Preliminary Analysis of the Cell BE Processor Limitations for Sequence Alignment Applications

Sebastian Isaza[1], Friman Sánchez[2], Georgi Gaydadjiev[1], Alex Ramirez[2,3], and Mateo Valero[2,3]

[1] Delft University of Technology, Computer Engineering Lab, The Netherlands
{sisaza,georgi}@ce.et.tudelft.nl,
[2] Technical University of Catalonia, Computer Architecture Department, Spain
{fsanchez}@ac.upc.edu,
[3] Barcelona Supercomputing Center-CNS, Spain
{mateo.valero,alex.ramirez}@bsc.es,

Abstract. The fast growth of bioinformatics field has attracted the attention of computer scientists in the last few years. At the same time the increasing database sizes require greater efforts to improve the computational performance. From a computer architecture point of view, we intend to investigate how bioinformatics applications can benefit from future multi-core processors. In this paper we present a preliminary study of the Cell BE processor limitations when executing two representative sequence alignment applications (Ssearch and ClustalW). The inherent large parallelism of the targeted algorithms makes them ideal for architectures supporting multiple dimensions of parallelism (TLP and DLP). However, in the case of Cell BE we identified several architectural limitations that need a careful study and quantification.

1 Introduction

Currently, bioinformatics is considered as one of the fields of computing technology with fastest growth and development [4]. This is a vast field composed of a variety of tasks, each with different computational requirements, algorithms, data, and so on. One of the most important tasks is the comparison and alignment of biological sequences (DNA, proteins, RNA), which is basically the problem of finding an approximate pattern matching between two or more sequences.

At the algorithmic level, researchers have developed various approaches for sequence comparison that fall into two categories: *global alignment* and *local alignment*. In the first case, the goal is to find the best possible alignment that span the entire length of the sequences. In contrast, the local alignment goal is to identify some regions of the sequences where similarity between them exists. Several algorithms using dynamic programming techniques (DP) for the two approaches have been proposed. Among them, the Smith-Waterman (SW) [17] and the Needleman-Wunsch (NW) [9] algorithms are widely recognized as the best optimal methods for local and global alignment, respectively [15].

Regardless of the method used, sequence comparison using DP techniques is computationally demanding. In addition, the scenario becomes more challenging when it is required to study the similarity between one sequence and hundreds of thousands of sequences stored in a database; or when it is required to compare complete genomes of several organisms. The computational complexity of these algorithms depends on the sequences length, for example, for two sequences of length n and m, both algorithms have a complexity of *O(nm)*. Efficient implementations of the targeted algorithms are available in the Ssearch [10] and ClustalW [7] applications. They are widely recognized applications to perform sequence comparison and are representative of the field. Ssearch performs pairwise sequence alignment using the SW algorithm while ClustalW performs multiple sequences alignment using a slightly modified NW version. It is important to note that sequence alignment is a typical operation in many other bioinformatics algorithms, making our analysis applicable to a wider range of applications.

The traditional general-purpose processors do not provide a sufficient solution for bioinformatics. In addition, processor designers are lately moving away from the old superscalar approach toward multi-core systems. This is also the case with the Cell Broadband Engine processor [8] developed jointly by IBM, Sony and Toshiba, whose original target was the game box market. However, several researchers have shown that due to its characteristics, this processor is able to achieve impressive performance in other application domains such as signal processing, encryption, scientific applications and more [11]. The Cell BE architecture has a PowerPC Processing Unit (PPU) connected to 8 128-bit SIMD cores called Synergistic Processing Units (SPUs). Each SPU has a 256KB scratch pad memory called Local Store (LS) and the nine cores are connected through the Element Interconnect Bus (EIB). The EIB is a circular bus made of two channels in opposite directions each. It is also connected to the L2 cache and the memory controller.

The main contributions of this paper are:

- mapping and optimization alternatives for Ssearch and ClustalW applications while targeting Cell BE;
- qualitative analysis of the architectural limitations identified during the mapping process and their impact on performance;
- some architectural guidelines for future multi-core systems aiming at improved performance for bioinformatics workloads.

This paper is organized as follows: Section 2 provides a brief overview of recent works related to bioinformatics applications implementations on different platforms. Section 3 describes our experimental methodology. Section 4 outlines Ssearch and ClustalW applications and their implementations on the Cell BE. Section 5 analyzes the limitations we found when porting the applications to Cell BE. Finally, section 6 summarizes the paper and describes some future work directions.

2 Related Work

Various implementations of bioinformatics applications on different platforms have been reported in the literature. Some of them are based on Single-Instruction Multiple-Data (SIMD) augmented general purpose processors [12, 14] to exploit the fine-grained parallelism present in the sequence alignment applications. In the past, the SIMD processing has proven its efficiency in other application domains such as multimedia. However, due to the permanent and almost exponential growth of the amount of biological data, it becomes clear that this solution alone does not satisfy the performance demands imposed by this field.

On the other hand, many studies about bioinformatics workloads target parallel machines combining the SIMD approach with multiple processing nodes. This in order to additionally distribute the job among the different nodes. Most of these studies focus on performance evaluation and parallelization on large high-performance supercomputers [16]. These alternatives, however, are expensive and exhibit severe limitations especially in terms of power consumption.

The use of heterogeneous multi-core architectures on a single chip, e.g. the Cell BE, combines the parallelism benefits of multiprocessor systems, with the lower power consumption and higher speed interconnects of the systems on a chip. However, these alternatives have not been completely studied as a solution for bioinformatics applications. Sachdeva et. al [13] present some results on the viability of Cell BE for bioinformatics applications (ClustalW, Ssearch and Hmmer), all performing sequence alignment. In the case of Ssearch, a preliminary evaluation is reported that uses the SPUs for a pairwise alignment of only 8 sequence pairs that fit entirely in the LS memories. We believe that in order to get valid conclusions and given the different programming strategies and models that Cell BE offers, it is important to analyze the architecture behavior under the most demanding conditions, such as using large, realistic databases containing many sequences with various sizes.

Vandierendonck et al. [18], describe their ClustalW parallelization experience for the Cell BE. Their work is mainly focused on various programming optimizations while our interest is on discovering architectural limitations for a wider range of bioinformatics applications.

3 Experimental Methodology

As starting point of our study, we selected Altivec-SIMD implementations of Ssearch [2] and ClustalW [1]. We ported them to the Cell BE ISA and used an IBM BladeCenter featuring two 3.2 GHz SMT-enabled Cell BE processors each with 512 MB of RAM to gain realistic performance results.

For the Ssearch inputs we use several protein query sequences against the SwissProt database [3]. These queries represent a range of well characterized protein families used in other works to evaluate different alignment approaches [12]. The SwissProt database contains 333,445 sequence entries. We used the *blosum62* amino-acid substitution score matrix [6]. For ClustalW, the default application parameters were used and the inputs are taken from *BioPerf* benchmark suite [5]. Here we show results for data set C, which is the most challenging case containing 318 sequences of average length 1043. The applications are implemented in C. The code segments running on the PPU were compiled with ppu-gcc 4.1.1 with -O3 -maltivec options. The code in the SPU side was compiled using spu-gcc with -O3 option.

4 Applications Description and Implementation on Cell BE

This section introduces Ssearch and ClustalW workloads and discusses specific issues related with their Cell BE implementations. It is important to recall that our intention is not the development of highly optimized Cell BE specific versions of the targeted applications. Our main focus is on the analysis of the limitations that Cell BE presents at several levels in order to guide the architecture design of future multi-core systems for bioinformatics applications.

4.1 Ssearch

The Ssearch execution scenario is as follows: a query sequence is compared against all sequences of the SwissProt database. Each comparison uses the SW algorithm to compute the similarity score between sequences. During this process, scores or weights are assigned to each character-to-character comparison: positive for exact matches/substitutions, negative for insertions/deletions. As described in the SW algorithm [17], the optimal score is computed recursively. This recursion has data dependencies as shown in figure 1, where computation of the matrix cell (i, j) depends on previous results $(i-1, j)$, $(i, j-1)$ and $(i-1, j-1)$. Note that the computation of cells across the anti-diagonals are independent and the final score is reached when all the symbols have been compared.

4.2 SIMD and Cell BE Implementation of the Ssearch

As previously mentioned, the SW algorithm is the main kernel of Ssearch. It takes about 90% of the execution time of the entire application, making it the target for optimizations. For our study, we use the following implementations:

* *Ssearch Altivec SIMD version:*
This version uses the Altivec SIMD extension of PowerPC architecture with 128-bit wide registers to extract data-level parallelism by calculating temporal vector of scores of cells parallel to the anti-diagonals, as it is shown in figure 2. The process starts at the upper left moving from left to right and from top to bottom. Every time a vector anti-diagonal is computed, some temporal results have to be stored in memory because they will be used in the computation of a vector in the next row. In previous works [14], advantages and limitations of this approach were discussed extensively. In this work, we concentrate on the

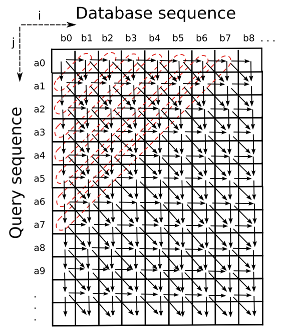


Fig. 1. Data dependencies in SW

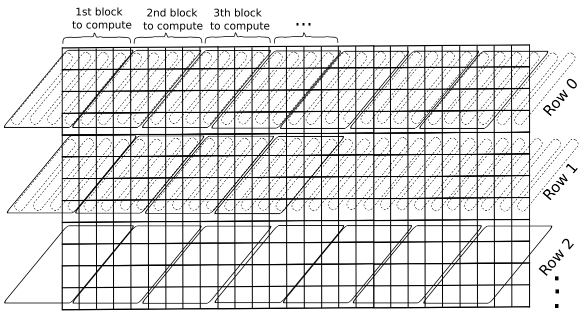


Fig. 2. Proccess by blocks of SW

analysis of Cell BE implementations.

* *Ssearch version using one SPU:*
Porting the SW implementation of Ssearch to Cell BE ISA has relevant details that impact the performance. They are described in section 5. In this implementation, we perform the same methodology that was commented in the previous paragraph for the Altivec SIMD version. In this case, the processing sometimes requires that temporal computations of a row (the border between rows) have to be stored back in memory instead of the SPUs LS. This is because the amount of computed temporal data of a row (that depends on the sequence sizes) does not always fit entirely into the LS. As a result of this, not only the traffic gets increased between memory and the LS, but also the processing of data has to wait for the DMA transfers completion. However, the impact of this limitation can be diminished by using multi-buffering to overlap the SPU processing with DMA transfers of the next data. The important decision here is the choice of appropriate block sizes of data to be computed and transferred using DMA. Figure 2 shows an example where the block size equals to 4.

* *Parallel Ssearch version 1 using multiple SPUs:*
In addition to the SIMD version above, a multi-SPU implementation was developed. The main idea was to use all available SPUs to perform the comparison between a query against the database sequences. This approach is shown in figure 3a, where every SPU is responsible for the comparison between the query and a group of database sequences, using the same scheme as in the previous paragraph. There are two important issues to consider: the efficiency of data transfers between the main memory and the LSs and the scalability. The former is related to the cases when all SPUs are communicating to the PPU simultaneously and saturating the Element Interconnect Bus. The latter is important when a higher number of SPUs are used for the parallelization.

* *Parallel Ssearch version 2 using multiple SPUs:*
Another parallel alternative is shown in figure 3b. In this case the available SPUs

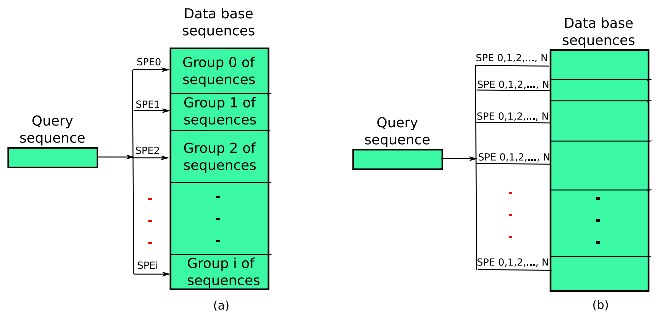


Fig. 3. Multi-SPU parallel options: (a) One SPU processes groups of sequences and (b) available SPUs process each sequence

perform the comparison between the query and a single database sequence. In this case, the computation of each matrix is distributed between the SPUs, that is, each SPU is responsible for computing several rows of the matrix. For example, when 8 SPUs are used, SPU0 computes row 0, row 8, row 16, etc; SPU1 computes row 1, row 9, row 17, and so on. There are some possible advantages of this alternative and the following issues are important: better bandwidth utilization and scalability than the previous approach. Improved bandwidth utilization can be achieved since not all the temporal results are written to main memory (the SPUs exchange data in a streaming fashion). This approach seems more scalable with increasing sequence sizes because each SPU is holding smaller pieces of data in its LS as compared to version 1 above. This alternative, however, requires additional synchronization between SPUs that can potentially degrade the performance.

4.3 ClustalW

Unlike pairwise sequence alignment, multiple sequence alignment (MSA) applications like ClustalW, align a set of sequences altogether, that are expected to have some evolutionary relationships. While for pairwise alignments it is still computationally feasible to produce optimal alignments with DP algorithms, for MSA is prohibitive and heuristics must be applied to avoid time and space complexity explosion. In particular, the time complexity of ClustalW is $O(n^4 + l^2)$, where n is the number of sequences and l their length. Using a technique called *progressive alignment* [7], ClustalW performs the multiple alignment in three main steps: 1) All-to-all pairwise alignment, 2) Creation of a phylogenetic tree, 3) Use of the phylogenetic tree to carry out a multiple alignment.

According to profiling results of the original code, the function that performs the alignments in the first step, i.e. *forward_pass*, consumes about 70% of the total execution time. This function calculates a similarity score among two sequences implementing a modified version of NW, following an approach similar to the one shown in figure 1. It is called *n(n-1)/2* times to perform the multiple alignment of n sequences (all-to-all). As opposed to the first step, the final alignment step performs only *n-1* alignments.

It is important to mention that *forward_pass* iterations are data independent making parallelization very appealing. Furthermore, the interior of the function can be at least partially vectorized to explore data-level parallelism.

4.4 Cell BE Implementation of the ClustalW

We ported *forward_pass* function to the SPU ISA and implemented a number of optimizations. DMA transfers are used to exchange data between main memory and the SPUs LS. Saturated addition and maximum instructions were emulated with 9 and 2 SPU instructions respectively. The first optimization uses 16-bit vector elements instead of 32-bit. This theoretically allows doubling the throughput but requires the implementation of an overflow check in software.

Inside the inner loop of the kernel there are instructions responsible for loading the sequence elements to be compared and using them to index a matrix that provides the comparison score. This is a random scalar memory access that is performed within a loop also containing a complex branch for checking boundary conditions. This type of operations are very inefficient in the SPUs. We have unrolled this loop and manually evaluated the boundary conditions outside the inner loop. Section 5 discusses the impact of these optimizations.

In the case of the multi-SPU versions of ClustalW, the PPU distributes pairs of sequences for each SPU to process independently. A first such a version was implemented using a simple round-robin strategy for load distribution. This version is not really efficient and is not further discussed. A second strategy uses a table of flags that SPUs can raise to indicate idleness. This way the PPU can take better decisions on where to allocate the tasks. As explained in the previous section, the parallelization of *forward_pass* in multiple threads is easy so there is no need to optimize much the load balancing nor the communication efficiency. Section 5 shows the scalability of our strategy.

5 Analysis of Cell BE Limitations and Results

Experiments performed included a number of optimizations that increase performance. We have looked mostly at parallel execution issues due to the inherent parallelism existing in the applications and some relevant Cell BE ISA aspects that impact the performance.

5.1 Performance results

Figure 4 shows the execution of Ssearch on different platforms. As expected, scalar executions are less efficient than the remaining alternatives. The G5 platform contains a powerful out-of-order superscalar PowerPC970 that runs scalar code very efficiently while the PPU has limited capabilities (less functional units and registers, in-order execution, etc). Similar observation is done for the *Altivec*

G5 and *AltivecPPU* alternatives. The *Cell1SPU* version is 1.07× slower than the *AltivecPPU* version. We have found two main reasons for this: 1) the non existing support for some instructions in the SPU ISA (discussed in the next section) and 2) the need of transferring data between LS and memory. We are currently working on further code optimizations to reduce the data reorganization overhead and the traffic between main memory and the LSs.

Figure 6 shows the execution time of Ssearch to compare a query sequence of length 553 and the whole SwissProt database. Results correspond to the parallel version 1. Each group of bars represents execution using a different number of SPUs. And each bar of the group corresponds to a different block size (in bytes) that is transfered between LS and memory as it was described in section 4.2. These results show that the strategy of computing several vector anti-diagonals of the same row before sending results to main memory is an important source of speedup (Using 1 SPU: 2,33× faster between 32 bytes and 512 bytes bars. Using 8 SPUs: 2,13× faster between 32 and 512 bytes bars). Other interesting observation is related to performance scalability across the number of SPUs used. Figure 5 shows how the performance scale almost linearly with the number of SPUs. However, part of our current work is to investigate this trend with greater number of SPUs.

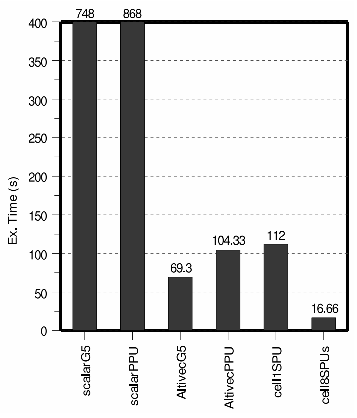


Fig. 4. Ssearch execution on different platforms

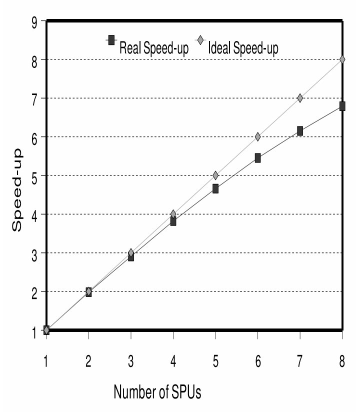


Fig. 5. Speedup across SPUs (using 512-byte block size)

Figure 7 shows a comparison of ClustalW running on various single-core platforms as compared to different versions using a single SPU. Since the clock frequency of the G5 is more than twice as low as the Cell, it is clear that in terms of cycles it outperforms any Cell 1SPU version. The fourth bar shows the straightforward SPU implementation of ClustalW, where only thread creation, DMA transfers and mailboxes are implemented for basic operation and no attention is given to optimizing the kernel code. The fifth bar shows a significant

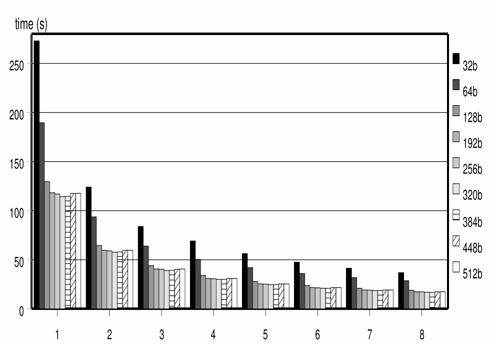
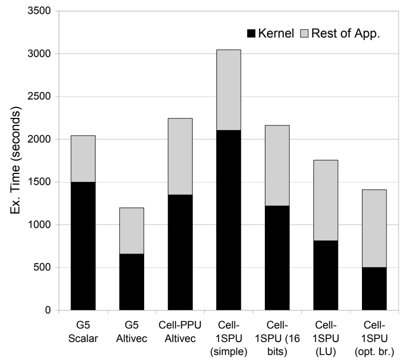


Fig. 6. Time vs SPUs for different block sizes in Ssearch

speedup (1.7×) when using 16-bit data type. This double vector parallelism is most of the time achievable but the program should always check for overflow and go back to the 32-bit version if needed. Since the SPUs do not provide any support for overflow check (unlike the PPU), this had to be implemented in software and consequently affecting the performance. The next two bars show results for unrolling a small loop located within the inner loop of the kernel allowing us to achieve accumulative 2.6× speedup. And the last two versions went further into optimizing this small loop by removing the boundary conditions involved in a scalar branch and handling them explicitly outside the loop. This final (accumulative) optimization provided about 4.2× speedup with respect to the initial version.



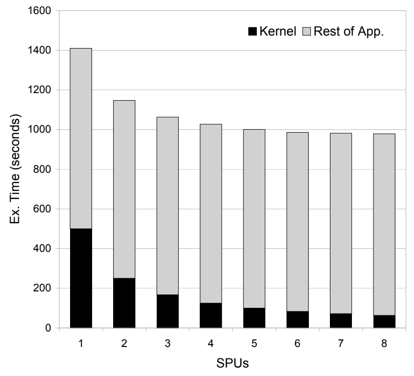


Fig. 7. ClustalW performance for different platforms and optimizations

Fig. 8. ClsutalW speedup using multiple SPUs

Figure 8 shows the scalability of ClustalW kernel when using multiple SPUs. The black part of the bars reveals a perfect scalability (8× for 8 SPUs). This is due to the relatively low amount of data transfered and the independence between every instance of the kernel. In future experiments, it will be interesting to see how far will this perfect scalability continue.

After the successful reduction of the execution time for *forward_pass*, significant application speedups are only possible by accelerating other parts of the program. The progressive alignment phase is now the portion consuming most of the time. This issue is currently being studied.

5.2 Analysis of Limitations

Here we list and discuss some limitations we found at both architecture- and micro-architecture levels. Although experiments have given us an insight about the individual contribution to performance degradation, an on-going quantitative study will tell us the real impact of every limitation.

* *Unaligned data accesses:* The lack of hardware support for unaligned data accesses is one of the issues that can limit the performance the most. When the application needs to do unaligned loads or stores, the compiler must introduce extra code that contains additional memory accesses plus some shift instructions for data reorganization. If this sort of situation appears in critical parts of the code (as is the case in ClustalW), the performance will be dramatically affected.

* *Scalar operations:* Given the SIMD-only nature of the SPUs ISA and the lack of unaligned access support, scalar instructions may cause performance degradation too. Since there are only vector instructions, scalar operations must be performed employing vectors with only one useful element. Apart from power inefficiency issues, this works well only if the scalars are in the appropriate position within the vector. If not, the compiler has to introduce some extra instructions to make the scalar operands aligned and perform the instruction. This limitation is responsible for a significant efficiency reduction.

* *Saturated arithmetics:* These frequently executed operations are present in Altivec but not in the SPU ISA. They are used to compute partial scores avoiding that they are zeroed when overflow occurs with unsigned addition. This limitation may become expensive depending on the data types. For *signed short*, 9 additional SPU instructions are needed.

* *Max instruction:* One of the most important and frequent operations in both applications is the computation of a maximum between two or more values. The SPU ISA, unlike Altivec, does not provide such an instruction. It is then necessary to replace it with two SPU instructions.

* *Overflow flag:* This flag is easily accessible in Altivec in case the application needs a wider data type to compute. In the SPU this is not available and it has to be implemented in software adding overhead.

* *Branch prediction:* The SPUs do not handle efficiently branches and the penalty of a mispredicted branch is about 18 cycles. The SPU will always predict branches as non-taken unless a software branch hint explicitly says the opposite. Although some control-dependencies (branches) can be converted in data dependencies (using select instruction) some others cannot and branches will remain.

The kernel of ClustalW has several branches that, when mispredicted, reduce the application execution speed.

* *Local Store size:* As was mentioned in section 4.2, the size of SPUs LS is relevant because it is not always possible to ensure that each database sequence, query sequence and temporal computations fit in the LS. Our SW implementation takes this into account by partitioning the work in blocks, as explained before. Other optimizations are being currently developed to dynamically identify whether space in main memory is required or not. This will help to reduce data transfer between LS and memory.

It is important to say that there are other commercial processors (apart from PowerPC) that support the missing features we found in Cell BE. For instance, TriMedia processor supports unaligned memory accesses, Intel SSE has saturating arithmetic instructions, etc. However, not all these features can be found on a single product.

6 Conclusions and Future Work

In this paper we described the mapping and some optimization alternatives of two representative bioinformatics applications targeting Cell BE. We have also presented a qualitative analysis of the architectural shortcomings identified during this process. Our study revealed various architectural aspects that negatively impact Cell BE performance for bioinformatics workloads. More precisely, the missing HW support for unaligned memory accesses, the limited memory bandwidth and LS sizes appear to be the most critical. However, additional experiments are being performed in order to measure bandwidth usage, load balancing, LS usage, functional units usage, stall rates and communication patterns. In addition, our future work involves the usage of architecture simulation techniques in order to evaluate possible solutions to the identified limitations. We are using this research as guidance for the architecture design of future multi-core systems targeting bioinformatics. We intend to widen our study to other applications of the same field.

This work is our first step towards future multi-core architectures incorporating domain specific bio-accelerators. We believe that heterogeneous multi-core architectures able to exploit multiple dimensions of parallelism are a valid option that will play an important role in the future of bioinformatics.

7 Acknowledgements

This work was partially sponsored by the European Commission in the context of the SARC Integrated Project #27648 (FP6), the HiPEAC Network of Excellence, the FEDER funds under contract TIN2007-60625 and by the Spanish Ministry of Science. The authors would like to thank the Barcelona Supercomputing Center for the access to the Cell BE blades.

References

1. Altivec enabled clustalw1.83. http://powerdev.osuosl.org/node/49.
2. Fasta web site. http://wrpmg5c.bioch.virginia.edu/fasta_www2/.
3. Swissprot, universal protein database. http://www.expasy.org/sprot/.
4. Bioinformatics market study for washington technology center, June 2003. www.altabiomedical.com.
5. D. A. Bader, Y. Li, T. Li, and V. Sachdeva. Bioperf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. *IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–8, Oct 2005.
6. J. Henikoff, S. Henikoff and S. Pietrokovski. Blocks+: a non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, 15, 1999.
7. D. Higgins, J. Thompson, T. Gibson, and J. Thompson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.
8. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, and D. Shippy. Introduction to the cell multiprocessor. *IBM Systems Journal*, 49(4/5):589–604, 2005.
9. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
10. W. R. Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the smith-waterman and FASTA algorithms. *Genomics*, 11:635–650, 1991.
11. F. Petrini, G. Fossum, J. Fernandez, A. L. Varbanescu, M. Kistler, and M. Perrone. Multicore surprises: Lessons learned from optimizing sweep3d on the cellbe. *IEEE International Parallel and Distributed Processing Symposium, IPDPS*, pages 1–10, 2007.
12. T. Rognes. *Rapid and sensitive methods for protein sequence comparison and database searching*. PhD thesis, Institue of Medical Microbiology, University of Oslo, 2000.
13. V. Sachdeva, M. Kistler, E. Speight, Tzeng, and T.H.K. Exploring the viability of the cell broadband engine for bioinformatics applications. *Proceedings of the 6th Workshop on High Performance Computational Biology*, pages 1–8, 2007.
14. F. Sanchez, E. Salami, A. Ramirez, and M. Valero. Performance analysis of sequence alignment applications. *Proceedings of the IEEE International Symposium on Workload Characterization. (IISWC)*, pages 51–60, 2006.
15. E. Shpaer, M. Robinson, D. Yee, J. Candlin, R. Mines, and T. Hunkapiller. Sensitivity and selectivity in protein similarity searches: A comparison of smith-waterman in hardware to blast and fasta. *Genomics*, 38:179–191, 1996.
16. S. Smith and J. Frenzel. Bioinformatics application of a scalable supercomputer-on-chip architecture. 1:385–391, 2003. Proceedings of the International Conference on Parallel and Distributed Processing Techniques.
17. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
18. H. Vandierendonck, S. Rul, M. Questier, and K. D. Bosschere. Experiences with parallelizing a bio-informatics program on the cell be. *Third International Conference, HiPEAC*, pages 161–175, 2008.