlayStation 3. It is also 5 times faster than the recent implementation of the Smith-Waterman utilizing Nvidia GPU.

Our implementation running on Sony PlayStation 3 has performance which is directly comparable with that of BLAST running on PC, being up to 4 times faster in the best case and no more than two times slower in the worst case. This performance level opens possibility for using the exact Smith-Waterman algorithm in applications, where currently approximate algorithms are used.

Keywords: bioinformatics, protein sequence alignment, Cell processor

1. Introduction

Era of modern molecular biology started with the discovery of the DNA structure [1] and subsequent determination of the genetic code [2, 3, 4]. Since then determining sequences of nucleic acids and protein molecules became on of the most important tasks of molecular biologists. It appeared very soon that protein molecules performing similar tasks are often very similar, not only when two proteins come from the same organism, such as haemoglobin and myoglobin, but when they come form different species as well, as for example bovine and human variants of haemoglobin. The sequence similarity between various protein molecules is a very important information, which is used for identification of unknown proteins, establishing the evolutionary relations between species and in many other ways.

The similarity of two molecules is measured by aligning them and counting number of identical or chemically similar residues which occupy the same position. It is an easy task for nearly identical molecules. Unfortunately, with decreasing similarity level, the number of alternative alignments is growing very quickly and problem becomes impossible to solve by hand.

Additional complication arises, due to chemical differences between amino acids. Different mutations occur with different probabilities, which are dependent on the chemical similarity between amino acids. One should take this into account when comparing various alternative alignments. Evolutionary similarity between all pairs of amino acids is represented in the form of so-called amino acid similarity matrix [5]. The algorithm, developed in 1970 by Needleman and Wunsch [6], utilizes such a matrix to find an optimal global alignment for a pair of protein sequences. It is a dynamic programming algorithm, which finds the optimal path through the rectangular matrix, which is filled with similarity scores for pairs of amino acids. Needleman-Wunsch algorithm returns biologically meaningful results for relatively similar sequences of similar length.

Nevertheless, this algorithm is unsuitable for alignment of sequences, where only fragment of sequences are similar. This situation can arise for example, when one compares two multi-domain proteins, sharing single similar domain. Also comparison of two highly diverged proteins may be meaningful only within the relatively short, highly conserved region. The modification of the Needleman-Wunsch algorithm, which finds optimal local alignment was proposed in 1981 by Smith and Waterman [7]. It was followed by an extension which allowed for using affine gap penalties and computation time proportional to the product of sequences' lengths proposed by Gotoh in 1982 [8]. These algorithms, although

exact, are relatively slow and rarely used in practical applications, because fast heuristic approximate algorithms, such as BLAST [9] or FASTA [10], give good enough results with much lower computational costs.

On the other hand, the structure of the Smith-Waterman algorithm allows parallelisation of the computations on the suitable hardware and significant performance gains. Several implementations taking advantage of SIMD instructions were proposed [11, 12, 13], the latest being almost as effective as the heuristic algorithms [14]. Indeed, the implementation presented by Farrar is only 4-7 times slower than BLAST on the same PC when executed using BLOSUM 62 matrix and standard gap penalties. Differences are even smaller for other scoring systems.

Recently a new microprocessor architecture devoted to high throughput parallel computations, called Cell Broadband Architecture, has been presented jointly by IBM, Toshiba and Sony. It has been demonstrated that this architecture has a high chance to become a serious alternative to the standard PC in scientific computations, because of high performance of integer and floating point operations for certain types of codes [15]. The theoretical peak performance of the Cell processor is 205 GFLOPs per second in single precision floating point calculations, which can be compared with the theoretical peak performance of roughly 10 GFLOPs per second for the current dual core general purpose microprocessors. Also, since Cell processors are used in mass-market-low-cost product, such as Sony PlayStation 3 (PS3), it gives this architecture advantage of being a relatively low cost, high volume product. Implementing of the basic scientific algorithms into this architecture may result in a boost of performance by an order of magnitude in comparison to a standard PC.

2. Implementation

2.1. Algorithm

The Smith-Waterman algorithm, with Gotoh extensions for handling affine gap penalties was implemented. Two sequences to be aligned, a query sequence and database sequence, are defined as $Q = \langle q_1, q_2, q_3, \ldots, q_m \rangle$ and $D = \langle d_1, d_2, d_3, \ldots, d_n \rangle$. The length of query sequence is m and database sequence is n, respectively. A scoring matrix W is used, where $W_{q,d}$ is a score for amino acids q and d. For similar amino acids q and d, $W_{q,d} \leq 0$, for dissimilar ones $W_{q,d} < 0$. The penalties for opening and continuation of the gap are defined as G_{init} and G_{ext}, respectively. The optimal alignment is obtained by processing a rectangular matrix H. Initially element H_{ij} holds score of for the pair i-th amino acid of the query sequence and j-th amino acid from the target sequence. After processing by the algorithm, H_{ij} holds the value of the best local alignment, ending at i and j. The matrix is processed, starting from upper left corner. Each element is obtained as a result of the following comparison:

$$H_{ij} = \max \left\{ \begin{array}{l} H_{i-1,j-1} + W_{q_i,d_j} \\ E_{ij} \\ F_{ij} \\ 0 \end{array} \right\} \tag{1}$$

where

$E_{ij} = \max \left\{ \begin{array}{l} E_{i,j-1} - G_{ext} \\ E_{i,j-1} - G_{init} \end{array} \right\}$ is the score arising from the gap introduced at the query,

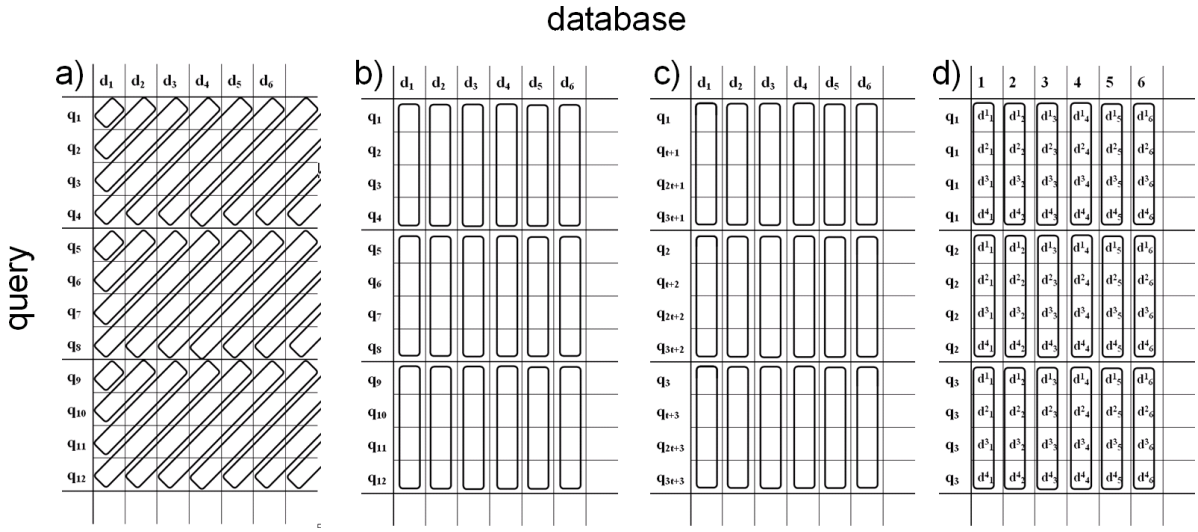

Figure 1. Data alignment in SIMD implementations of the Smith-Waterman algorithm.

Vectors aligned along minor diagonals (a), vectors aligned along columns in a straightforward- (b) and striped- (c) way as well as vectors containing entries from different matrices (d).

$F_{ij} = \max \left\{ \begin{array}{l} F_{i-1,j} - G_{ext} \\ F_{i-1,j} - G_{init} \end{array} \right\}$ is the score arising from the gap introduced at the target sequence and the first argument is the score arising from the ungapped alignment. The values of E_{ij} and F_{ij} are set to zero if $i = 0$ or $j = 0$.

2.2. Existing Solutions

In the SIMD single operation is performed on multiple data, for example by forming a sixteen-element byte vector and storing it in a 128-bit quad word. Each operation on a 128-bit quad word becomes then an operation on sixteen independent variables. This allows in theory to increase the speed of computations sixteen-fold. The SIMD operations for Smith-Waterman algorithm were previously used by Alpern [11], Wozniak [12], Rognes and Seeberg [13] , and Farrar [14]. As can be seen in the Formula 1, the value of each cell in the dynamic programming matrix depends on the values which are above and to the left of the cell. In parallel implementation cells which are processed in parallel should be independent.

There are three general methods for this. The first one is based on observation that values on the minor diagonals of the matrix are independent of each other and can be processed in parallel. The second one uses observation that the algorithm is usually used for scanning large databases of sequences, which are not similar to the query. One may notice that if a query is not similar to a database sequence, then the scores in the dynamic programming matrix are low and with sufficiently high gap opening penalty G_{init} E_{ij} and F_{ij} in Eq. 1 are equal zero for most of the cells. One can use this observation by temporarily assuming that $F_{ij} = 0$ when processing the column of the matrix. With such assumption the value in the cell does not depend on the values in the cells above the current one and computations of several cells

within the column can be performed in parallel. Then one has to check if the assumption that $F_{ij} = 0$ was correct. If it was not, one needs to recompute affected cells, however, if the sequences are not similar this happens relatively infrequently. The third approach is based on the observation that when the SW algorithm is used for database scan one can align query with several database sequences in parallel. The graphical representation of the data layout in these approaches is given in Figure 1.

The first approach has been proposed by Wozniak. In this implementation the four-element vector was aligned with minor diagonal of the matrix H. Unfortunately the code for loading the substitution scores to the vector is rather complicated and not very efficient, because each substitution score is indexed with different pair of amino acids from database and query sequence for each element of the vector.

The second approach was proposed by Rognes and Seeberg, and recently optimized version was implemented by Farrar. In the original algorithm of Rognes and Seeberg processing is performed for vector parallel to query, consisting of 8 adjacent cells of the same column, see Figure 1b. First is checked if $F_{ij} = 0$ for all cells. It is possible to check this condition for all elements of the vector simultaneously. If this condition is met, the vector version of the algorithm is called, otherwise the scalar version with all data dependencies is used. If sequences are not similar the scalar version is called relatively infrequently. The main improvement with comparison to Wozniak approach is achieved due to more efficient retrieval of scores for cells beeing processed. In the Wozniak approach each cell represents alignment of different pair of amino acids. Therefore one needs to retrieve this score from a score matrix and place it in the right position of the score separately for each element of the vector. On the other hand one may note that all scores A_{ij} are for the same $j - th$ amino acid of the database sequence. One can take advantage of this observation, by precomputing so called query profile. The query profile is a matrix 20 $\times$ query length. Each column holds all scores for single amino acid when paired with the query. Loading of all scores for the single vector can be executed with one vector copy operation, which is much more effective than in Wozniak approach.

Graphic representation of the construction of the query profile and the score matrix in Rognes and Farrar approach is displayed in the Figure 2. The next improvement of the SIMD implementation of the SW algorithm was introduced recently by Farrar. Also in this case vectors are parallel to the query sequence, but here the data dependence within the vectors is avoided by using stripped layout of the dynamic programming matrix. Single vector holds matrix elements from the same column, however, they are not adjacent, see Figure 1c. Let us assume that vector holds four values and query length is $4t$.

Then vectors are constructed as:

$$\begin{aligned} V_1 &= \langle H_{i,1}, H_{i,t+1}, H_{i,2t+1}, H_{i,3t+1} \rangle , \\ V_2 &= \langle H_{i,2}, H_{i,t+2}, H_{i,2t+2}, H_{i,3t+2} \rangle , \\ &\vdots \\ V_k &= \langle H_{i,k}, H_{i,t+k}, H_{i,2t+k}, H_{i,3t+k} \rangle . \end{aligned}$$

Following Rognes and Seeberg, Farrar assumes that in most cases computations of the H-value do not depend on the values of the cells above. This assumption is used to perform computations for the first vector. It is not necessary when processing the following vectors, since values of the cells located above current one have already been computed in the previous step. Finally, after the whole column is processed, it is checked whether the cells computed in the first step indeed do not depend on the cells above. If this assumption is false, then the computations for the first vector are repeated and it is checked,

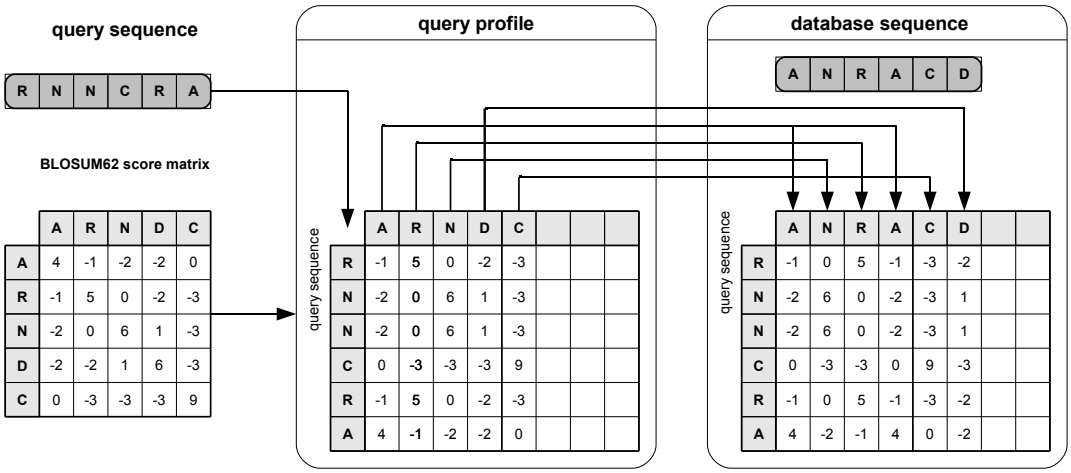

Figure 2. Construction of the query profile from the similarity matrix and application of the query profile for construction of the dynamic programming matrix. For clarity the reduced set of five amino acids is used. Also vector length is reduced to four elements.

whether this influences results for the second vector. This correction step is repeated for further vectors until the results of the current vector do not influence results for the next vector. One should note, that query profile introduced in Rognes algorithm is used by Farrar, albeit in a stripped manner. Graphic representation of the construction of the query profile and the score matrix in Rognes and Farrar approach is displayed in the Figure 2.

Farrar approach improves Rognes and Seeberg algorithm in two ways. In the Rognes-Seeberg version of the algorithm the first element of the currently computed vector depends on the last element of the previous vector and this forces usage of inefficient operations for manipulating single elements of vectors. By organizing computations in the striped manner these non-aligned value dependencies between vectors are avoided. The striped approach removes also direct dependencies between elements of a single vector. The cost of avoiding these data dependencies is relatively small. In the best case, when no data dependencies are discovered between the last and the first vector one needs to perform a single run of the correction step per column. On average only very few correction steps are required. Nevertheless these correction steps require additional operations, which are not easy to handle efficiently in the vectorised manner. Additionally Farrar implementation uses one-byte integers. This allows for processing 16 element vectors, instead of 8 elements as in earlier approach. The score range (0-255) is sufficient for the database scan with commonly used scoring systems. In such application the overflow can happen only when relatively similar sequences are aligned - and such situation is recorded as a hit, which needs to be aligned with another version of Smith Waterman algorithm.

2.3. W4A Implementation

The Cell processor is a multi-core design, with one core being the general purpose processor, called PPE, and eight cores, called Synergistic Processing Elements (SPE), which are specialized for high

performance SIMD floating point and integer computations (Only six SPE units are available for users in the Sony Playstation 3). Each SPE unit can run a different code. The single SPE unit performs computations using 128 bit words. The internal processor instructions allow to use these 16 byte words as 16 bytes, 8 integers, 4 floating point single precision numbers, or 2 double precision floating point numbers.

The W4A implementation is designed to take advantage of the Cell processor architecture. The main program, which reads data and schedules computations resides on the PPE core. The vectorized SW alignment routines are executed in parallel on SPE cores, each core processes different sequences. The vectorization is based on the approach proposed originally by Alpern. Each thread runs sixteen parallel searches with sixteen different database sequences within single pass of the algorithm. There are sixteen independent matrices which are processed simultaneously and each vector holds one element from each matrix.

There are two reasons why this design of the algorithm should improve performance in comparison with that of the previous implementations. First, there are no dependencies between elements of the same vector, and therefore neither the correction step of the Farrar algorithm, nor checking of independence condition of the Rognes and Seeberg algorithm, are needed. Second, the additional performance gain over the previous SIMD implementation arises due to more efficient usage of the vector variables in our approach. In the previous implementations the effective query length was a multiple of the vector size. For queries which were not multiples of the vector size this results in processing of the unoccupied vector elements at the last section of the query. In the current approach all vector elements are used, since the query is matched with N database sequences of identical length. Also this approach is independent of the scoring function, whereas performance of both Rognes and Seeberg as well as Farrar approach strongly dependends on the gap penalties. Nevertheless, the simplification of the computations comes with a cost, namely the computation of the alignment score for sixteen sequences at once is more complicated than in the previous implementations.

In our approach each score vector holds scores for a single query residue and N database residues. In principle any combination of these N residues is possible, making pre-computing of the query profile in the analogous way impossible. Nevertheless, we do use the pre-computed query profile, but in a different manner. The score vector is composed by performing an efficient lookup of the single row of the query profile, see Figure 3. The score vector is composed using `shuffle` command, which is specific to Cell and Power processors. This useful command allows us to load any byte from two source vectors to the single result vector using mask stored in the third vector. This allows very efficient lookup of the query profile. Two source vectors hold a row of the query profile. The mask vector is constructed from the sixteen amino acids located at the current position of the database sequences.

The layout of the dynamic programming matrix described above results in a significant optimisation in comparison with the Farrar implementation since correction phase of the algorithm is not necessary. Also the execution time is independent of gap penalties. Figure 4 illustrates the order of computations and memory layout in three implementations discussed above. Looking at Figure 4 it is easy to understand that an efficient application of our implementation requires sorting of the database according to the length of the sequences. Due to sorting in most cases all sequences computed within a single algorithm run have identical length. Therefore, the dynamic programming matrix is filled with real data and no waste of the processor cycles is incurred. The sorting of the database is performed after each update.

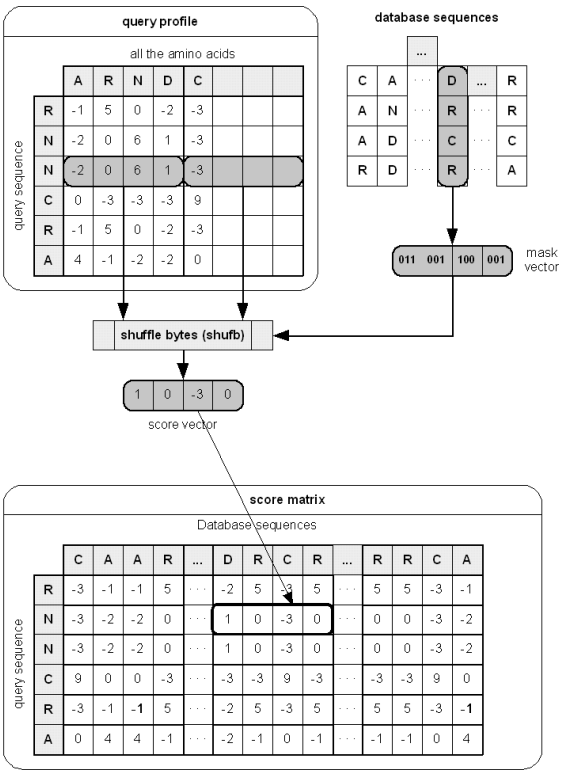

Figure 3. Construction of the dynamic programming matrix from query profile using shuffle command.

`Shuffle` command uses four vectors of memory, each consisting of N bytes. Vectors 0 and 1 hold the source data, vector 2 holds control data and vector 3 is output. The k-th byte of the vector 2 holds the number of the byte from vectors 0 and 1, which should be put in the k-th byte of the output vector. Let's assume that algorithm updates scores for the i-th position of the query sequence and j-th position of the database sequences. Then the vectors 0 and 1 hold an i-th row of the query profile, vector 2 holds addresses corresponding to N amino acids from the j-th position in N database sequences. The k-th byte of the output vector holds a $W(q_i, d_j)$ for the k-th database sequence. The output vector is copied to the score matrix.

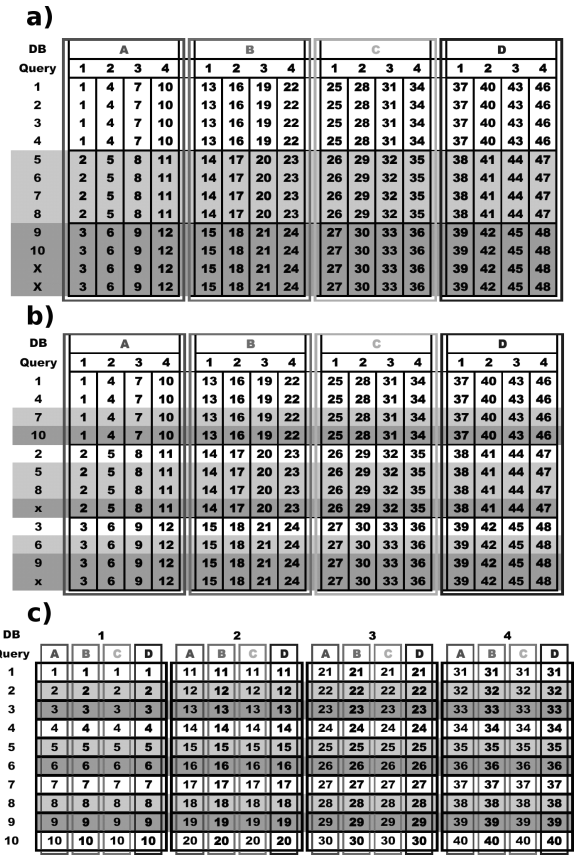

Figure 4. Processing order for a single query sequence in three implementations of the Smith-Waterman algorithm.

Rognes a), Farrar b) and W4A c).

Elements of the dynamic programming matrix corresponding to the same database sequence are enclosed in a box of the same colour. Rognes and Farrar implementations use vectors parallel to the query and perpendicular to the database sequence, whereas our implementation uses vectors which are perpendicular both to the query and database sequences. It can be observed that a number of operations necessary to process N database sequences is larger in two former implementations, unless the length of a query sequence is an exact multiple of the vector length.

3. Testing

To compare performance of current implementation with earlier SW implementations and other algorithms for sequence similarity search we developed a standardized environment. All tests were performed using ASTRAL SCOP Genetic Domain Sequences database [16], rel. 1.71[1], containing 71796 protein sequences and 13740191 residues. Average sequence length is 191. The Astral database used in the tests was selected because it fits in the memory of the Sony Playstation3, whereas, for example, NCBI NR database does not.

Forty sequences from the NCBI NR database with varying length were used as query sequences. The shortest and the longest sequences were respectively 42 and 1202 residues long. Tests were performed using standard similarity matrices used with BLAST at NCBI interface, for all combinations of parameters. These include two matrices of the PAM family [5] (PAM 30 and PAM 70) as well as three matrices of the BLOSUM family [17] (BLOSUM 80, BLOSUM 62 and BLOSUM 45).

All tests were performed on Sony Playstation 3 running Fedora 7 and on a standard desktop PC with the cost comparable to that of PS3. PC was equipped with AMD Athlon(tm) 64 Processor 3200+, clocked at 2 GHz with 512 KB of cache and 1 GB of RAM. PC was running Centos 5.0. For all codes the total time of the core sequence alignment phase was measured, without time necessary for reading data from the disk. All tests were performed fifteen times for each query, and the best result for each query was used in comparisons. Tests were performed for all standard gap penalty parameters used for BLAST at NCBI.

4. Results and Discussion

The execution times for BLAST, Farrar's version of Smith-Waterman implemented on PC, Farrar's version implemented on Sony PS3 and our version implemented on Sony PS3, which here is called W4A, are shown in Figure 5.

The numbers used for comparisons are taken from runs performed using the gap penalty parameters resulting in minimal gap opening penalty for given matrix. These values usually gave results closest to the average of all standard combinations of parameters. The differences between execution times for the same algorithm run with identical similarity matrix and various gap penalty parameter sets for W4A algorithm were negligible in comparison to the differences between runs with different similarity matrices. For BLAST algorithm the differences varied between 0 and 13% and for Farrar algorithm between 3% and 27%. It is interesting to note that the highest differences for Farrar algorithm were observed for short sequences, whereas for BLAST for long sequences.

The performance of all algorithms measured in Cell Updates Per Second (CUPS) is shown in Figure 6. One can notice that the performance of all algorithms varies with the query length and scoring system, see also Table 1. Dependence of performance on the query length is visible for all algorithms and all scoring systems. The smallest variability is observed for W4A implementation, where the highest performance is 25% larger than the lowest one for all scoring systems. The largest variability is for BLAST where the highest performance is 4 times better than the lowest for the same scoring system and 6.7 times when performance for all scoring systems is compared.

[1] http://astral.berkeley.edu/scopseq-1.71.html

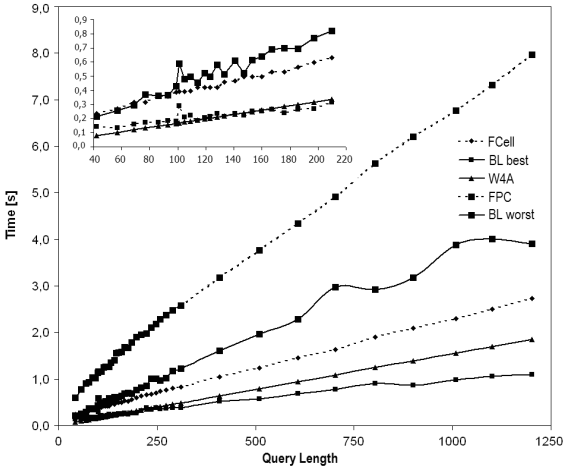

Figure 5. Execution times for tested algorithms as a function of query length.

Test with the best (BLOSUM 62) and the worst (BLOSUM 45) performing scoring system are presented for BLAST. Data for short queries is shown in inset.

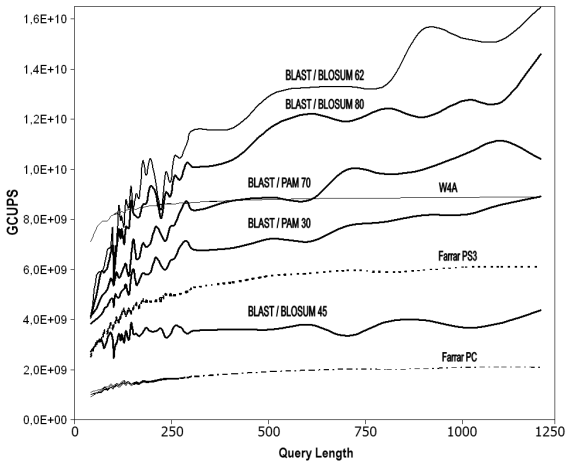

Figure 6. Performance of tested algorithms as a function of query length.

Results for all scoring systems are displayed. Results for different scoring systems are indistinguishable for all implementations of Smith-Waterman algorithm.

Also the variability between scoring systems is significant for BLAST. The execution time of BLAST for the BLOSUM45 matrix is on average more than 3 times higher than that for BLOSUM62 matrix. on the other hand performance of both implementations of the Smith-Waterman algorithm is much less dependent on the scoring system. For Farrar implementation the difference is less than 5% or 2% on PC and PS3, respectively. The difference between the fastest and the slowest run for W4A implementation is 0.02%, which is negligible. This can be seen in Figure 6, where performance curves for BLAST algorithm for different scoring systems are well separated, whereas for all implementations of Smith-Waterman algorithm curves for various scoring systems are almost indistinguishable.

One can notice that for short queries (those of length less than 150 residues) performance of our implementation is better than that of any other algorithm, including BLAST. Then for query lengths

Table 1. Variability of performance of the tested algorithms.

	PAM 30	PAM 70	BLOSUM 80	BLOSUM 62	BLOSUM 45	Overall
BLAST	3.57	4.06	1.61	2.52	2.32	6.69
Farrar PC	2.09	2.07	1.91	2.30	2.11	2.31
Farrar PS3	2.40	2.40	2.35	2.44	2.41	2.45
W4A	1.25	1.25	1.25	1.25	1.25	1.25

Variability is defined as the ratio of the best and the worst performance achieved for given scoring system. Overall variability is defined as the ratio of the best to the worst performance for all scoring systems.

Table 2. Relative performance of W4A versus other tested algorithms.

	PAM 30		PAM70		BLOSUM 80		BLOSUM 62		BLOSUM 45	
	best	worst	best	worst	best	worst	best	worst	best	worst
BLAST	1.75	0.80	2.06	1.00	1.74	0.61	1.78	0.54	3.79	2.04
Farrar PS3	3.07	1.46	2.95	1.46	2.95	1.46	2.95	1.46	2.86	1.46
Farrar PC	9.85	4.23	8.99	4.22	8.74	4.21	8.53	4.23	7.47	4.22

Numbers are higher than 1 if the performance of W4A is better than that of the other algorithm.

up to 300 residues the performance of our algorithm is still comparable to that of BLAST with best performing scoring systems. One may notice, that for two scoring systems performance of BLAST is lower than that of our algorithm for all query lengths.

Comparison with the previous implementations of the Smith-Waterman algorithm is even more favourable for this implementation. The speedup in comparison with Farrar implementation on PS3 varies between 1.5 and 3.1, with highest values for short queries. The speedup with respect to PC implementation of Farrar algorithm varies between 4.2 and 9.8, again the highest value are for short queries. The summary of the relative speeds for all scoring systems is presented in Table 2.

To analyse the algorithm performance more precisely, one should consider how much the results depend on the selection of the database. Both theoretical analysis and practical tests (data not shown) shows that the performance of the two tested variants of the Smith-Waterman algorithm does not depend on the length of the sequence database. Therefore this choice of test database does not influence the results of comparison between variants of Smith-Waterman algorithm.

The comparison between Smith-Waterman and BLAST is more complicated, because relative performance may depend to larger extent on the percentage of hits encountered on the database. It can be seen that the performance of BLAST varies in our tests more than that of Smith-Waterman variants (see Figures 5 and 6). Therefore the exact values of relative performance of two algorithms should be taken with the grain of salt. Nevertheless, the performance of BLAST run against ASTRAL using the longest queries is very similar to that of BLAST run against NCBI NR database on PC (data not shown) and therefore it is a good representation of BLAST performance. In practice one can expect that relative performance of these algorithms for large databases would be close to that reported as worst relative performance in Table 2.

We presented a new implementation of the SW algorithm using SIMD instructions of the Cell architecture. This implementation is up to 10 times faster than implementation presented by Farrar on PC and up to three times faster than Farrar implementation on PS3. It is also faster than the PC implementation of the BLAST algorithm for some scoring systems. Taking into account that the cost of a Sony PS3 is comparable to that of a PC, the application of exact algorithm instead of approximate heuristic becomes feasible, time effective and cost-effective.

The improvement in performance in comparison to Farrar version is due to an efficient vectorisation of the code, in our implementation. The code of the inner loop is simple since no tests for the independence of vector elements are necessary. Also the additional correcting loop is omitted. This makes the code easier to optimize. Thanks to the true independence of computations between subsequent cells

the additional optimisation is possible in our approach, what is impossible for Farrar method. Cell processor can issue new instruction in every cycle, but for most instructions execution takes at least two cycles. Therefore for algorithms requiring strict order of execution the processor needs to wait for the result of the previous step. Simple reordering of operations within the inner loop is not sufficient to overcome this limit in the case of Smith-Waterman algorithm. Nevertheless, thanks to the real independence of the results between steps of the algorithm one can intertwine computations for two cells located on the same minor diagonal of the matrix. In this way processor is never waiting for instructions.

The algorithm described above is responsible for the first phase of the homology search procedure – finding the hits. This is the most compute-intensive phase and it is performed on SPE cores of the Cell processor. To reduce computational cost and memory requirement the algorithm does not store the traceback information. The result of the procedure is limited to the value of the highest score achieved. The actual sequence alignment is performed by another variant of Smith-Waterman algorithm, which stores traceback information. Since finding the hit is in fact a rare event the cost for computing a full version of a Smith-Waterman algorithm for the hits is more than offset by performance improvements during the hit finding phase. The sequence alignment is performed on the PPE unit of the Cell processor.

There are certain practical limitations of the implementation presented here, which need to be addressed before using it on the production server for the protein homology searches. These limitations arise due to the constraints of both PS3 as well as Cell architectures.

The local memory of the SPE unit of the Cell processor is 256 KB, what puts a constraint for the size of the protein which can be aligned in the single run. Within the current implementation the sequence length is limited to 3000 residues for database sequences. As of September 2007 there are more than 3.6 million protein sequences in the NCBI nr database, and 6318 (0.18%) are longer than this limit. Total length of these long sequences is 2.11% of the total database length. Alignment of these sequences is performed by the Farrar algorithm.

Practical applications of the algorithm could be limited by the constraints of Sony PlayStation 3 design as well. In particular the main memory of the PS3 is limited to 256 MB, what is not sufficient to hold many current sequence databases, including for example NCBI NR database. To overcome this limit we are developing the mixed PC/SP3 cluster system consisting of a single PC node and several PS3 nodes. Each PS3 node is connected to a separate network interface on a PC. PC is a control node which holds sequence databases and user interface. It feeds the execution nodes with the sequences and collects the hits. PS3 nodes perform the computations.

Currently our implementation is limited to the Cell BE architecture. The key element of our implementation is computation of the query profile for multiple database sequences using single Cell's shuffle instruction. This instruction is specific to the Cell architecture. Nevertheless it might be possible to adapt this approach to other processors using different instructions. This issue remains to be investigated.

5. Conclusions

In the current paper we present the implementation of the Smith-Waterman algorithm on the Sony PS3 variant of the Cell architecture. This implementation takes advantage of the parallel internal architecture and SIMD instruction set. The SIMD registers of the SPE units are parallel to the query sequence in the striped pattern, each strip coming from different database sequence. In this way we process 96 database sequences in a single execution of the SW algorithm and completely avoid data dependen-

cies between parallel registers. This allows us to take full advantage of the parallel SIMD architecture of the system.

Calculation speeds approaching 9 GCUPS are achieved, which is a speedup of a factor 4.5 over Farrar implementation on PC and comparable to that of BLAST on PC. The relative speed comparisons were performed using single core CPU of the PC and six SPEs of the Cell processor. Using multiple cores of PC processor would have shifted balance towards PC. on the other hand, we used only 6 out of 8 SPEs of the Cell processor (due to SP3 limitations). Also we assume, that compilers for Cell platform are less mature than those for x86 platform and therefore additional performance gains on Cell are possible. Taking all this factors into account we believe that comparison is fair.

The speed of current implementation of the Smith-Waterman algorithm is comparable to the implementations performed on FPGA, see for example [18, 19, 20, 21, 22] estimate that the execution speeds up to 23.8 GCUPS are achievable on the high end FPGA systems. Recently the Smith-Waterman implementation on the Nvidia GPU was presented, which achieves calculation speed of 1.8 GCUPS or 3.5 GCUPS using single or dual GPU configuration, respectively [23].

According to our best knowledge algorithm presented here is currently the fastest implementation of the Smith-Waterman algorithm on the commodity hardware.

5.1. Availability and Requirements

The code for the W4A-SW algorithm as well as for the cluster system is available at
http://bioinfo.icm.edu.pl/algorithm/source/W4A-SW/
Operating systems: Linux
Programming language: C/C++
Other requirements: IBM's Cell SDK
License: GPL

Acknowledgements

This project was partially supported by ICM grant 501-64-13-BST1345. Computations were performed at ICM, grant G34-5.

References

[1] Watson, J. D., Crick, F. H. C.: A Structure for Deoxyribose Nucleic Acid, *Nature*, **171**, 1953, 737–738.

[2] Crick, F. H., Barnett, L., Brenner, S., Watts-Tobin, R. J.: General nature of the genetic code for proteins, *Nature*, **192**, 1961, 1227–1232.

[3] Nirenberg, M. W., Matthaei, J. H.: The Dependence of Cell-Free Protein Synthesis in E. coli upon Naturally Occurring or Synthetic Polyribonucleotides, *Proceedings of the National Academy of Sciences*, **47**, 1961, 1588–1602.

[4] Söll, D., Ohtsuka, E., Jones, D. S., Lohrmann, R., Hayatsu, H., Nishimura, S., Khorana, H. G.: Studies on polynucleotides, XLIX. Stimulation of the binding of aminoacyl-sRNA's to ribosomes by ribotrinucleotides and a survey of codon assignments for 20 amino acids, *Proceedings of the National Academy of Sciences*, **54**, 1965, 1378–85.

[5] Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C.: A model of evolutionary change in proteins. In: *Atlas of Protein Sequence and Structure*. Edited by Dayhoff, M.O., vol. 5, National Biomedical Research Foundation, 1978.

[6] Needleman, S. B., Wunsch, C. D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins, *Journal of Molecular Biology*, **48**, 1970, 443–53.

[7] Smith, T. F., Waterman, M. S.: Identification of common molecular subsequences, *Journal of Molecular Biology*, **147**, 1981, 195–197.

[8] Gotoh, O.: An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, **162**, 1982, 705–708.

[9] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., Lipman, D. J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, *Nucleic Acids Research*, **25**, 1997, 3389–3402.

[10] Pearson, W. R., Lipman, D. J.: Improved Tools for Biological Sequence Comparison, *Proceedings of the National Academy of Sciences*, **85**, 1988, 2444–2448.

[11] Alpern, B., Carter, L., Gatlin, K. S.: Microparallelism and High-Performance Protein Matching. In: *ACM/IEEE Supercomputing Conference*, San Diego, California, 1995.

[12] Wozniak, A.: Using video-oriented instructions to speed up sequence comparison, *Comput. Appl. Biosci.*, **13**, 1997, 145–150.

[13] Rognes, T., Seeberg, E.: Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors, *Bioinformatics* (Oxford, England), **16**, 2000, 699–706.

[14] Farrar, M.: Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* (Oxford, England), **23**, 2007, 156–161.

[15] Samuel, W., John, S., Leonid, O., Shoahib, K., Parry, H., Katherine, Y.: The potential of the Cell processor for scientific computing. In: *3rd Conference on Computing Frontiers*, Ischia, Italy, ACM Press, 2006, 9-20.

[16] Brenner, S. E., Koehl, P., Levitt, M.: The ASTRAL compendium for protein structure and sequence analysis, *Nucleic Acids Research*, **28**, 2000, 254–256.

[17] Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks, *Proceedings of the National Academy of Sciences*, **89**, 1992, 10915–10919.

[18] Dydel, S., Bała, P.: Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices. In: *Field Programmable Logic and Application*. Edited by Becker, J., Platzner, M., Vernalde, S., vol. 3203. Berlin / Heidelberg, Springer, 2004, 23–32.

[19] Oliver, T., Schmidt, B., Maskell, D.: Hyper customized processors for biosequence database scanning on FPGAs. In: *ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, Monterey, USA, ACM, 2005, 229–237.

[20] Benkrid, K., Liu, Y., Benkrid, A.: High Performance Biosequence Database Scanning using FPGAs. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2007*, Honolulu, USA, IEEE, 2007, pp. I-361–I-364.

[21] Van Court, T., Herbordt, M. C.: Families of FPGA-Based Algorithms for Approximate String Matching. In: *Application-Specific Systems, Architectures, and Processors, ASAP'04*, Galveston, USA, 2004, 354–364.

[22] Li, I. T., Shum, W., Truong, K.: 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). *BMC Bioinformatics*, **8**:185, 2007.

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