Limitations of Current Multicore Architectures for Bioinformatics Applications – A Case Study on Cell BE

Sebastian Isaza, Georgi Gaydadjiev

Computer Engineering Lab, Delft University of Technology, The Netherlands

ABSTRACT

The fast growth of biological databases has attracted the attention of computer scientists calling for greater efforts to improve computational performance. From a computer architecture point of view, we intend to investigate how bioinformatics applications can benefit from future multicore processors. Here we present a preliminary study of the Cell BE limitations when executing a representative bioinformatics application performing multiple sequence alignment (i.e. ClustalW). The inherent large parallelism of the core algorithm used makes it ideal for architectures supporting multiple dimensions of parallelism. However, in the case of Cell BE we identified several architectural limitations that need a careful study.

KEYWORDS: bioinformatics; multicore architectures

1 Introduction

Bioinformatics is a rapidly growing field that requires High Performance Computing (HPC) systems in order to cope with the fast increase of biological databases. One of the most important tasks in bioinformatics is the alignment of biological sequences (DNA, proteins, RNA). Popular alignment algorithms like Needleman-Wunsch (NW) [1] use dynamic programming techniques and are in most cases extremely computationally intensive. ClustalW [2] is a widely used application that features NW as its main hot-spot kernel. The inherent multi-dimensional parallelism present in this type of applications makes them ideal to be mapped on a multicore platform where both thread-level and data-level parallelism can be exploited. We have used Cell BE processor [3] as an example of a modern multicore processor.

With this research we aim at identifying the architectural and microarchitectural limitations that Cell BE exhibit when targeting a representative multiple sequence alignment application such as ClustalW. We present different optimization and parallelization strategies and analyze the factors that limit the performance.

Recent publications like [4][5] have mapped bioinformatics applications on Cell BE with a focus on software optimization. Our work aims at identifying limitations of current architectures in order to guide the design of future multicore systems.

2 ClustalW Implementation on Cell BE

ClustalW performs the multiple alignment of a set of sequences in three main steps: 1) all-to-all pairwise alignment, 2) creation of a phylogenetic tree, 3) final multiple alignment. Profiling experiments reveal that the core function of the first step (i.e. *forward_pass*) consumes about 70% of the total execution time. This function is called *n(n-1)/2* times to calculate a similarity score among two sequences, implementing a modified version of NW. Not only the independence among calls makes parallelization appealing but also vectorization of the inner loop is possible. 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 not present in the SPUs were emulated with 9 and 2 SPU instructions respectively. The first optimization uses 16-bit vector elements instead of 32-bit allowing a theoretical double throughput but requiring the implementation of an overflow check in software.

The inner loop contains random scalar memory accesses and 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. In the multi-SPU versions, the PPU distributes pairs of sequences for each SPU to be processed independently. A first such a version was implemented using a simple round-robin strategy but the load distribution was not efficient. 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.

3 Results and Analysis

Figure 1 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 G5 platform contains a powerful out-of-order superscalar PowerPC970 that runs scalar code very efficiently while the PPU has more limited capabilities (less functional units and registers, in-order execution, etc). The fourth bar shows the straightforward SPU implementation of ClustalW without optimizations. The fifth bar shows a significant 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 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. The last two versions removed boundary conditions involving a scalar branch and handled them explicitly outside the loop. This final (accumulative) optimization provided about 4.2× speedup with respect to the initial version.

Figure 2 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 transferred and the independence between every instance of the

kernel. In future experiments, it will be interesting to see how far this perfect scalability will 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.

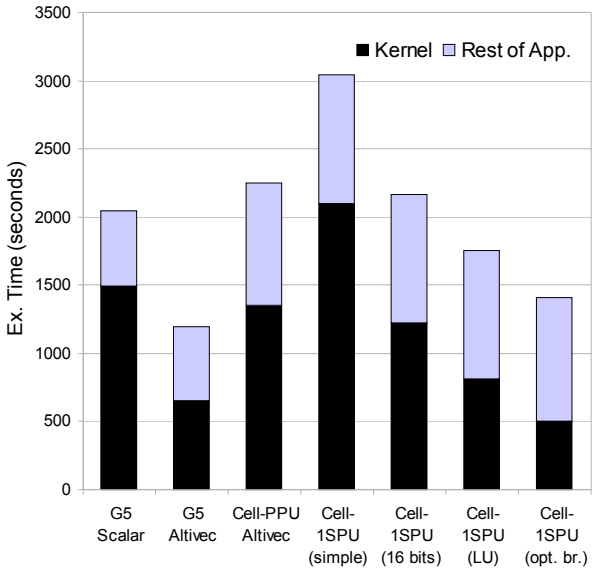

Fig. 1. ClustalW performance for different platforms and optimizations

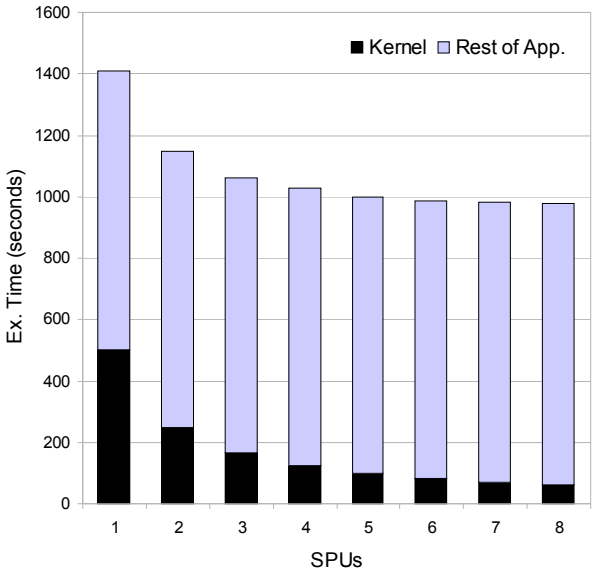

Fig. 2. ClsutalW speedup using multiple SPUs

The following is a list of the limitations we have found in the experiments:

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 instructions for data reorganization. In ClustalW, this situation appears in critical parts of the code and then performance is 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 required.

Max instruction: One of the most important and frequent operations in both applications is the computation of a maximum between two or more values. Since the SPU ISA does not provide such an instruction, it is necessary to use two SPU instructions.
Overflow flag: This flag is not available in the SPUs and has to be implemented in software, adding overhead.
Branch prediction: The SPUs do not handle branches efficiently and the penalty of a mispredicted branch is about 18 cycles. The kernel of ClustalW has several branches that, when mispredicted, reduce the application execution speed.

4 Conclusions and Future Work

We have described the mapping and some optimization alternatives of a representative bioinformatics application targeting Cell BE. 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 and the lack of saturating arithmetics instructions appear to be the most critical. We are performing additional experiments in order to have a quantitative measure of all these aspects.

Additionally, we intend to explore solutions to the issues found and validating them with simulations using UNISIM. We are using this research as guidance for the architecture design of future multicore systems incorporating domain specific accelerators for bioinformatics. We intend to widen our study to other applications of the same field. We believe that heterogeneous multicore architectures able to exploit specialization and multiple dimensions of parallelism will bring great performance improvements for bioinformatics workloads.

Acknowledgements

This work was sponsored by the European Commission in the context of the SARC Integrated Project #27648 and the HiPEAC Network of Excellence. The authors would like to thank Friman Sanchez and Alex Ramirez from the Barcelona Supercomputing Center for their contribution to this work and for the access to the Cell BE blades.

References

[1] 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*, 1994.
[2] 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*, 1970.
[3] 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.
[4] 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*, 2008.
[5] A. Aji, F. Blagojevic, W. Feng, and D. Nikolopoulos. Cell-SWat: Modeling and Scheduling Wavefront Computations on the Cell BE. *Proc. of the 2008 ACM International Conference on Computing Frontiers*, May 2008.