Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications

Vipin Sachdeva[1], Michael Kistler[1], Evan Speight,[1] and Tzy-Hwa Kathy Tzeng[2]
[1]IBM Austin Research Lab
Austin, TX 78759
{vsachde, mkistler, speight}@us.ibm.com
[2]IBM Systems and Technology Group
Poughkeepsie, NY
tzy@us.ibm.com

Abstract

This paper evaluates the performance of bioinformatics applications on the Cell Broadband Engine recently developed at IBM. In particular we focus on two highly popular bioinformatics applications – FASTA and ClustalW. The characteristics of these bioinformatics applications, such as small critical time-consuming code size, regular memory accesses, existing vectorized code and embarrassingly parallel computation, make them uniquely suitable for the Cell processing platform. The price and power advantages afforded by the Cell processor also make it an attractive alternative to general purpose processors. We report preliminary performance results for these applications, and contrast these results with the state-of-the-art hardware.

1 Computational Biology and High-Performance Computing

With the discovery of the structure of DNA and the development of new techniques for sequencing the entire genome of organisms, biology is rapidly moving towards a data-intensive, computational science. Biologists search for biomolecular sequence data to compare with other known genomes in order to determine functions and improve understanding of biochemical pathways. Computational biology has been aided by recent advances in both algorithms and technology, such as the ability to sequence short contiguous strings of DNA and from these reconstruct the whole genome [1, 24, 26]. In the area of technology, high-speed micro array, gene, and protein chips [21] have been developed for the study of gene expression and function determination. These high-throughput techniques have led to an exponential growth of available genomic data. As a result, the computational power needed by bioinformatics applications is growing exponentially and it is now apparent that this power will not be provided solely by traditional general-purpose processors.

The recent emergence of accelerator technologies like FPGAs, GPUs and specialized processors have made it possible to achieve an order-of-magnitude improvement in execution time for many bioinformatics applications compared to current general-purpose platforms. Although these accelerator technologies have a performance advantage, they are also constrained by the high effort needed in porting the application to these platforms.

In this paper, we focus on the performance of sequence alignment and homology applications on the Cell Broadband Engine. The Cell Broadband Engine (Cell BE) (tm) processor, jointly developed by IBM, Sony and Toshiba, is a new member of the IBM Power/PowerPC processor family [10]. The primary target is the PlayStation 3 (tm) game console, but its capabilities also make it well suited for various other applications such as visualization, image and signal processing and a range of scientific/technical workloads.

In previous research, we presented *BioPerf* [2], a suite of representative applications assembled from the computational biology community. Using our previous workload experience, we focus on two critical bioinformatics applications – FASTA (*ssearch34*) and ClustalW (*clustalw*). The FASTA package uses the *ssearch34* Smith-Waterman kernel to perform pairwise alignment of gene sequences, and ClustalW is used for multiple sequence alignment. A key issue in porting any application to the Cell processor is programmer productivity: the highly specialized nature of the Cell processor, such as multiple vectorized cores as well as programmer-managed caches make the porting of an application to the Cell platform more difficult than other general purpose platforms. However, our experience with these applications indicates that this effort can result in significant performance improvements over what can be achieved with current state-of-the-art general purpose microprocessors. In this paper, we discuss our implementations of FASTA and ClustalW, describing the changes required to utilize the capabilities of the Cell BE processor and the resulting perfor-

1-4244-0910-1/07/$20.00

mance improvements.

2 The Cell Broadband Engine

The Cell BE is a heterogeneous, multi-core chip optimized for compute-intensive workloads and broadband, rich media applications. The Cell BE is composed of one 64-bit Power Processor Element (PPE), 8 specialized coprocessors called Synergistic Processing Elements (SPEs), a high-speed memory controller and high-bandwidth bus interface, all integrated on-chip. The PPE and SPEs communicate through an internal high-speed Element Interconnect Bus (EIB). The memory interface controller (MIC) provides a peak bandwidth of 25.6GB/s to main memory. The Cell BE has a clock speed of 3.2GHz and theoretical peak performance of 204.8 GFLOPS (single precision) and 21 GFLOPS (double precision).

The PPE is the main processor of the Cell BE and is responsible for running the operating system and coordinating the SPEs. It is a traditional 64-bit PowerPC (PPC) processor core with a VMX unit, 32 KB Level 1 instruction cache, 32 KB Level 1 data cache, and 512 KB Level 2 cache. The PPE is a dual issue, in-order execution design, 2-way SMT.

Each SPE consists of a Synergistic Processing Unit (SPU) and a Memory Flow Controller (MFC). The SPU is a RISC processor with 128 128-bit SIMD registers and a 256KB Local Store (LS). The SIMD pipeline can run at four different granularity: 16-way 8b integers, 8-way 16b integers, 4-way 32b integers or single-precision floating-point numbers, or 2 64b double-precision floating point numbers. The 256 KB local store is used to hold both the instructions and data of an SPU program. The SPU cannot access main memory directly. The SPU issues DMA commands to the MFC to bring data into the LS or write the results of a computation back to main memory. Thus, the contents of the LS are explicitly managed by software. The SPU can continue program execution while the MFC independently performs these DMA transactions.

There are currently other efforts for porting computational biology to the Cell processor: Folding@Home, a distributed computing project for protein folding was recently ported to the Cell processor [18] with outstanding results. The Charm++ runtime system, used for NAMD simulations, is currently in the process of being ported to the Cell [11]. There are also preliminary results for the performance of BLAST on the Cell processor[16]. Terrasoft Solutions (`www.terrasoftsolutions.com`) has recently built a facility for about 2500 playstations, to be used for bioinformatics research using popular gene-finding and sequence alignment software, available free for national laboratories and other university professionals. Folding@Home has also used the computational power of GPU's [17]. All this work points to the increasing use of accelerator technologies for achieving performance from bioinformatics kernels not available from general-purpose computing platforms. The downside of employing these technologies is that the implementation is highly specific to the accelerator technology being employed, and the effort of porting the code is non-trivial due to the non-emergence of any productive programming platform so far.

3 Sequence Analysis and its Applications

Sequence analysis refers to the collection of techniques used to identify similar or dissimilar sequences or subsequences of nucleotides or amino acids. Sequence analysis is one of the most commonly performed tasks in bioinformatics. Within the area of sequence analysis, one of the most well-known and frequently employed techniques is pairwise alignment.

Pairwise alignment is the process of comparing two sequences and involves aligning and inserting gaps in one or both sequences to produce an optimal score. Scores are computed by adding constant or nucleotide (amino-acid) specific match scores while subtracting constant scores for gaps or mismatches. The problem of comparing two entire sequences is called *global alignment*, and comparing portions of two sequences is called *local alignment*. Dynamic programming techniques can be used to compute optimal global and local alignments. Smith and Waterman [22] developed one such dynamic programming algorithm (referred to as "Smith-Waterman") for optimal pairwise global or local sequence alignment. For two sequences of length n and m, the Smith-Waterman algorithm requires O(nm) sequential computation and O(m) space.

Due to the quadratic complexity of the Smith-Waterman algorithm, various attempts have been made to reduce its execution time. One approach has employed MMX and SSE technology common in today's general purpose microprocessors to achieve significant speedups for Smith-Waterman. Other approaches utilize multiple processors to perform parts of the computation in parallel. Parallel strategies for Smith-Waterman and other dynamic programming algorithms range from fine-grained ones in which processors collaborate in computing the dynamic programming matrix cell-by-cell [15] to coarse-grained ones in which query sequences are distributed amongst the processors with no communication needs [6].

Recently, several efforts have employed FPGA's [4] or GPU's [13] for computing Smith-Waterman alignments. An implementation of Smith-Waterman has also been developed for the Crays XD1, a hybrid platform of an AMD Opteron and FPGA connected through a hypertransport link [14]. Finally, specialized single-purpose hardware [7] has been developed for the Smith-Waterman and other dynamic programming algorithms in sequence alignment

which can achieve an order of magnitude improvement in performance over most contemporary processors [20].

Despite various attempts to improve the performance of Smith-Waterman, the exponential growth of biological data exceeds the growth in computational power. This has led to the wide usage of approximate algorithms and other heuristics for finding matching sequences which, though being computationally inexpensive compared to the quadratic complexity of Smith-Waterman, do not guarantee an optimal solution to the alignment problem [3]. Thus, there exists an urgent need for ever faster solutions to the Smith-Waterman alignment problem.

In our work, we have ported two popular bioinformatics applications – FASTA and ClustalW. We have also begun the work to port HMMER to the Cell processor, and briefly describe the status of this effort in the Future Work section. These three codes along with BLAST (Basic Local Alignment Search Tool) encompass the sub-field of sequence analysis in computational biology.

The *FASTA* package applies the Smith-Waterman [22] dynamic programming algorithm to compare two input sequences and compute a score representing the alignment between the sequences. This is commonly used in similarity searching where uncharacterized but sequenced "query"genes are scored against vast databases of characterized sequences. A score larger than a threshold value is considered a match. Such a scoring mechanism is useful for capturing a variety of biological information including identifying the coding regions of a gene, identifying similar genes, and assessing divergence among other sequences.

ClustalW is a popular tool for multiple sequence alignment, which is needed to organize data to reflect sequence homology, identify conserved (variable) sites, perform phylogenetic analysis, and other biologically significant results. ClustalW is an example of a pairwise alignment, wherein all sequences are compared pairwise, based on which a hierarchy for alignment is constructed. Using this hierarchy, an alignment is constructed step by step according to the guide tree. ClustalW does not give an optimal alignment, but is fast and efficient and gives reasonable alignments for similar sequences.

HMMER aligns a sequence with a database of hidden markov models, constructed previously from biological sequence families. Each of these alignments is performed using either the Viterbi algorithm [25] or the forward algorithm.

To begin our analysis for porting the applications to the Cell processor, we used the *gprof* tool to determine the most time-consuming functions for each application. Figure 1 shows the execution profile from gprof for our three applications and the BLAST sequence analysis application. The results shown in Figures 1 are for the largest *class-C* inputs included in the BioPerf suite (`www.bioperf.org`). These results indicate that all three of our applications spend more than half of their execution time in a single function: *dropgsw* for FASTA,*forward_pass* for ClustalW, and *P7Viterbi* for HMMER.

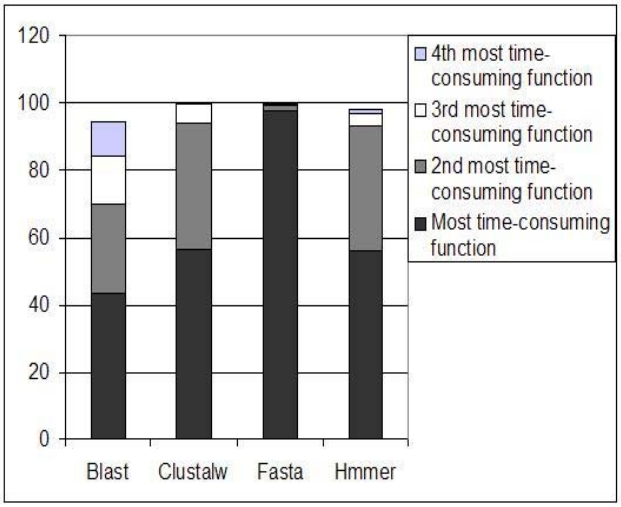

Figure 1. Function-wise breakout of BLAST, ClustalW, FASTA, and HMMER

This is a useful fact for an implementation for the Cell processor, as it implies that we might obtain a significant speedup for these applications by only porting these functions to run on the SPUs. In addition, all these applications perform multiple alignments, which are completely independent and thus can be performed in parallel across the 8 SPUs of the Cell Processor. This is also a common trait among many bioinformatics applications, which make them suitable for porting to the Cell processor with relatively little effort (compared to other applications) but potentially large performance benefits.

Another helpful characterstic for a Cell implementation is that for two out of these three kernels, *dropgsw* for FASTA and *P7Viterbi* for HMMER have open-source Altivec/SSE implementations. In addition, IBM Life Sciences has developed a vectorized kernel for the *forward_pass* kernel of ClustalW. These existing SIMD implementations make a port to the SPUs much simpler, as many of the Altivec APIs map one-to-one to the SPU APIs. The APIs which do not map one-to-one can be executed using multiple instructions on the SPU. In the following sections, we discuss the implementation of each application in detail along with issues in porting and bottlenecks that exist for every application. For further details on ClustalW, FASTA and HMMER, please refer to [23], [19], and [5] respectively.

4 Pairwise Alignment on the Cell Processor

As mentioned above, an efficient implementation of a Altivec-enabled Smith-Waterman is already included in the FASTA package by Eric Lindahl. For porting the FASTA package to the Cell processor, we began with porting this Smith-Waterman Altivec kernel *smith_waterman_altivec_word* to the SPUs converting the Altivec APIs to the SPU APIs. Many of the Altivec APIs could be converted to SPU APIs with little effort (replacing the *vec_* of Altivec to *spu_* for the SPU), and thus such an implementation could be pretty straightforward. However, there are two Altivec APIs used in FASTA that are not available on the SPU, which required us to implement these APIs with multiple instructions:

- *vec_max*: This API computes the element-wise maximum of two vectors and stores the result in the output vector. We implemented *vec_max* for the SPU by using *spu_cmpgt* to create a mask from the comparison of the two input vectors, and *spu_sel* using this mask to extract the greater of the two vectors. For more details of the SPU APIs *spu_cmpgt* and *spu_sel*, please refer to [8].

- *vec_subs*: This API performs saturated subtraction, meaning that if any element of the result vector is negative, that element is set to zero. This is a very useful API for the Smith-Waterman execution, since it needs a positive value at every matrix cell for local alignment. We implemented *vec_subs* on the SPU by performing a signed subtraction using *spu_sub* and then finding the maximum of the signed result and a constant vector of all zeros, using our implementation of *vec_max* described above.

To execute the Smith-Waterman kernel on the SPU, the alignment scores are pre-computed on the PPU, and are DMAed to the SPU along with the query and the library sequence. Other parameters such as the alignment matrix and the gap penalties are also included in the context for every SPU. With the approach above, Figure 2 shows the results of the execution of Smith-Waterman on several current general-purpose processors along with our implementation for the Cell processor. We executed a pairwise alignment of 8 pairs of sequences, using one SPU for each pairwise alignment. The Cell processor, despite the absence of instructions explained above, still outperforms every superscalar processor currently in the market. This superior performance is mainly due to the presence of 8 SPU cores and the vector execution on the SPUs. We should further state that these results are still preliminary, and further optimization of the kernel performance is still underway. For the Cell, the codes were compiled with xlc version 8.1 with the compilation flag *-O3* , which gave better or equal performance

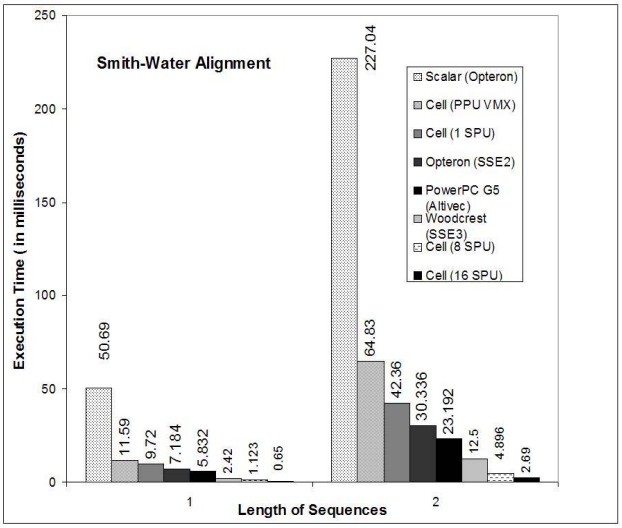

Figure 2. Performance of Smith-Waterman Alignment for different processors

compared to gcc for both SPU and PPU. For PowerPC G5, we used *-O3 -mcpu=G5 -mtune=G5*, and for Opteron and Woodcrest we used the -O3 flag. Since the PPU also supports Altivec instructions, it is also possible to use the PPU as a processing element, thus enabling 9 cores on the Cell processor, with even better performance results. Further profiling of our implementation indicates that the computation dominates the total runtime (up to 99.9% considering a bandwidth of 18 GB/s for the SPUs), and hence multi-buffering is not needed for this class of computation.

The Cell implementation discussed above is not fully functional as of now: our current implementation requires both sequences to fit entirely in the SPU local store of 256 KB, which limits the sequence size to at most 2048 characters. To do genome-wide or long sequence comparisons, a pipelined approach similar to [12] among the SPUs could be implemented. Each SPU performs the Smith-Waterman alignment for a block, notifies the next SPU through a mailbox message, which then uses the boundary results of the previous SPU for its own block computation. Support of bigger sequences on the Cell is a key goal of our future research.

Once a fully functional Smith-Waterman implementation exists on the Cell, we can employ this kernel in the FASTA package. The FASTA package compares each sequence in a query sequence file with every sequence in a library sequence file, and hence multiple issues for load-balancing could be evaluated. For now, we have a simple round-robin strategy, in which the sequences in the query library are allocated to the SPUs based on the sequence numbers and the SPU number. In many ways, the load-

balancing approach will be similar to the one discussed below in Section 5 for ClustalW. For FASTA, we consider extending it for bigger sequences as the more important and difficult problem.

5 ClustalW on the Cell Processor

ClustalW is a progressive multiple sequence alignment application. There are three basic steps to this process. In the first step, all sequences are compared pairwise using the global Smith-Waterman algorithm. A cluster analysis is then performed on each of the scores from the pairwise alignment to generate a hierarchy for alignment (guide tree). Finally, the alignment is built step by step, adding one sequence at a time, according to the guide tree.

The major time-consuming step of the ClustalW alignment is the all-to-all pairwise comparisons which based on the inputs, could take 60%-80% of the execution time. The *pairalign* function performs the task of comparing all input sequences against each other, thus performing a total of $\frac{n(n-1)}{2}$ alignments for n sequences.

Our algorithm design focused on running this code on the SPUs, with the rest of the code executing on the PPUs. While *pairalign* itself is made up of 4 different functions, *forward_pass* which computes the maximum score and the location of the cell inside the matrix cell for two sequences, is the most time-consuming step of *pairalign*. The open source-release of ClustalW has a scalar version of *forward_pass* which has multiple branches used for finding the maximum at every matrix cell. Since the SPUs lack dynamic branch prediction, and the branches are not easily predictable, due to the random nature of the inputs, this function is not ideally suited for SPU computation. Instead, there is a IBM Life Sciences modified version, which has a vectorized *forward_pass*, which we used for porting to the SPUs.

As with the FASTA kernel, we had to develop SPU implementations for some Altivec instructions that are not supported on the SPU. The saturated addition instruction *vec_adds* and *vec_max* of Altivec are not present in the SPU, and hence have to be executed using multiple instructions. Also, the vectorized code used the *vector status and the control register* for overflow detections, while doing computation with 16-bit (*short*) data types. The overflow detects that the maximum score is greater than the range of the *short* data type, and therefore does a recomputation using integer values. Since, this mechanism is not supported inside the SPUs, we changed the code to use 32-bit (*int*) data types to begin with. This lowers the efficiency of the vector computation, since now only four values can be packed inside a vector, unlike the eight of the original code, but this was necessary for correct execution without overflow detection.

One of the bottlenecks with the ClustalW code is the alignment score lookup, in which an alignment matrix score is read for finding the cost of match/mismatch among two characters in the sequences. This lookup is a scalar operation, and hence does not perform well on the SPU, since the SPU has only vector registers. For vector execution, four (32 bit) values are loaded into one vector, however in the code, this step is also preceded by a branch involving multiple conditions, involving both the loop variables for handling of boundary cases. Since the SPUs have only static branch prediction, such a branch, even thought *mostly taken*, was difficult to predict for the SPU. We broke the inner loop of the alignment into several different loops so that the branch evaluation now depends on a single loop variable, and the boundary cases computation can be handled explicitly. This change alone helped us to get a more than 2X performance gain.

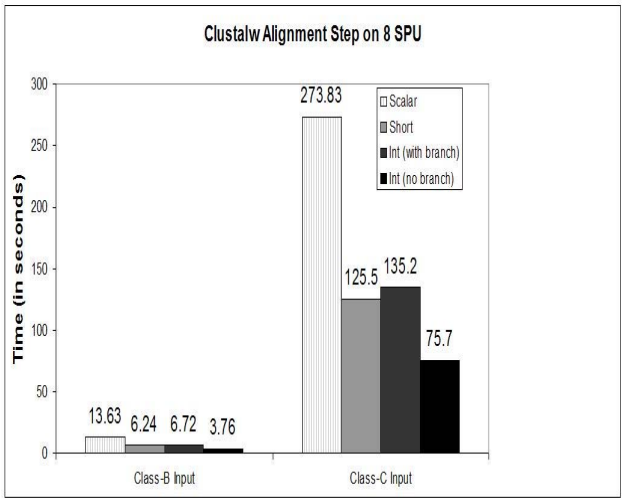

Figure 3. Improvement of performance of ClustalW alignment function with different code changes

Besides the innermost kernel execution, we also focused on partitioning of the work amongst the SPUs. Assuming there are n sequences in the query sequence file, we have a total of $\frac{n(n-1)}{2}$ computations to be performed. To distribute these computations on the SPUs, we packed all the sequences in a single array, with each sequence beginning at a multiple of sixteen bytes. This is important, as the MFC can only DMA in/out from 16-byte boundaries. This array, along with an array of the lengths of library sequences, allows the SPUs to pull in the next sequence without PPU intervention. The PPU fills in the context values, such as matrix type, gap-open and gap-extension penalties, with other inputs such as the pointers to the input array based on the command-line or default values, and the length of the se-

quence array.

For the SPU computation to begin, the PPU creates the threads and passes the maximum sequence size through a mailbox message. The SPUs allocate memory only once in the entire computation based on the maximum size, and then wait for the PPU to send a message for them to pull in the context data and begin the computation. Work is assigned to the SPUs using a simple round-robin strategy: each SPU is assigned a number from 0 to 7, and SPU k is responsible for comparing sequence number i against all sequences $i+1$ to n if $imod8 = k$. Such a strategy prohibits reuse of the sequence data, but since the communication costs are very low in comparison to the total computation, this strategy seems to work fairly well. For storing of the output values, the SPUs are also passed a pointer to a array of structures, which are 16-byte aligned, in which they can store the output of the *forward_pass* function executed for two sequences.

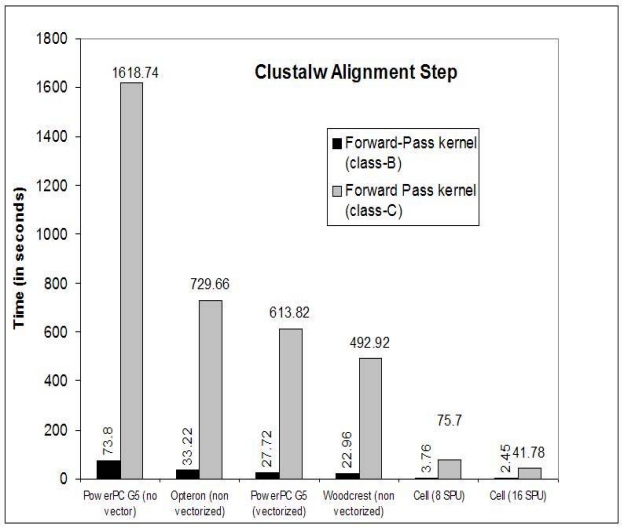

Figure 4. Comparison of Cell Performance with other processors for only alignment function

Figure 3 shows the time of computation of our Cell processor implementation of ClustalW using the strategies described above for two inputs from the BioPerf suite: 1290.seq has 66 sequences of average length 1082, and 6000.seq has 318 sequences of average length 1043. The charts show the performance of the Cell processor for scalar and vector (short and int) datatypes. It also shows how the performance notably improves with no branches. Figure 4 shows that for the *forward_pass* function, the Cell outperforms other processors by a significant margin, even with the simple round-robin strategy. We show the best implementation strategy for the Cell processor, namely using integer datatypes with no branches. The Opteron and the Woodcrest performance is non-vectorized, as we could not find a open-source SSE-enabled *forward_pass* on these platforms.

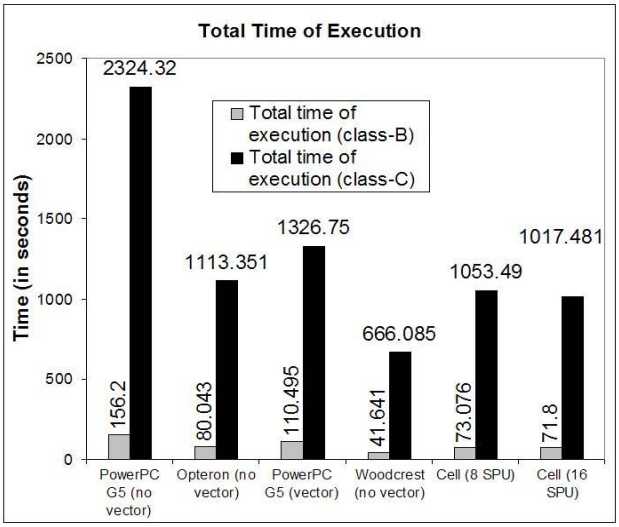

Figure 5. Comparison of Cell Performance with other processors for total time of execution

The *ClustalW* code, executing on the PPU side, uses the output for the *forward_pass* function to generate the guide tree from the scores received from the SPU, and to compute the final alignment. Computing the final alignment takes most of the remaining execution time of the application. Despite the order of magnitude gain in the execution of pairwise alignment step, the remaining code executing on the PPU is much slower in comparison to the other superscalar processors. We have not focussed on the PPU performance till now, and would look further into optimizing PPU performance for better results. Due to the PPU code execution, the overall performance of ClustalW tends to be close in comparison to the other processors. Figure 5 shows the total time of execution of ClustalW for Cell and contemporary architectures. As can be seen from the figure, the performance of Cell, despite being notably better in the *forward_pass* function is only marginally better in the overall execution time due to the performance of the PPU. There are two subproblems to this problem: either we can find more avenues to execute more code on the SPUs, or we could use Cell as an accelerator, in tandem with a modern superscalar processor could give outstanding results for ClustalW. The RoadRunner project [9] is already exploring such hybrid architectures, based on Opteron and Cell for accelerated application performance. Such a hybrid solution will be capable of best performance for ClustalW.

6 Conclusions and Future Work

In this paper, we discussed the implementation and results of two popular bioinformatics applications, namely FASTA for Smith-Waterman kernel and ClustalW. Our preliminary results show that the Cell Broadband Engine is an attractive avenue for bioinformatics applications. Considering that the total power consumption of the Cell is less than half of a contemporary superscalar processor, we consider Cell a promising power-efficient platform for future bioinformatics computing.

Our future work will focus on making the applications discussed fully operational, and trying to find other avenues for optimization. We have also begun the work to port HMMER to the Cell processor, but we do not yet have results suitable for publication. Our work to date has revealed that one of the critical issues with the HMMER implementation is that the working set size of the key computational kernel, *hmmpfam*, exceeds the space available in the 256 KB local store of the SPU for most HMM and sequence alignments. Thus, solving such inputs will require partitioning of the input among 8 SPUs while making sure that the data dependency between the SPUs are still fulfilled correctly. Thus, FASTA and HMMER suffer from similar implementation issues in the use of the multiple SPUs to solve problems which do not fit inside the local store of any one SPU. ClustalW, due to the mostly limited size of its inputs is fully functional as discussed in this paper. Besides these critical applications, we would intend to work with other applications in diverse areas such as protein docking, RNA interference, medical imaging and other avenues of computational biology to determine their applicability for the Cell processor.

References

[1] E. Anson and E.W. Myers. Algorithms for whole genome shotgun sequencing. In *Proc. 3rd Ann. Int'l Conf. on Computational Molecular Biology (RECOMB99)*, Lyon, France, April 1999. ACM.

[2] D. A. Bader, Y. Li, T. Li, and V. Sachdeva. BioPerf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *Proc. IEEE Int'l Symposium on Workload Characterization*, Austin, TX, October 2005.

[3] D. L. Brutlag, J. P. Dautricourt, S. Maulik, and J. Relph. Improved sensitivity of searches of biological sequence databases. *CABIOS*, 6(3):237–245, 1990.

[4] O. Cret, S. Mathe, B. Szente, Z. Mathe, C. Vancea, F. Rusu, and A. Darabant. Fpga-based scalable implementation of the general smith-waterman algorithm. In *Proc. 18th Int'l Conf. Parallel and Distrib. Comput. Systems (PDCS 06)*, Dallas, TX, November 2006. IASTED.

[5] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK, 1998.

[6] Z. Galil and K. Park. Parallel dynamic programming. Technical Report CUCS-040-91, Computer Science Department, Columbia Univ., 1991.

[7] R. Hughey. Parallel hardware for sequence comparison and alignment. *Comput. Applications BioSci*, 12(6):473–479, 1996.

[8] IBM. C/c++ language extensions for cell broadband engine architecture. `http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/30B3520C93F437AB87257060006FFE5E/\$file/Language_Extensions_for_CBEA.v2.2.1.pdf`, 2006.

[9] IBM. Ibm to build world's first cell broadband engine based supercomputer. `http://www-03.ibm.com/press/us/en/pressrelease/20210.wss`, 2006.

[10] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM Systems Journal*, 49(4/5):589–605, 2005.

[11] David Kunzman, Gengbin Zhang, Eric Bohm, and Laxmikant V. Kale. Charm++, offload api, and the cell processor. http://charm.cs.uiuc.edu/papers/CellPMUP06.pdf, 2006.

[12] Weiguo Liu and Bertil Schmidt. Parallel design pattern for computational biology and scientific computing applications. In *Proc. IEEE Intl. Conf. on Cluster Computing(CLUSTER'03)*, pages 456–459, Hong Kong, December 2003.

[13] Y. Lui, W. Huang, J. Johnson, and S. Vaidya. Gpu accelerated smith-waterman. In *Proc. GPGPU Workshop (GPGPU06)*, University of Reading, UK, May 2006. `http://www.mathematik.uni-dortmund.de/~goeddeke/iccs/index.html`.

[14] Steve Margerm. Reconfigurable computing in real-world applications. `http:`

//www.fpgajournal.com/articles_2006/20060207_cray.htm, 2006.

[15] W. S. Martins, J. B. Del Cuvillo, F. J. Useche, K. B. Theobald, and G. R. Gao. A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison. In *Proc. of the Pacific Symposium on Biocomputing*, pages 311–322, Hawaii, jan 2001.

[16] Chris Mueller. Blast on ibm's cell broadband engine. http://www.osl.iu.edu/ chemuell/projects/presentations/cell-blast-sc06.pdf, 2006.

[17] Vijay Pande. Folding@home on ati gpu's: a major step forward. http://folding.stanford.edu/FAQ-ATI.html, 2006.

[18] Vijay Pande. Folding@home on the ps3, December 2006. http://folding.stanford.edu/FAQ-PS3.html.

[19] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences USA*, 85:2444–2448, 1988.

[20] T. Rognes and E. Seeberg. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common multiprocessors. *Bioinformatics*, 16(8):699–706, 2000.

[21] M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467–470, 1995.

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

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

[24] J.C. Venter and *et al.* The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[25] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, IT-13:260–269, 1967.

[26] J.L. Weber and E.W. Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7(5):401–409, 1997.