Sequence Analysis

Optimizing Smith-Waterman for the Cell Broadband Engine

Michael S. Farrar

ABSTRACT

Motivation: As new processors become available, the Single-Instruction Multiple-Data Smith-Waterman implementations need to be adapted to the processors instruction set to get maximum performance. One recent processor, the Cell Broadband Engine has eight independent vector processors. To take advantage of the Cell's vector engines, the implementation needs to take into account the limited resources of the vector engine and the limits of the instruction set.

Results: The adapted Smith-Waterman implementation running on a single 3.2 GHz Cell Broadband Engine achieved speeds of >16 billion cell update per second with the ability to handle sequences of 32K residues.

Availability: http://farrar.michael.googlepages.com/striped.tgz

Contact: farrar.michael@gmail.com

1 INTRODUCTION

The Smith–Waterman (Smith and Waterman, 1981) algorithm is one of the slowest sequence search algorithms but the only one guaranteed to return the optimal score. As the size of the GenBank/EMBL/DDBJ double every 15 months (Benson *et al*., 2000), faster implementations of the Smith–Waterman algorithm have been developed using Single-Instruction Multiple-Data (SIMD) microprocessors to speed the calculations. A SIMD instruction is able to perform the same operation on multiple pieces of data in parallel.

The Cell Broadband Engine (B.E.) is a heterogeneous, multi-core processor optimized for compute-intensive workloads (IBM Handbook, 2007). The two main processing components of the Cell B.E. are the 64-bit PowerPC core (PPE) and eight specialized SIMD co-processors called Synergistic Processing Elements (SPE).

The main processing engine of the SPE is the Synergistic Processing Unit (SPU). The SPU is a SIMD processor with 128 128-bit registers and 256KB of memory referred to as Local Store (LS). The LS is used to hold both the instructions and data of the program to execute. Since the SPU cannot directly access main memory, DMA transfers are used to copy data to and from main memory and the LS. The DMA transfers are entirely controlled by software and are independent of the programs execution.

Three of the more common SIMD Smith–Waterman implementations are the Wozniak (1997), the Rognes and Seeberg (2000) and the Striped (Farrar, 2007). The implementations differ in how the data is accessed for the calculations. The Wozniak algorithm accesses the data values parallel to the minor diagonal. The Rognes implementation accesses the data parallel to the query sequence. The Striped algorithm, like the Rognes, accesses the data parallel to the query sequence, but in a striped pattern.

One of the first Smith–Waterman implementations running on the Cell B.E. was a port of SSEARCH34 (Pearson and Lipman, 1988). Erik Lindhal's Altivec SIMD version, a Wozniak implementation, was the starting point for the Cell B.E. port (Sachdeva *et al*., 2007). The Cell B.E. port uses half word values, 16 bits, when doing the calculations. Half words are the smallest elements supported by the Cell B.E. instruction set. To generate the weight vector, a Position Specific Scoring Matrix (Gribskov *et al*., 1987) (PSSM) is created based on the query sequence and the scoring matrix. The PSSM, *H* and *F* buffers requires 50 bytes per query residue. With the code and data buffers the largest sequence that can be processed with this implementation are 2,000 residues in length.

This paper presents the Striped Smith–Waterman implementation optimized for the Cell B.E. This optimized implementation improves search speeds 3 times over the Sachdeva port, achieving speeds >16 billion cell updates per second (GCUPS) per socket. In addition to improved throughput, this implementation is able to handle sequences of 32K residues.

2 METHODS

2.1 Smith–Waterman

The algorithm used to compute the optimal local alignment is the Smith–Waterman (Smith and Waterman, 1981) with the Gotoh (1982) improvements for handling multiple sized gap penalties. The two sequences to be compared, the query sequence and the database sequence, are defined as Q and D with lengths m and n respectively. The individual residues for Q and D are $q_1, q_2 \ldots q_m$ and $d_1, d_2 \ldots d_n$. A scoring matrix $W(q_i, d_j)$ is defined for all residue pairs. The penalties for starting and continuing a gap are defined as G_{init} and G_{ext}. The Smith–Waterman equation is defined in (1), (2) and (3). The values for $H_{i,j}$, $E_{i,j}$ and $F_{i,j}$ are defined as 0 when $i < 1$ or $j < 1$.

$$E_{i,j} = \mathbf{max}\begin{Bmatrix} 0 \\ E_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{init} \end{Bmatrix} \quad (1)$$

$$F_{i,j} = \mathbf{max}\begin{Bmatrix} 0 \\ F_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{init} \end{Bmatrix} \quad (2)$$

$$H_{i,\,j} = \mathbf{max}\begin{Bmatrix} 0 \\ E_{i,\,j} \\ F_{i,\,j} \\ H_{i\,-\,1,\,j\,-\,1} + W\left(q_i, d_j\right) \end{Bmatrix} \quad (3)$$

2.2 Striped Smith–Waterman

The Striped Smith–Waterman (Farrar, 2007) is divided into three major parts. The first part is the generation of the scoring profile used in the calculations. The next part is the actual calculations used in computing the local alignment score. Finally the *Lazy F* loop, used to correct any errors from the initial calculations.

The layout used by the query profile is a striped access parallel to the query sequence. The query is divided into p equal length segments S, where p is equal to the number of elements being processed in the SIMD register. The length of each segment t is $\left(m + p - 1\right)/p$. If the query is not long enough to completely fill all the segments, $t \times p > m$ then the segments are padded with null entries that have a weight of zero. Letting S_k^j be the j^{th} entry in the k^{th} segment S where $1 \le j \le t$ and $1 \le k \le p$ then vector $\left\langle H \right\rangle_i$ is defined as $\left\langle S_1^i, S_2^i, \ldots, S_p^i \right\rangle$.

The calculation of a match score for $H_{i,j}$ is defined as $H_{i-1,j-1} + W\left(q_i, d_j\right)$. Using vectors, the match calculation for $\left\langle H \right\rangle_i$ is defined as $\left\langle H \right\rangle_{i-1} + \left\langle W \right\rangle_i$. When $i = 1$, vector $\left\langle H \right\rangle_0$ is defined as $\left\langle 0, S_1^t, S_2^t, \ldots, S_{p-1}^t \right\rangle$ to carry the values of the last vector to the first vector. The initial value for $\left\langle F \right\rangle_0$ is set to all zeros. Any errors are corrected in the *Lazy F* loop.

For most cells, F remains at zero and does not contribute to the value of H. Only when H is greater than $G_{init} + G_{ext}$ will F start to influence the value of H. The *Lazy F* loop is executed while any element of $\left\langle F \right\rangle_{i-1} > \left\langle H \right\rangle_i - G_{init}$. If $i = 1$ then $\left\langle F \right\rangle_0$ is defined as $\left\langle F \right\rangle_t$ shifted right by one element. Shifting the contents of $\left\langle F \right\rangle_t$ moves the values in the vector to the next column. H is corrected by $\left\langle H \right\rangle_i = \mathbf{max}\left(\left\langle H \right\rangle_i, \left\langle F \right\rangle_i\right)$. The penalty G_{ext} is subtracted from the vector F and the loop is repeated. If the loop has iterated t times, the contents of vector F are again shifted one element to the right and the loop continues at the beginning with $\left\langle H \right\rangle_1$.

2.3 Implementation

When porting the Smith–Waterman (Smith and Waterman, 1981) algorithm to the Cell B.E. special attention needs to be paid to the limitations of the SPU. The small size of the LS, 256 KB, will impact the size of the two sequences compared. With no instructions supporting saturated math, a solution is needed that will not greatly impact performance of the inner loop. Finally, with only support for 16 and 32 bit integer arithmetic, a solution needs to be found to increase throughput of the Smith–Waterman calculation.

The Striped (Farrar, 2007) implementation of the Smith–Waterman algorithm was used when optimizing for the Cell B.E. This is the fastest of the SIMD implementations and can easily be adapted to the Cell B.E. instruction set. With some modifications, the issues concerning space, saturated math and throughput can all be addressed.

The Striped implementation relies on two buffers for storing the E and H values and lookup table generated from the scoring matrix W matching Q for each possible residue. This table greatly resembles a PSSM (Gribskov *et al*., 1987). This PSSM is used to load the weights for a vector with a single instruction. The size of the PSSM is $m \times r \times s$ where r is the number of columns in the PSSM and s is the size of the vector element. For each residue in Q, 50 bytes are needed to store the PSSM.

To free up more space in the LS, the W vector is calculated for each iteration.. The vector W is generated using the shuffle instruction (IBM C/C++, 2007). The shuffle instruction reorders the data of two source registers into a third target register based on a shuffle mask. Using the query sequence as the shuffle mask and the scoring matrix as the two source vectors, the W vector is generated with one shuffle instruction.

This approach greatly increases the available memory for the H and E arrays. Now each residue in Q needs only six bytes, two bytes for the query residue and two bytes each for the H and E values. This implementation divides the available space equally between the query sequence and database sequence resulting in ability to handle sequences of 32K residues.

The lack of support in the SPUs for saturated math needs to be worked around. Saturated math keeps the values of the vector within the range of the specified type. If an over-flow condition occurs, the value is clipped to the ceiling of the specified type, i.e. 32,767 for a signed short. If an under-flow condition occurs, the value is clipped to the floor of the specified type, i.e. -32,768 for a signed short. Sachdeva *et al*., (2007) replaced the saturated math instruction with seven instructions. The saturated math operations easily dominated the time used to calculate a cell's values.

An additional way of implementing saturated math is to artificially limit the range an all vector calculations preventing any one calculation for under-flowing or over-flowing. Then use the maximum operator to keep the values within the artificial limits. This faster method is used in this implementation.

The inputs for a cell's calculations are limited to G_{init}, G_{ext} and W. By biasing the initial E, F and H values, these calculations cannot under-flow and maximum operators can take the place of the expensive saturated math operations. The initial bias is defined as $b = \left|\mathbf{min}\left(0, -G_{init}, -G_{ext}, W\right)\right|$. The values for $H_{i,j}$, $E_{i,j}$ and $F_{i,j}$ are defined as b when $i < 1$ or $j < 1$. Since the Smith–Waterman calculations now have a floor of b, the equations (1), (2) and (3) become (4), (5) and (6) respectively.

$$E_{i,\,j} = \mathbf{max}\begin{Bmatrix} b \\ E_{i\,-\,1,\,j} - G_{ext} \\ H_{i\,-\,1,\,j} - G_{init} \end{Bmatrix} \quad (4)$$

$$F_{i,j} = \mathbf{max}\begin{Bmatrix} b \\ F_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{init} \end{Bmatrix} \quad (5)$$

$$H_{i,j} = \mathbf{max}\begin{Bmatrix} b \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + W(q_i, d_j) \end{Bmatrix} \quad (6)$$

Saturated addition is not needed in the Smith–Waterman calculations. If the *H* value saturates, the calculated scores are incorrect and need to be recalculated with at a higher precision to prevent the over-flow. Therefore, the saturated addition can be replaced with unsaturated addition and a test for the over-flow condition. The over-flow test is a simple as testing $H_{i,j}$ for a value greater than the ceiling. The ceiling *c* is defined as $X - \mathbf{max}(0, W)$, where *X* is the maximum value for the specified vector type, i.e. 65,535 for an unsigned short. If any element of $\langle H \rangle_i \geq c$, where $1 \leq i \leq t$, the next column has the potential to over-flow and the calculations need to be restarted either with a scalar or a higher precision implementation to prevent the over-flow. To improve calculation times, this test is moved out of the inner loop and performed once after a column's calculations by comparing the maximum score vector to the ceiling. If any value in the maximum score vector is greater than the ceiling, the calculations are rerun with a higher precision.

Another modification has been made to the *Lazy F* loop. This loop has a check to see if the calculations have reached the last vector, and need to rollover. If so, the *F* values are shifted right and the loop is restarted with the first *H* element. This check has been moved out of the *Lazy F* loop to the end of the loop. This reduced the number of cycles per iteration by 1/3. For a moderately sized query sequence with a length of 500 residues, the number of rollovers is < 1%. For those few cases of rollover, the loop is repeated at the cost of a branch miss-prediction penalty.

The SPU's native instruction set support arithmetic operations on vectors divided into 16 bit and 32 bit elements. This limits the number of cells processed per vector register to eight. The SSE2 and Altivec instructions also support 8 bit elements in the vector registers. This allows 16 cells to be processed per vector register, effectively doubling the number of cells calculated per loop iteration. The drawback to using 8 bit elements is the range of scores possible to calculate. An 8 bit element is able to hold a score between 0-255. If the score is greater than 255, the calculation needs to be run using a higher precision.

To increase the throughput of this implementation, the scores are packed in 8 bits. The *W* vector is arranged with the weights also in 8 bit elements. The ceiling is calculated for an 8 bit value by $255 - \mathbf{max}(0, W)$ to prevent any over-flows from 8 bits. The calculations still use the 16-bit arithmetic instructions, but all comparisons will be using 8-bit instructions. The comparison casts the vectors to unsigned characters enabling the compiler to generate the correct instruction for the compare. After each column is calculated, if $\langle H \rangle_i \geq c$ where $1 \leq i \leq t$ the calculations will continue but with an increased range of 16-bits. Using this method, each loop iteration will process 16 cells.

3 RESULTS

A multi-threaded test framework was developed to run the Striped Smith-Waterman (Farrar, 2007) implementations on both an Intel and Cell B.E. processors. All the Smith-Waterman implementations were written in C using intrinsic functions for that processor. The program for the Intel processor was compiled using GCC 4.1.1 and for the Cell B.E. the XLC 8.2 compiler was used. By using intrinsic functions instead of assembler, the compiler was responsible for optimizations, such as register usage, instruction selection and instruction scheduling.

The programs were tested on three computers. The first was a Dell blade with two 1.6 GHz Xeon 5130 processors (a total of eight cores) with 4 GB of RAM. The next computer was an IBM QS20 blade with two 3.2 GHz Cell B.E. processors (a total of 16 SPEs) and 1 GB of RAM. The last computer was Sony's PlayStation 3 (PS3) with a single 3.2 GHz Cell B.E. processor (a total of 6 SPEs) and 256 MB of RAM. Additionally, the PS3 has a hypervisor running which controls access to all hardware.

All queries were run against Swiss-Prot release 45 comprising 59,631,787 amino acids in 163,235 sequence entries. To test the different Smith-Waterman implementations, 11 query sequences were used ranging is size from 143 to 567 amino acids. These sequences were used to test other algorithms including BLAST 2 (Altschul *et al*., 1997), SWMMX (Rognes and Seeberg, 2000) and SWSSE2 (Farrar, 2007). Two different scoring matrices were used Blosum50 and Blosum62 (Henikoff and Henikoff, 1992). With higher scores, the *Lazy F* loop is executed more often than with lower scores.

To measure the speed of the Smith-Waterman implementations on the different processors and not the peripherals, the test frame work loads the entire sequence database into memory. For this reason, an older and smaller version of Swiss-Prot was chosen that would fit into the available memory of the smallest machine. Once

	Wozniak		Striped					
	QS20		PS3		QS20		Intel	
Sequence	Time	Speed	Time	Speed	Time	Speed	Time	Speed
P00762	21.2	0.7	9.1	1.6	9.1	1.6	7.7	1.9
P01008	39.6	0.7	14.1	2.0	14.1	2.0	11.5	2.4
P01111	16.4	0.7	7.7	1.5	7.7	1.5	6.7	1.7
P02232	12.4	0.7	6.5	1.3	6.5	1.3	5.8	1.5
P03435	48.4	0.7	16.8	2.0	16.7	2.0	13.8	2.5
P05013	16.4	0.7	7.6	1.5	7.5	1.5	6.5	1.7
P07327	32.1	0.7	12.2	1.8	12.2	1.8	10.1	2.2
P10318	31.4	0.7	11.8	1.8	11.7	1.8	10.7	2.0
P10635	43.0	0.7	15.3	1.9	15.2	1.9	12.6	2.4
P14942	19.2	0.7	8.4	1.6	8.4	1.6	7.2	1.8
P25705	47.7	0.7	16.5	2.0	16.5	2.0	13.9	2.4

Table 1. Single threaded scan times for the using the Blosum50 scoring matrix with a penalty of 10-2k. The scan time is in seconds and speed is in billion cell updates per second.

	Wozniak				Striped									
	QS20				PS3		QS20				Intel			
	8 Threads		16 Threads		6 Threads		8 Threads		16 Threads		4 Threads		8 Threads	
Sequence	Time	Speed	Time	Speed	Time	Speed	Time	Speed	Time	Speed	Time	Speed	Time	Speed
P00762	2.7	5.5	1.3	11	1.5	9.6	1.1	12.9	0.6	25.6	2.2	6.6	1.1	13.8
P01008	5.0	5.6	2.5	11.1	2.4	11.7	1.8	15.6	0.9	31.2	3.0	9.2	1.6	17.6
P01111	2.1	5.5	1.0	10.9	1.3	8.7	1.0	11.5	0.5	22.5	1.7	6.5	0.9	12.4
P02232	1.6	5.5	0.8	10.9	1.1	7.8	0.8	10.4	0.4	20.7	1.5	5.7	0.8	11.2
P03435	6.1	5.6	3.1	11.1	2.8	12.0	2.1	15.9	1.1	31.4	3.5	9.6	1.8	18.6
P05013	2.1	5.5	1.0	10.9	1.3	8.9	1.0	11.9	0.5	23.7	1.6	6.8	0.9	12.9
P07327	4.0	5.5	2.0	11	2.0	10.9	1.5	14.5	0.8	28.6	2.6	8.7	1.3	16.8
P10318	3.9	5.5	2.0	10.9	2.0	10.9	1.5	14.6	0.7	28.9	2.7	8.1	1.3	16.7
P10635	5.4	5.5	2.7	10.9	2.6	11.4	1.9	15.3	1.0	29.9	3.3	9.1	1.8	16.9
P14942	2.4	5.5	1.2	11	1.4	9.4	1.1	12.5	0.5	24.9	1.8	7.2	1.0	13.2
P25705	6.0	5.5	3.0	11	2.8	11.9	2.1	15.7	1.1	30.9	3.6	9.2	1.8	18.2

Table 2. Multi threaded scan times using Blosum50 scoring matrix with a penalty of 10-2k. The scan time is in seconds and speed is in billion cell updates per second.

loaded in memory, the database is divided into equal size blocks for each thread to process. With each block independent of the other, no thread synchronization is necessary during the calculations. After the database has been loaded and each thread has initialized any necessary data, the timer is started. When all the threads have completed, the timer is stopped. The GCUPS are calculated by the multiplying the length of the query sequence by the length of the database divided by the elapsed time.

For the Cell B.E. processors, the elapsed time includes the time to transfer the sequences from main memory to the LS. To improve transfer efficiency, the database sequences in main memory and in the LS are aligned within the cache line and data structures are initialized during the transfer of the sequence to improve. Since the transfers take < 1% of the execution time, more elaborate transfer techniques such as double buffering were not implemented (IBM Tutorial, 2007).

For a performance baseline, the Sachdeva *et al*., (2007) implementation was modified to use the test frame work. Since all the test sequences are less than 600 residues, this implementation was modified to handle only query sequences less than 600 residues. This enabled more space to be allocated to the *H* and *F* buffers, increasing the maximum database sequence size from 2,000 residues to 6,500 residues. All sequences in the Swiss-Prot database longer the maximum allowed length, were truncated. This reduced the number of residues for the Wozniak tests by 11,840.

The first test runs the eleven searches against the database using a single thread, Table 1. The scoring matrix used was Blosum50 with a gap penalty of 10-2k. Comparing the times of the PS3 to the QS20 would indicate that the hypervisor has a minimal impact on the execution of the calculations. By increasing the number of cells processed per SIMD register and decreasing the number of instructions in the inner loop, the Striped implementation executed 2-3 times faster than the Wozniak. Another point of interest is the X86 times compared to the Cell B.E. Even though the Intel processor has a much lower clock speed, 1.6 GHz vs. 3.2 GHz, its

	Wozniak				Striped									
	QS20				PS3		QS20				Intel			
	8 Threads		16 Threads		6 Threads		8 Threads		16 Threads		4 Threads		8 Threads	
Sequence	Time	Speed	Time	Speed	Time	Speed	Time	Speed	Time	Speed	Time	Speed	Time	Speed
P00762	2.7	5.5	1.3	11	1.4	10.4	1.1	13.9	0.5	27.6	1.8	7.9	0.9	17.2
P01008	5.0	5.6	2.5	11.1	2.2	12.4	1.7	16.6	0.8	33.1	2.6	10.5	1.3	20.5
P01111	2.1	5.5	1.0	10.9	1.2	9.6	0.9	12.7	0.5	24.9	1.3	8.5	0.7	16.3
P02232	1.6	5.5	0.8	10.9	1.0	8.7	0.7	11.6	0.4	23.2	1.1	7.6	0.6	15.3
P03435	6.1	5.6	3.1	11.1	2.7	12.6	2.0	16.7	1.0	33.0	3.1	11.0	1.6	20.9
P05013	2.1	5.5	1.0	10.9	1.2	9.7	0.9	12.9	0.4	25.8	1.3	8.8	0.7	16.7
P07327	4.0	5.5	2.0	11	1.9	11.6	1.4	15.5	0.7	30.7	2.2	10.3	1.1	20.1
P10318	3.9	5.5	2.0	10.9	1.9	11.6	1.4	15.5	0.7	30.7	2.3	9.3	1.1	19.8
P10635	5.4	5.5	2.7	10.9	2.4	12.1	1.8	16.2	0.9	31.8	2.9	10.1	1.5	19.2
P14942	2.4	5.5	1.2	10.9	1.3	10.3	1.0	13.7	0.5	27.3	1.4	9.3	0.8	16.2
P25705	6.0	5.5	3.0	11	2.6	12.5	2.0	16.7	1.0	32.7	3.1	10.7	1.6	20.8

Table 3. Multi threaded scan times using Blosum62 scoring matrix with a penalty of 10-2k. The scan time is in seconds and speed is in billion cell updates per second.

calculation times were 20% faster. This can be attributed to differences in the SIMD instructions. Intel's SSE2 instructions support saturated math and a maximum instruction. This leads to the inner loop of the Intel having eight fewer SIMD instructions than the Cell B.E.

The final two searches used the Blosum50 and Blosum62 scoring matrices Table 2 and Table 3 respectively, with a gap penalty of 10-2k to test the full capability of the different configurations. The searches were run using all the cores of a single socket and dual sockets configurations. The PS3 using six SPEs peaked at speeds >12 GCUPS. The QS20 using a single Cell B.E. and two Cell B.E. reached speeds >16 and >33 GCUPS respectively. Finally the Intel using four and then eight cores reached speeds >11 and >20 GCUPS respectively.

4 DISCUSSION

Further optimizations of the Striped Smith-Waterman implementation on the Cell B.E. processors are still possible. Unrolling the inner loop would allow the compiler more freedom in scheduling instruction thus reducing the number of data dependency stalls. Another optimization would be to remove the updating of *E* in the *Lazy F* loop when an insert cannot immediately follow a delete (Durbin *et al*., 1998). Attention must be taken as to not introduce additional branches miss-prediction penalties which could easily out strip any performance gains.

Other bioinformatics applications which make use of the dynamic programming algorithm could benefit from Cell B.E. eight SPEs such as ClustalW (Thompson *et al.*, 1994), SSEARCH (Pearson and Lipman, 1988) and HMMER (Eddy, 1999). Each of these programs would pose unique problems when ported to the Cell B.E. Additionally, the slow speed of the PPC core will require more routines be ported to the SPU or offloaded to a faster processor.

Additionally, the divide and conquer algorithms used in bioinformatics could also benefit from the multiple SPUs of the Cell B.E. One example would be the Myers and Miller (1988) linear space alignment algorithm. As the problem is sub-divided, keep assigning an SPU a portion of the work until all available resources are used. A multi-threaded implementation of this algorithm has improved the progressive alignment times of ClustalW (Chaichoompu *el at.*, 2006).

ACKNOWLEDGEMENTS

The author acknowledges Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation, for the use of Cell Broadband Engine resources that have contributed to this research.

REFERENCES

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., Rapp, B. A. and Wheeler, D.L. (2000). Genbank. *Nucleic Acids Res.*, **28**, 15-18.

Chaichoompu, K., Kittitornkun, S. and Tongsima, S. (2006) MT-ClustalW: Multithreading Multiple Sequence Alignment. In *Proceedings of the 20th IEEE International Parallel and Distributed processing Symposium: Rhodes Island, Greece, April 25-29, 2006*. http://www.hicomb.org/HiCOMB2006-07.pdf. Accessed 2008 April 16.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, A. (1998) *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK, 29-30.

Eddy, S. (1999) Profile hidden Markov models. *Bioinformatics*, **14**, 755-763.

Farrar, M. (2007) Striped Smith-Waterman speeds database searches six times over other simd implementations. *Bioinformatics*, **23**, 156-161.

Gotoh, O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.*, **162**, 705-708.

Gribskov, M., McLachlan, A. D. and Eisenberg D. (1987) Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, **84**, 4355-4358.

Henikoff, S., and Henikoff, J. G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, **89**, 10915-10919.

IBM (2007) Cell Broadband Engine Programming Handbook. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9F820A5FFA3ECE8C8725716A0062585F/$file/CBE_Handbook_v1.1_24APR2007_pub.pdf. Accessed 2008 April 15.

IBM (2007) Cell Broadband Engine Programming Tutorial. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/FC857AE550F7EB83872571A80061F788/$file/CBE_Programming_Tutorial_v3.0.pdf. Accessed 2008 April 15.

IBM (2007) C/C++ Language Extensions for Cell Broadband Engine Architecture. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/30B3520C93F437AB87257060006FFE5E/$file/Language_Extensions_for_CBEA_2.5.pdf. Accessed 2008 April 15.

Myers, E. and Miller, W. (1988) Optimal alignments in linear space. *Comput. Appl. Biosci.*, **4**, 11–17.

Pearson, W. R. and Lipman, D. J. (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, **85**, 2444-2448.

Rognes, T. and Seeberg, E. (2000) Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, **16**, 699-706.

Smith, T. F. and Waterman, M. S. (1981) Identification of common molecular subsequencees. *J. Mol. Biol.*, **147**, 195-197.

Thompson, J. D., Higgins, D. G. and Gibson, T. J. (1994) CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Sachdeva, V., Kistler, M., Speight, E. and Tzeng, T. K. (2007) Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications. In *Proceedings of the 21st IEEE International Parallel and Distributed processing Symposium: Long Beach, California, March 26-30, 2007*. http://www.hicomb.org/HiCOMB2007/proceedings.html. Accessed 2008 Mar 20.

Wozniak, A. (1997) Using video-oriented instructions to speed up sequence comparison. *Comput. Appl. Biosci.*, **13**, 145-150.