Heterogeneous multi-core systems for bioinformatics

Adrianto Wirawan

2010

Adrianto, W. (2010). Heterogeneous multi-core systems for bioinformatics. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/42096

https://doi.org/10.32657/10356/42096

HETEROGENEOUS MULTI-CORE SYSTEMS FOR BIOINFORMATICS

by

Adrianto Wirawan

Supervisor: Dr. Kwoh Chee Keong and Dr. Bertil Schmidt

Division of Information Systems
School of Computer Engineering
Nanyang Technological University

A thesis submitted to the Nanyang Technological University in fulfillment of the requirement of the degree of Doctor of Philosophy

July 2009

STATEMENT OF ORIGINALITY

I hereby certify that the content of this thesis is the result of work done by myself and has not been submitted for higher degree to any other University or Institution.

…… ……

Date Signature

ABSTRACT

The bioinformatics research area is now faced with an obstacle of ever-increasing biological data to verify their biological discovery. As data increases, so does the workload for managing, processing and analysing this data. Combined with the inherent complexity of biological problems, traditional approaches results in long run-time and huge memory requirements. The emergence of accelerator technologies such as multi-core architectures provides the opportunity to achieve significant improvements in execution time for many bioinformatics applications, compared to sequential general-purpose platforms. Using multi-cores to solve large scale bioinformatics applications, such as sequence analysis, is therefore a promising and challenging research field, since large-scale computational bioinformatics problems can benefit much from this kind of processing power.

In order to implement efficient and scalable code for this type of architecture, a shift of paradigm in applications development and novel programming techniques are required. In this thesis, we investigate algorithms and techniques on how to efficiently map bioinformatics applications onto a heterogeneous multi-core system, the Cell Broadband Engine (Cell/BE). In particular, we have focused on the following important and widely used applications, i.e. alignment of long DNA sequences, Smith-Waterman algorithm, BLASTP algorithm and pairwise distance matrix computations, which is an integral part of the multiple sequence alignment algorithms such as ClustalW.

Aligning long DNA sequences is a common and often repeated task in molecular biology. We have developed a novel, efficient and scalable parallel algorithm for very long DNA sequence alignment on a heterogeneous multi-core system, the Cell Broadband Engine.

Our implementation utilizes two types of parallelization techniques: (i) SIMD vectorization within a processor and (ii) wavefront parallelization between processors. We have also introduced a partitioning scheme to overcome the local storage limitation of the Synergistic Processor Elements (SPEs) as well as a direct SPE to SPE DMA transfer communication technique. Performance evaluation shows that our implementation achieves almost linear speedup and leads to significant computational time savings for large datasets.

Next, we have demonstrated how the PlayStation® 3, powered by the Cell Broadband Engine, can be used as a computational platform to accelerate the Smith-Waterman algorithm, a method for optimal pairwise sequence alignment. For large protein datasets, our implementation on the PlayStation® 3 provides a significant improvement in running time compared to other implementations such as SSEARCH, Striped Smith-Waterman and CUDA-SW.

Furthermore, we have developed a novel implementation to accelerate a heuristic protein sequence database scanning algorithm, the BLASTP heuristic, on to a heterogeneous multi-core system, the Cell Broadband Engine. To our knowledge, this is the first ever reported parallelization of BLASTP on a heterogeneous multi-core system. We have also introduced a new parallel communication pattern, in which the Power Processor Element (PPE) coordinates the data transfer. Furthermore, we have utilized a data structure similar to compressed *deterministic finite-state automaton* (DFA) to fit the codeword lookup data in the SPEs. The BLASTP implementation on a Playstation®3 leads to significant runtime savings compared to corresponding sequential implementations.

Finally, we have developed an efficient parallel implementation that accelerates the distance matrix computation used in multiple sequence alignments on the x86 and Cell Broadband Engine architecture, a homogeneous and heterogeneous multi-core system, respectively. By taking advantage of multiple processors as well as SIMD vectorization, we are able to achieve speedups of two orders of magnitude compared to the publicly available implementations utilized in multiple sequence alignment algorithms. We have also compared the performance of our implementation on the Playstation®3 with other accelerator technologies, i.e. reconfigurable accelerators, such as FPGAs, and GPUs with CUDA programming model.

ACKNOWLEDGEMENT

First and foremost, I would like to thank God for His grace and blessings throughout the entire study.

I wish to express my deepest gratitude to my PhD supervisor Dr. Kwoh Chee Keong and Dr. Bertil Schmidt, for their valuable guidance, assistance and advice throughout the entire study. Their continuous encouragement and motivational support have been the driving force behind this study.

I would also like to thank Mr. Gerrit Voss, Mr. Tan Chee Hian, Mr. Nim Tri Hieu, Mr. Liu Yongchao and Mr. Zhang Huiliang and all others who have supported my research.

Last but not least, I would like to thank all my family and friends for their endearing love and faithful support.

AUTHOR'S PUBLICATION

Journal Papers

1. A. Wirawan, C.K. Kwoh, B. Schmidt: *Multi Threaded Vectorized Distance Matrix Computation on the Cell/BE and x86/SSE2 Architectures*, Bioinformatics, 2010, in press. doi:10.1093/bioinformatics/btq135 (Impact factor: 4.328)
2. A. Wirawan, B. Schmidt, H. Zhang, C.K. Kwoh: *High Performance Protein Sequence Database Scanning on the Cell B.E. Processor*, Scientific Programming, Vol. 17, No. 1-2, pp. 97-111, 2009
3. A. Wirawan, C.K. Kwoh, T.H. Nim, B. Schmidt: *CBESW: Sequence Alignment on the Playstation 3*, BMC Bioinformatics, Vol. 9:377, 2008 (Impact factor: 3.78).
4. Y. Liu, B. Schmidt, A. Wirawan, C.K. Kwoh, D.L. Maskell: *Comparison of Accelerator Architectures for Large-Scale Biological Sequence Alignment*, IEEE Transactions on Parallel and Distributed Systems, under review.

Conference Papers

1. A. Wirawan, B. Schmidt, C.K. Kwoh: *Pairwise Distance Matrix Computation for Multiple Sequence Alignment on the Cell Broadband Engine*, The International Conference on Computational Science 2009 (ICCS 2009), Baton Rouge, Louisiana, Springer, LNCS, Vol. 5544, pp. 954-963, 2009.
2. A. Wirawan, B. Schmidt, C.K. Kwoh: *Parallel DNA Sequence Alignment on the Cell Broadband Engine,*7th International Conference on Parallel Processing and

Applied Mathematics (PPAM 2007), Gdansk, Poland, Springer, LNCS Vol. 4967, pp. 1249-1256, 2008.

3. A. Wirawan, B. Schmidt: *Parallel Discovery of Transcription Factor Binding Sites*, IEEE Asia Pacific Conference on Circuits and Systems (APCCAS 2006), Singapore, IEEE Press, 2006.
4. N.T. Hieu, C.K. Kwoh. A. Wirawan, B. Schmidt: *Applications of Heterogeneous Structure of Cell Broadband Engine Architecture for Biological Database Similarity Search*, 2nd International Conference on Bioinformatics and Biomedical Engineering (iCBBE2008), IEEE Press, pp. 5-8, 2008.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF EQUATIONS

LIST OF FIGURES

1. INTRODUCTION

1.1. OVERVIEW

Due to the rapid progress of genome sequencing projects in the past decade, there has been an exponential increase in the amount of available genomic sequence data. The three principal comprehensive databases of nucleotide sequences currently are:

1. GenBank[1]

   GenBank is the National Institute of Health (NIH) genetic sequence database, which is composed of an annotated collection of all publicly available DNA sequences. It is maintained at the National Center for Biotechnology Information (NCBI) in Maryland, USA.

2. European Molecular Biology Laboratory (EMBL) Nucleotide Sequence Database[2]

   The EMBL Nucleotide Sequence Database constitutes Europe's primary nucleotide sequence resource. Main sources for DNA and RNA sequences are direct submissions from individual researchers, genome sequencing projects and patent applications. It is maintained at the European Bioinformatics Institute (EBI) in Cambridge, UK.

3. DNA Data Bank of Japan (DDBJ)[3]

   DDBJ is based in Japan's National Institute of Genetics. DDBJ is the sole DNA data bank in Japan, which is officially certified to collect DNA sequences from researchers and to issue the internationally recognized accession number to data submitters. It is maintained at the National Institute of Genetics in Mishima, Japan.

These three databases form the International Nucleotide Sequence Database Collaboration[4], which has led to many beneficial projects, e.g. the taxonomy project[5] and the feature table[6]. Since all three databases exchange the collected data on a daily basis, the three data banks share virtually the same data at any given time. Their objective is to ensure that nucleotide sequence information's are stored publicly and freely, such that it is easily accessible for researchers and scientists worldwide. This policy has proved to be tremendously successful for the progress of science and has led to an enormous increase in size and usage of genome databases.

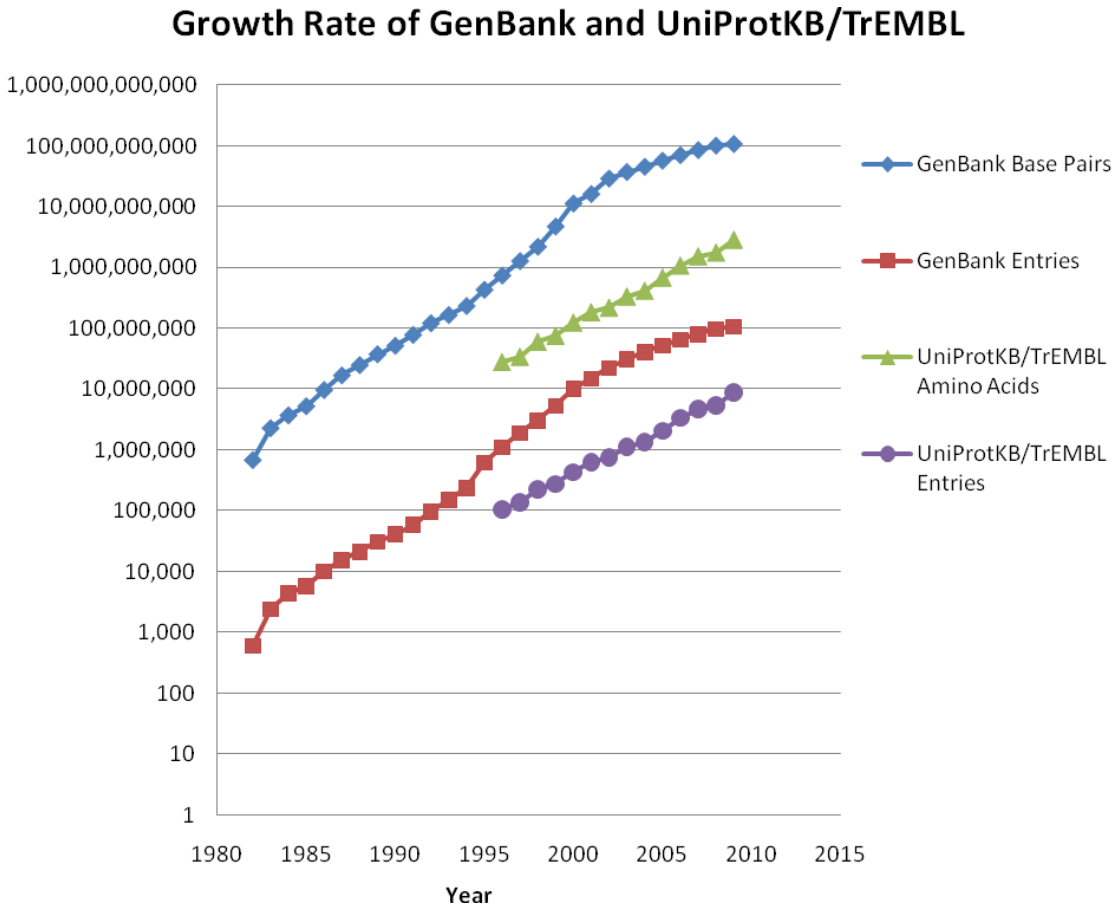

Figure 1. Growth rate of the GenBank and UniProtKB/TrEMBL databases on a semi-log graph

Genome and protein databases are growing exponentially and this growth rate will continue for a foreseeable future. The GenBank release notes for release 172.0 in June 2009 state that "from 1982 to the present, the number of bases in GenBank has doubled approximately every 18 months." This trend is also reflected in protein databases. The UniProtKB/TrEMBL Database[7, 8] release notes for release 40.5 in July 2009 states that compared to a previous release 3 months ago, the current dataset" represents an increase of 16%". Figure 1 illustrates the number of base pairs and entries in the GenBank from 1982 to June 2009 as well as the number of amino acids and entries in the UniProtKB/TrEMBL Database from 1996 to July 2009 on a semi-log graph.

Furthermore, the advent of high-throughput next generation sequencing technologies also brought a need for high throughput in bioinformatics. Two new sequencing technologies were introduced in 2005, i.e. the 454 system using pyrosequencing technology [9], and the Solexa system, which detects fluorescence signals [10]. Both sequencing techologies execute millions of sequencing reactions in parallel, producing data at ultrahigh rates [11]. These next generation technologies offer drastically faster and cost-effective sequence throughput and are vastly superior to shotgun sequencing due to the high volume of data and the drastically short time to sequence a whole genome or disease genome, although genome assembly is much more computational expansive. Therefore, the next generation sequencing technologies will foster enormous potential applications of high performance computing techniques in bioinformatics.

Bioinformatics is a growing research field which involves the use of compute-intensive techniques to solve and analyze biological data. Major research efforts in the field such as sequence alignment, prediction of gene expression, and protein-protein interactions relies

on fast and reliable computational approaches. Furthermore, most bioinformatics applications with optimal solutions are often associated with long runtimes and expensive resources. These are due to various factors:

- Biological data are obtained by experiments. Hence, they are prone to errors. The need to deal with errors and uncertainties results in high complexity algorithms.
- Some problems that can be solved using polynomial time algorithms have massive computational requirements due to large data sets that have to be analyzed.
- Many problems are computationally intensive due to their inherent algorithmic complexities, e.g. protein folding[12]. Some problems are even NP-hard problems, which means an exact solution cannot be solved in polynomial time.

The work presented in this thesis is mainly concerned with constructing efficient multi-core algorithms and techniques that address bioinformatics problems, especially in the area of sequence alignment.

1.2. MOTIVATION

In the last few decades, scientists have tried to understand how life evolved by studying the flow of genetic information in a cell. DNA (deoxyribonucleic acid) is the genetic material, which are read and translated into proteins with specific functions. A common theme throughout biological systems at all levels is that structure and function are intimately related. Therefore, the first step would be to know and understand the DNA and protein structure thoroughly, as well as the organization of the whole molecule as the genomes of an organism. Bioinformatics is a field that would provide approaches for research on DNA and protein sequences as it relies on extensive computational

approaches to decode the information hidden behind billions of nucleotides and amino acids, respectively.

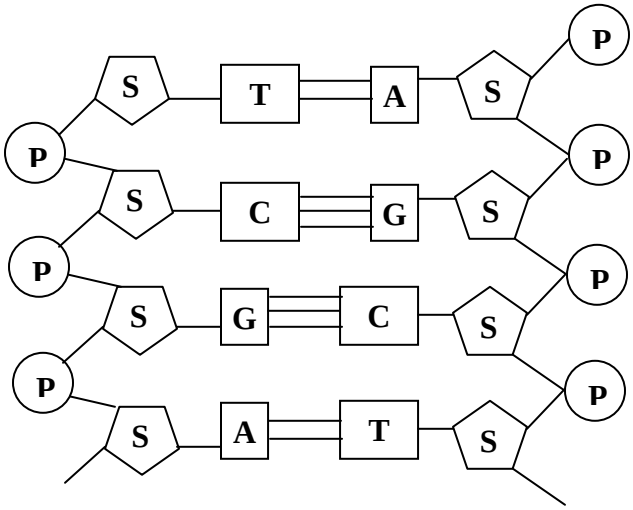

Figure 2. DNA structure

DNA was discovered in 1869. The two strands of a DNA molecule are tied together in a helical structure, known as the double helix structure[13]. Four different bases are used to form the DNA molecules: *adenine*, *cytosine*, *guanine*, *thymine* (**A**, **C**, **G**, **T**). Pairs are always formed between the bases **A** and **T**, and between **G** and **C**. Each base is attached to a phosphate group and a deoxyribose sugar to form a nucleotide, as shown in Figure 2. DNA contains the genetic instructions used in the development and functioning of all known living organisms and some viruses. The main role of DNA molecules is the long-term storage of information. DNA is often compared to a set of blueprints or a recipe, or a code, since it contains the instructions needed to construct other components of cells, such as proteins and RNA molecules. The DNA segments that carry this genetic information are called genes. Other DNA segments have structural purposes, or are involved in regulating the use of this genetic information.

Proteins (also known as polypeptides) are organic compounds made of amino acids arranged in a linear chain polymer and joined together by peptide bonds between the carboxyl and amino groups of adjacent amino acid residues. The sequence of amino acids in a protein is defined by the sequence of a gene, which is encoded in the genetic code. In general, the genetic code specifies 20 standard amino acids, however in certain organisms the genetic code can include selenocysteine and pyrrolysine. Shortly after or even during synthesis, the residues in a protein are often chemically modified by post-translational modification, which alter the physical and chemical properties, folding, stability, activity, and ultimately, the function of the proteins. Proteins can also work together to achieve a particular function, and they often associate to form stable complexes.

Like other biological macromolecules such as polysaccharides and nucleic acids, proteins are essential parts of organisms and participate in virtually every process within cells. Many proteins are enzymes that catalyze biochemical reactions and are vital to metabolism. Proteins also have structural or mechanical functions, such as actin and myosin in muscle and the proteins in the cytoskeleton, which form a system of scaffolding that maintains cell shape. Other proteins are important in cell signaling, immune responses, cell adhesion, and the cell cycle. Proteins are also necessary in animals' diets, since animals cannot synthesize all the amino acids they need and must obtain essential amino acids from food. Through the process of digestion, animals break down ingested protein into free amino acids that are then used in metabolism.

Any alignment between two or more nucleotide or amino acid sequences represents an hypothesis regarding the evolutionary history of these sequences[14]. By aligning nucleotide or amino acid sequences, scientists have been able to determine and identify

important matched and mismatched regions. Matched regions may turn out to be functional homolog pairs, conserved regulatory regions or long repeats. Mismatched regions, on the other hand, may either be Single Nucleotide Polymorphisms (SNPs) or foreign fragments inserted due to transposition, sequence reversal or lateral transfer from another organism. Hence, comparisons of related nucleotide and protein sequences have assisted many recent developments in understanding the content, relationship and function of genetic sequences. As a direct result, sequence alignment and comparison techniques as well as database sequence searching techniques have been the cornerstone of bioinformatics.

Given the continuing improvements in high throughput genomic sequencing and the exponential growth in the size of sequence databases, new advances for bioinformatics area are needed by the research and scientific community. High Performance Computing (HPC) is one of the most popular technique to improve the performance without sacrificing the correctness of the solution[15]. The recent emergence of accelerator technologies such as FPGAs, GPUs and multi-core processors have made it possible to achieve an excellent improvement in execution time for many bioinformatics applications, compared to current general-purpose platforms. Examples of bioinformatics application that takes advantage of HPC are MPIBlast[16], MPI-HMMER[17], ClustalW-MPI[18, 19], PAxML[20], Folding@home[21], Phusion[22], GPU-ClustalW[23] and ClustalW using FPGA[24].

Multi-core technology was first discussed in 1989[25]. Conceptually, multi-core architecture refers to a single processor package containing two or more processor execution cores or computational engines that deliver fully parallel execution of multiple

software threads. The operating system treats each of its execution cores as a discrete processor, with all associated execution resources.

One of the ideas behind the movement to multi-core architectures is parallelism. It is one of the best ways to address the issue of power while maintaining performance where higher data throughput may be achieved with lower voltage and frequency. The result is a larger transistor count, but overall lower power dissipation and power density. Instead of classifying based upon speed, one could classify products based upon the number of working cores or overall data throughput. The integration of multiple cores on a chip also allows lower interconnect latency and therefore higher bandwidth between cores than their discrete counterparts. Hence, microprocessor designers and manufacturers have turned to building chip multi-processors[26-29]. A survey conducted in 2009[30] shows that 90% of common computers today uses multi-core processors and this trend is expected to continue. Figure 3 illustrates the survey according to [30].

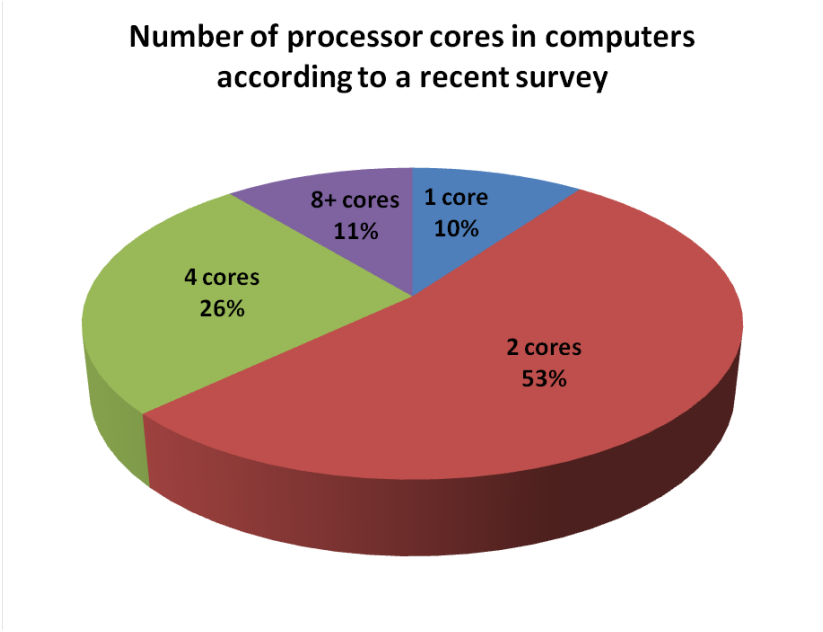

Figure 3. Number of processor cores in computers according to a recent survey

Multi-core architectures may take on a number of forms. One form is the heterogeneous multi-core architecture, which can address a variety of applications. Another form is a large number of remedial homogeneous cores which divide and conquer computationally intensive applications and yet individually address less computationally intensive applications. Yet another form consists of a few complex homogeneous cores in which a single core could multitask between several remedial applications or individually handle computationally intensive applications. For any form of multi-core architecture, the application or algorithm development process must be significantly changed in order to fully explore the potential of multi-core processors.

The development of new homogeneous and heterogeneous multi-core architectures brings a shift of paradigm in applications development. In order to implement efficient and scalable code for this type of architecture, novel programming techniques are required. This continues to remain a largely unexplored territory and is the principal motivation behind our work.

1.3. OBJECTIVES

The exponential growth of available biological data has caused bioinformatics to be rapidly moving towards a data-intensive, computational science. As a result, the computational power needed by bioinformatics applications is growing exponentially as well. Traditional approaches to sequence analysis techniques are expensive in terms of time and memory. HPC is a widely used method to improve performance. The emergence of accelerator technologies such as multi-core architecture has made it possible to achieve an excellent improvement in execution time for many bioinformatics applications,

compared to current general-purpose platforms. Therefore, using multi-cores to solve sequence analysis problems is a promising and challenging research field since large-scale computational bioinformatics problems can benefit much from this kind of processing power. Our objectives are as follows.

- Various parallel algorithms for solving sequence analysis problems have been presented for different parallel architectures, e.g. Field Programmable Gate Array (FPGA) and Graphical Processing Unit (GPU). However, multi-cores have their own characteristics. Therefore, new sequence analysis algorithms have to be presented in order to execute efficiently on multi-core architectures.
- The development of sequence analysis algorithms for multi-core architecture is made challenging by the heterogeneous nature of the resources involved. Therefore, new parallel communication patterns and partitioning scheme in parallel models are required.
- The emergence of commonly available accelerator technologies, such as FPGA, GPU, and the Cell/BE processor provide an opportunity to achieve orders-of magnitude performance. Hence, performance evaluation and comparison between these accelerator technologies is required to give a comprehensive understanding of the advantages and disadvantages of these accelerators as well as to provide a reference for mapping algorithms or applications onto them.

1.4. CONTRIBUTIONS

The contributions of our work can be briefly summarized as follows.

- We have developed a novel, efficient and scalable parallel algorithm for very long DNA sequence alignment on a heterogeneous multi-core system, the Cell Broadband Engine. Our implementation utilizes two types of parallelization techniques: (i) SIMD vectorization within a processor and (ii) wavefront parallelization between processors. We also introduced a partitioning scheme to overcome the local storage limitation of the Synergistic Processor Elements (SPEs) as well as a direct SPE to SPE DMA transfer communication technique. Performance evaluation shows that our implementation shows almost linear speedup and leads to significant computational time savings.
- We have demonstrated how the PlayStation® 3, powered by the Cell Broadband Engine, can be used as a computational platform to accelerate the Smith-Waterman algorithm, an optimal pairwise sequence alignment. For large protein datasets, our implementation on the PlayStation® 3 provides a significant improvement in running time compared to other implementations such as SSEARCH, Striped Smith-Waterman and CUDA-SW.
- We have developed a novel implementation to accelerate a heuristic protein sequence database scanning algorithm, the BLASTP heuristic, on to a heterogeneous multi-core system, the Cell Broadband Engine. To our knowledge, this is the first ever reported parallelization of BLASTP on a heterogeneous multi-core system. We also introduced a new parallel communication pattern, in which the Power Processor Element (PPE) coordinates the data transfer. Furthermore, we utilized a data structure similar to compressed *deterministic finite-state automaton* (DFA) to fit the codeword lookup data

in the SPEs. The BLASTP implementation on a Playstation®3 leads to significant runtime savings compared to corresponding sequential implementations.

- We have developed an efficient parallel implementation that accelerates the distance matrix computation used in multiple sequence alignments on the x86 and Cell Broadband Engine architecture, a homogeneous and heterogeneous multi-core system, respectively. By taking advantage of multiple processors as well as SIMD vectorization, we are able to achieve speedups of two orders of magnitude compared to the publicly available implementation utilized in multiple sequence alignment algorithms. We have also compared the performance of our implementation on the Playstation®3 with other accelerator technologies, i.e. FPGA and GPU.

1.5. SYNOPSIS OF THESIS

The rest of the thesis is structured as follows:

- Chapter 2 reviews algorithm design techniques for sequence alignment problems as well as parallel computation models and parallel architectures. Furthermore, we present a general survey of the state-of-the-art accelerator technologies in High Performance Computing (HPC).
- Chapter 3 introduces the Cell Broadband Engine, a recently introduced heterogeneous multi-core architecture system. Moreover, we discuss its characteristics, how it overcomes the *three wall limitations* as well as strategies and techniques on how to map applications onto such architecture in order to gain good performance.
- Chapter 4 elaborates our parallel algorithm to align very long DNA sequences as well as the implementation and performance evaluation on the Cell Broadband Engine.

- Chapter 5 demonstrates how the PlayStation® 3, powered by the Cell Broadband Engine, can be used as a computational platform to accelerate the Smith-Waterman algorithm for large protein datasets.
- Chapter 6 discusses our mapping of the popular heuristic protein sequence database scanning algorithm, the BLASTP on a heterogeneous multi-core system. The Playstation®3 implementation and performance evaluation are presented at the end of the chapter.
- Chapter 7 elaborates our efficient parallel implementation that accelerates the distance matrix computation used in multiple sequence alignments on a homogeneous and heterogeneous multi-core system. We also present a performance evaluation of our implementation on the Playstation®3 with other accelerator technologies.
- Chapter 8 concludes the achievement of our research work and suggests possible area of future work.

2. STATE OF THE ART

This chapter reviews algorithm design techniques, sequence alignment as a popular and important genome analysis task as well as parallel computation and parallel architectures. Furthermore, we present a general survey of the state-of-the-art accelerator technologies in High Performance Computing (HPC).

2.1. ALGORITHM IMPLEMENTATION TECHNIQUES

This section provides an overview of common algorithm design techniques used for sequence analysis.

2.1.1. EXHAUSTIVE SEARCH ALGORITHMS

Exhaustive search, or brute-force search, is a trivial but very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

Exhaustive search algorithms are simple to implement, and guaranteed to find an optimal solution if it exists. However, their costs are proportional to the number of candidate solutions, which, in many practical problems, tend to grow exponentially as the size of the problem increases. Therefore, exhaustive search algorithm is typically only used for very small problem sizes or when the simplicity of implementation is more important than speed.

2.1.2. BRANCH-AND-BOUND ALGORITHMS

Branch-and-Bound is a general design technique to find optimal solutions of optimization problems, especially in discrete and combinatorial optimization. It consists of a

systematic enumeration of all candidate solutions, where large subsets of fruitless candidates can be discarded, by using upper and lower estimated bounds of the quantity being optimized.

A branch-and-bound algorithm starts by considering the root problem (or the original problem with the complete feasible region) and applying the lower and upper bounding procedures. If the bounds match, then an optimal solution has been found and the procedure terminates. Otherwise, the feasible region is divided into two or more sub-problem partitions. The algorithm is applied recursively to the sub-problems. If an optimal solution is found to a subproblem, it is a feasible solution to the full problem, but not necessarily be a global optimal solution. If the lower bound for a node exceeds the best known feasible solution, no globally optimal solution can exist in the sub-space of the feasible region represented by that particular node. Therefore, the node can pruned (removed from consideration). The search proceeds until all nodes have been solved or pruned, or until some specific threshold is met.

Examples of branch-and-bound algorithms used in bioinformatics include computational assignment of protein backbone NMR peaks[31] and matching protein structures[32].

2.1.3. DYNAMIC PROGRAMMING ALGORITHMS

Dynamic programming algorithms solve complex problems by breaking them down into simpler steps. It is suitable to solve problems that exhibit the properties of overlapping subproblems and optimal substructure. A problem that can be broken down into subproblems, which are reused repeatedly, indicates that the problem has overlapping subproblems. Whereas, a problem with optimal substructure mean that an optimal solution can be constructed efficiently from optimal solutions to its subproblem.

Examples of dynamic programming algorithms used in bioinformatics include Smith-Waterman[33] and Needleman-Wunsch[34] for sequence alignment and Nussinov[35] and Zuker-Stiegler[36] for RNA folding.

2.1.4. GREEDY ALGORITHMS

Greedy algorithms make the locally optimal choice at each iteration with the hope of finding the global optimum. They make whatever choice seems best at the moment, without regard for future consequences. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. When the algorithm terminates, the local optimum is hopefully equal to the global minimum. If this is the case, then the algorithm is correct. Otherwise, the algorithm has produced a sub-optimal solution.

Examples of greedy algorithms used in bioinformatics include G-PRIMER[37] and GreedyEM[38].

2.1.5. DIVIDE AND CONQUER ALGORITHMS

As the name implies, a divide and conquer algorithm has two distinct phases, i.e. a divide phase and a conquer phase. In the divide phase, the algorithm splits the problem into smaller problem instances and solves them independently. The solutions of these smaller problems instances are combined into a solution of the original problem in the conquer phase.

The divide and conquer approach is similar to dynamic programming in that the solution of a large problem depends on a previously obtained solutions of sub-problems. The significant difference, however, is that sub-problems of the divide and conquer approach must be completely separate and can be solved independently.

Example of divide and conquer algorithms used in bioinformatics include a multiple alignment algorithm in [39] and [40].

2.1.6. MACHINE LEARNING ALGORITHMS

Machine learning approaches are best suited for areas where there is a large amount of data but little theory[41]. Machine learning algorithms try to build a model from training data by deriving important insights about the parameter, which is often hidden. As the amount of training data increases, the accuracy of the machine learning algorithm typically increases as well. The parameters learned during training represents knowledge, while application of the algorithm to new data represents the algorithm's use of that knowledge.

Examples of machine learning algorithms used in bioinformatics include identification of structurally conserved residues[42], Support Vector Machine (SVM)-based MiRTif[43] and GIST[44].

2.1.7. HEURISTIC ALGORITHMS

Heuristic algorithms do not guarantee that the optimal best solution will be found. Heuristic algorithms are typically used to solve problems with the following properties:

- Problems with large search spaces such that they cannot realistically be enumerated or searched exhaustively.
- There are no known methods for finding the best solution to the problems that do not employ a strategy that is fundamentally similar to exhaustive search.

Examples of heuristic algorithms used in bioinformatics include BLAST[45], FASTA[46], T-Coffee[47] and M-Coffee[48].

2.2. SEQUENCE ALIGNMENT

Sequence alignment is one of the most popular sequence analysis tasks, in which two or more sequences are compared by searching for a series of substrings that are in the same order in the sequences. It is utilized to infer a relationship between the sequences and also gives an impression on how close they are in terms of sequence similarity. Hence, it is essential for discovering functional, structural and evolutionary information in biological sequences.

A list of key issues that are related to sequence alignment are identified in [49]. These key issues are summarized as follows:

- What type of alignment should be considered?
- What scoring system is used?
- What algorithm is used to obtain the optimal (or good) scoring alignments?
- What statistical methods used to evaluate the significance of an alignment score?

2.2.1. TYPES OF ALIGNMENT

In general, sequence alignment can be categorized into two groups, i.e. pairwise sequence alignment and multiple sequence alignment.

2.2.1.1. Pairwise sequence alignment

Consider the following pair of DNA sequences: **ATAGAC** and **ATTAGGC**. At a glance they look very much alike and this becomes more obvious when they are aligned together, as shown below.

A−TAGAC
ATTAGGC

The differences lie in the extra T in the second sequence and a change from A to G in the second to last position. Note that a gap, marked with a "-" sign, is introduced in the first sequence in order to allow the bases before and after the gap to align perfectly. This is an example of a pairwise sequence alignment.

A pairwise sequence alignment is defined as an alignment of two sequences to determine how similar they are. In most sequence similarity calculations, a similarity score is inferred from the alignment. Gap insertions are allowed until the resulting sequences are of the same size and the alignment must obey the restriction that gaps cannot appear in the same position in both sequences. The example above satisfies the definition of an alignment.

Ideally, the alignment of two sequences should be in agreement with their evolution, i.e. the patterns of descent as well as molecular structural and functional evolution[50]. Unfortunately, the evolutionary traces are often very difficult to detect, e.g. amino acid mutations, insertions and deletions of residues, transposed gene segments and the like can blur the ancestral relationship beyond recognition. In the absence of observed evolutionary traces, pairwise sequence alignment is regarded as mimicking evolution best when the minimum number of mutations is used to arrive at one sequence to the other. An approximation of this is to find the highest similarity value determined from summing substitution scores along matched residue pairs minus any insertion/deletion penalties. Such alignment is generally called the optimal alignment.

Unfortunately, testing all possible alignments, including the insertion of a gap at each position of each sequence is unfeasible. For example, 10^{88} possible alignments exists of a pairwise sequence alignment of 300 amino acid[33]. The number of calculations

managed to be reduced greatly by introducing gaps as assigned scoring values such that they can be treated in the same manner as the mutation of one residue to another. The technique to calculate the highest scoring or optimal alignment, generally known as the dynamic programming (DP) technique, has been introduced by Needleman and Wunsch[34] in 1970.

There are two basic types of sequence alignment: global alignment and local alignment. Global alignment implies the matching of sequences over their complete lengths, whereas with local alignment the sequences are aligned only over the most similar parts of the sequences, carrying the clearest trace of evolutionary relationship. It is no always clear which of the two alignments (global or local) is biologically the most meaningful. In general, where there is a large difference in the lengths of two sequences to be compared, local alignment should be included in the analysis.

The first pairwise algorithm for local alignment was developed by Smith and Waterman[33] in 1981 as an adaptation of the algorithm of Needleman and Wunsch. The Smith-Waterman technique selects the most similar region in each of the two sequences, which are then aligned. In 1987, Waterman and Eggert[51] generalized the local alignment routine by devising an algorithm that allows the calculation of user-defined number of top-scoring local alignments instead of only the optimal local alignment.

2.2.1.2. Multiple sequence alignment

Multiple sequence alignment is an extension of pairwise alignment to incorporate more than two sequences at a time. Multiple alignment methods try to align all of the sequences in a given query set. Multiple alignments are often used in identifying conserved sequence regions across a group of sequences hypothesized to be

evolutionarily related. Ideally, in order to generate an accurate multiple alignments, in-depth knowledge of the evolutionary and structural relationships within the family would have to be utilized. However, these information are often lacking or difficult to use. General empirical models of protein evolution[52] are widely used instead, but these can be difficult to use when sequences are less than 30% identical[53]. Furthermore, mathematically sound methods for carrying out alignments using the models can be extremely demanding in computer resources for more than a handful sequences[54]. Therefore, heuristic methods have been developed to be able to cope with practical datasets.

Progressive alignment method[55, 56] is the most commonly used heuristic method. It adds sequences one by one to the existing alignment to build a new alignment. Many implementations determine the order of the sequences to be added to the new alignment by using an approximation of a phylogenetic tree, which is often called a guide tree. The guide tree is constructed using the similarity of all possible pairs of sequences stored in the distance matrix. The disadvantage of the progressive alignment method is that it suffers from greediness. Errors made in the first alignments during the progressive procedure cannot be corrected later. Global sequence weighting schemes[57, 58] are introduced to minimize such alignment errors . However, such schemes carry the risk of propagating rather than reducing error when used in progressive multiple alignment strategies[59]. ClustalW[58, 60] is the most widely used progressive alignment implementation. Up to 2009, ClustalW has over 26,000 citations in the ISI Web of Science. ClustalX[61] is the graphical version of ClustalW. Other multiple sequence alignment methods include MUSCLE[62], T-Coffee[47], and PRALINE[63].

2.2.2. SCORING SCHEME

Aligning two or more sequences can produce multiple possible results. In order to determine which of those possible alignments are optimal alignments, a scoring scheme is required. In general, scoring schemes used in sequence alignment consists of substitution matrix and gap penalties.

2.2.2.1. Substitution Matrix

Substitution matrix consists of substitution score terms for each aligned residue pair. The substitution score *s(x,y)* indicates the scores of aligning residue *x* with residue *y*. In the case of DNA, $x,y \in$ {A, G, C, T} and in the case of proteins, $x,y \in$ {A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V}.

Various popular substitution matrices utilized in sequence alignments include:

2.2.2.1.1. Unitary Scoring Matrix

Early sequence alignment programs used unitary scoring matrix. A unitary matrix scores all residue matches as well as penalizes all mismatches with the same value, as shown in equation 1, where *c* and *d* are constants.

$$s(x,y) = \begin{cases} c, if\ (x = y) \\ d, if\ (x \neq y) \end{cases}$$

Equation 1. Unitary scoring matrix equation

Although this scoring is sometimes appropriate for DNA and RNA comparisons, for protein alignments using a unitary matrix amounts to proclaiming ignorance about protein evolution and structure.

2.2.2.1.2. Log-odds ratio

Log-odds ratio substitution matrix consists of individual scores *s(x,y)* for each aligned pair of residues. The value of *s(x,y)* is defined as the *odds ratio* between two probabilities that describe the probability that some residue *x* will change to residue *y* over time, as shown in equation 2.

$$s(x, y) = \log \frac{M_{xy}}{P_y}$$

Equation 2. Log-odds ratio scoring matrix equation

where M_{xy} is the probability that we expect to observe residues *x* and *y* aligned in homologous sequence alignments and P_y is the probability we expect to observe residue *y* on average in a random sequence.

2.2.2.1.3. Point Accepted Mutation (PAM)

The PAM[64] matrix was developed by Margaret Dayhoff in the 1978. It is calculated by observing the differences in closely related proteins. The PAM1 matrix estimates what rate of substitution would be expected if 1% of the amino acids had changed. The PAM1 matrix is used as the basis for calculating other matrices by assuming that repeated mutations would follow the same pattern as those in the PAM1 matrix, and multiple substitutions can occur at the same site. Using this logic, Dayhoff derived matrices as high as PAM250. Example of the PAM250 matrix is shown in Figure 4.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	0	0	-8
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	0	-1	-8
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	1	0	-8
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	3	-1	-8
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-3	-8
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	3	-1	-8
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	3	-1	-8
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	0	-1	-8
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	2	-1	-8
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	-2	-1	-8
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	-3	-1	-8
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	0	-1	-8
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	-2	-1	-8
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	-5	-2	-8
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	0	-1	-8
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	0	0	-8
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	0	-8
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-6	-4	-8
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-4	-2	-8
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	-2	-1	-8
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	2	-1	-8
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	-1	-8
X	0	-1	0	-1	-3	-1	-1	-1	-1	-1	-1	-1	-1	-2	-1	0	0	-4	-2	-1	-1	-1	-1	-8
*	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	1

Figure 4. PAM250 matrix

2.2.2.1.4. Block Substitution Matrix (BLOSUM)

Dayhoff's methodology of comparing closely related species turned out not to work very well for aligning evolutionarily divergent sequences. Sequence changes over long evolutionary time scales are not well approximated by compounding small changes that occur over short time scales. The BLOSUM[52] series of matrices rectifies this problem. Henikoff and Henikoff constructed these matrices using multiple alignments of evolutionarily divergent proteins. The probabilities used in the matrix calculation are computed by looking at "blocks" of conserved sequences found in multiple protein alignments. These conserved sequences are assumed to be of functional importance within related proteins. To reduce bias from closely related sequences, segments in a block with a sequence identity above a certain threshold were clustered giving weight 1

to each such cluster. For the BLOSUM62 matrix, this threshold was set at 62%. Pair frequencies were then counted between clusters, hence pairs were only counted between segments less than 62% identical. One would use a higher numbered BLOSUM matrix for aligning two closely related sequences and a lower number for more divergent sequences. Example of the BLOSUM62 matrix is shown in Figure 5.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	6	-2	-2	-3	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	2	0	-4	-3	0	-2	-1	-1	-6
R	-2	8	-1	-2	-5	1	0	-3	0	-4	-3	3	-2	-4	-3	-1	-2	-4	-3	-4	-2	0	-2	-6
N	-2	-1	8	2	-4	0	0	-1	1	-5	-5	0	-3	-4	-3	1	0	-6	-3	-4	5	0	-2	-6
D	-3	-2	2	9	-5	0	2	-2	-2	-5	-5	-1	-5	-5	-2	0	-2	-6	-5	-5	6	1	-2	-6
C	-1	-5	-4	-5	13	-4	-5	-4	-4	-2	-2	-5	-2	-4	-4	-1	-1	-3	-4	-1	-5	-5	-3	-6
Q	-1	1	0	0	-4	8	3	-3	1	-4	-3	2	-1	-5	-2	0	-1	-3	-2	-3	0	5	-1	-6
E	-1	0	0	2	-5	3	7	-3	0	-5	-4	1	-3	-5	-2	0	-1	-4	-3	-4	1	6	-1	-6
G	0	-3	-1	-2	-4	-3	-3	8	-3	-6	-5	-2	-4	-5	-3	0	-2	-4	-5	-5	-1	-3	-2	-6
H	-2	0	1	-2	-4	1	0	-3	11	-5	-4	-1	-2	-2	-3	-1	-3	-4	3	-5	-1	0	-2	-6
I	-2	-4	-5	-5	-2	-4	-5	-6	-5	6	2	-4	2	0	-4	-4	-1	-4	-2	4	-5	-5	-2	-6
L	-2	-3	-5	-5	-2	-3	-4	-5	-4	2	6	-4	3	1	-4	-4	-2	-2	-2	1	-5	-4	-2	-6
K	-1	3	0	-1	-5	2	1	-2	-1	-4	-4	7	-2	-5	-2	0	-1	-4	-3	-3	-1	1	-1	-6
M	-1	-2	-3	-5	-2	-1	-3	-4	-2	2	3	-2	8	0	-4	-2	-1	-2	-1	1	-4	-2	-1	-6
F	-3	-4	-4	-5	-4	-5	-5	-5	-2	0	1	-5	0	9	-5	-4	-3	1	4	-1	-5	-5	-2	-6
P	-1	-3	-3	-2	-4	-2	-2	-3	-3	-4	-4	-2	-4	-5	11	-1	-2	-5	-4	-4	-3	-2	-2	-6
S	2	-1	1	0	-1	0	0	0	-1	-4	-4	0	-2	-4	-1	6	2	-4	-3	-2	0	0	-1	-6
T	0	-2	0	-2	-1	-1	-1	-2	-3	-1	-2	-1	-1	-3	-2	2	7	-4	-2	0	-1	-1	-1	-6
W	-4	-4	-6	-6	-3	-3	-4	-4	-4	-4	-2	-4	-2	1	-5	-4	-4	16	3	-4	-6	-4	-3	-6
Y	-3	-3	-3	-5	-4	-2	-3	-5	3	-2	-2	-3	-1	4	-4	-3	-2	3	10	-2	-4	-3	-2	-6
V	0	-4	-4	-5	-1	-3	-4	-5	-5	4	1	-3	1	-1	-4	-2	0	-4	-2	6	-5	-4	-1	-6
B	-2	-2	5	6	-5	0	1	-1	-1	-5	-5	-1	-4	-5	-3	0	-1	-6	-4	-5	5	0	-2	-6
Z	-1	0	0	1	-5	5	6	-3	0	-5	-4	1	-2	-5	-2	0	-1	-4	-3	-4	0	5	-1	-6
X	-1	-2	-2	-2	-3	-1	-1	-2	-2	-2	-2	-1	-1	-2	-2	-1	-1	-3	-2	-1	-2	-1	-2	-6
*	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	1

Figure 5. BLOSUM62 matrix

2.2.2.2. Gap Penalties

Gaps are expected to be penalized in an alignment. The standard gap penalty *w(k)* associated with a gap of length *k* can either be given by a linear penalty or an affine penalty. Equation 3 shows the linear gap penalty equation.

$$w(k) = g \cdot k$$

Equation 3. Linear gap penalty equation

Affine gap penalty introduces the concept of gap open and gap extension penalty. Equation 4 shows the affine gap penalty equation, where *h* is the gap open penalty and *g* is the gap-extension penalty.

$$w(k) = h + g \cdot k$$

Equation 4. Affine gap penalty equation

2.2.3. ALIGNMENT ALGORITHMS

Below we introduce several basic types of alignment algorithms.

2.2.3.1. Global alignment: Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm[34] is a dynamic programming algorithm, that obtains the optimal global alignment between two sequences, allowing gaps. Gotoh[65] modified the algorithm to run at O(*mn*) complexity by considering affine gap penalties.

The main idea of this algorithm is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. Given a matrix *M* and two sequences **X** = {x_1, x_2, …, x_m} and **Y** = {y_1, y_2, …, y_n}, *M(i,j)* is the score of the best alignment between the segments $x_1 \ldots i$ up to x_i and $y_1 \ldots j$ up to y_j. Hence, *M*(0,0) is initialized to be 0 and *M(i,j)* is then build recursively.

The value of *M(i,j)* could only be calculated if the values of *M(i*-1,*j*-1*)*, *M(i*-1,*j)* and *M(i,j*-1*)* are known. There are three possible ways that the best score *M(i,j)* of an alignment up to x_i, y_j could be obtained:

- x_i is aligned to a gap, in which case *M(i,j)* = *M(i*-1,*j)* – *g*
- y_j is aligned to a gap, in which case *M(i,j)* = *M(i,j*-1*)* – *g*

- x_i is aligned to y_j, in which case $M(i,j) = M(i\text{-}1,j\text{-}1) + sub(x_i,y_j)$

where g is the gap penalty and $sub\ (x_i,y_j)$ is the substitution score of aligning residues x_i and y_j. The best score up to (i,j) will be the largest of these three options. Therefore, we have following equation:

$$M(i,j) = \max \begin{cases} M(i\text{-}1,j\text{-}1) + sub(x_i,y_j), \\ M(i-1,j) - g, \\ M(i,j-1) - g \end{cases}$$

Equation 5. Needleman-Wunsch equation

Initialization values are given as the following: for $0 \le i \le m$, $M(i, 0) = -i \cdot g$ and for $0 \le j \le n$, $M(0, j) = -j \cdot g$. Equation 5 is repeatedly applied to fill in the matrix of $M(i,j)$ values, calculating the value in the bottom right-hand corner of each square of four cells from one of the other three values (above-left, above, or left), as shown in Figure 6.

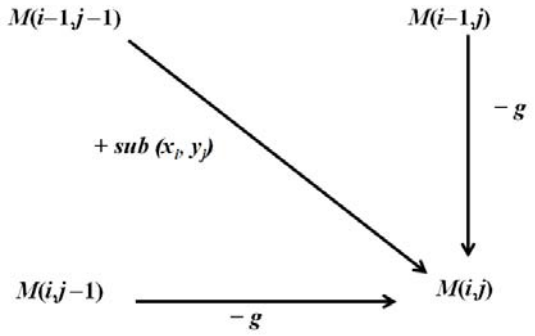

Figure 6. Data dependency in Needleman-Wunsch algorithm

The most bottom right cell of the matrix $M(m,n)$ is the score of the best global alignment for an alignment of **X** and **Y**. A *traceback* procedure is needed to determine the actual alignment(s) from the corresponding score. The traceback for the Needleman-Wunsch algorithm starts from the cell with the best score $M(m, n)$ to $M(0, 0)$.

2.2.3.2. Local alignment: Smith-Waterman algorithm

A lot of biological problems, e.g. search for a common domain between two protein sequences, comparison of extended sections of genomic DNA sequences and similarity detection between two very divergent sequences, require us to look for the best alignment between subsequences. Such alignment is called local alignment. The Smith-Waterman algorithm[33] is a dynamic programming algorithm, that obtains the optimal local alignment between two sequences.

The algorithm is closely related to the global alignment algorithm. There are, however, two main differences. The first is that the value of *M(i,j)* will be 0 if its value is 0 or less. Taking the option 0 basically corresponds to starting a new alignment. If the best alignment up to a certain point reaches a negative score, a new alignment is preferred, rather than continue and extend the old one. This is reflected in Equation 6.

$$M(i,j) = \max \begin{cases} 0, \\ M(i\text{-}1,j\text{-}1) + sub(x_i, y_j), \\ M(i-1,j) - g, \\ M(i,j-1) - g \end{cases}$$

Equation 6. Smith-Waterman equation

The second difference is that the best score of the alignment is no longer *M(m,n)*, but it is the cell with the highest value of *M(i,j)* over the whole matrix. That particular cell indicates where the alignment ends.

Further details of the Smith-Waterman algorithm will be elaborated in Chapter 4.

2.2.3.3. Algorithms with affine gap penalty

The simplest gap model implemented in most algorithms is a simple multiplication of the length with the gap penalty. This type of model, however, is not ideal for biological sequences. In the real world, when gaps do occur, they are more likely to have a large gap, rather than many small gaps. For example, a biological sequence is much more likely to have one big gap of length *k*, due to a single insertion or deletion event, than it is to have *k* small gaps of length 1.

To account for this tendency, affine gap penalty is introduced. Affine gap penalty consists of a gap opening penalty, α, and a gap extension penalty, β. A gap of length *k* would then have an affine gap penalty $w(k) = \alpha + (k-1)\beta$. The value of α and β are usually always negative because gap extension are encouraged, rather than gap introduction.

2.2.3.4. Heuristic alignment algorithms

All the alignment algorithms described so far produce optimal result. However, they are not the fastest methods and in some cases, speed is an issue. Heuristic alignment algorithms offer fast solutions with a trade off of accuracy and sensitivity. The goal of these methods is to search as small a fraction as possible of the cells in the dynamic programming matrix, while still looking at all the high scoring alignments. Two of the best-known algorithms are the Basic Local Alignment Search Tool (BLAST)[45] and FAST-All (FASTA)[46].

2.2.3.4.1. BLAST

The BLAST package[45] provides programs for finding high scoring local alignments between a query sequence and a database. The sequences can either be DNA or protein sequences. The main idea of the BLAST algorithm is that true match alignments are very likely to contain within them a short stretch of identities or very high scoring matches. Such short stretches are called seeds, from which they are extended out in search of a good longer alignment.

Table 1. Traditional BLAST Programs

Program	Query	Database	Typical Usage
BLASTN	Nucleotide	Nucleotide	Mapping oligonucleotides, cDNAs and PCR products to a genome; screening repetitive elements; annotating genomic DNA; vector clipping
BLASTP	Protein	Protein	Identifying common regions between proteins; collecting related proteins for phylogenetic analysis
BLASTX	Nucleotide translated into protein	Protein	Finding protein-coding genes in genomic DNA; determining if a cDNA corresponds to a known protein
TBLASTN	Protein	Nucleotide translated into protein	Identifying transcripts from multiple organisms; mapping a protein to genomic DNA
TBLASTX	Nucleotide translated into protein	Nucleotide translated into protein	Cross-species gene prediction at the genome or transcript level

BLAST creates a *k*-length word list of the query sequence, with the default value of $k = 3$ for protein sequences and $k = 11$ for DNA sequences. It then scans through the database and whenever a word in word list is found to have a score higher than a pre-determined threshold, the possible match is extended as an ungapped alignment in both directions, stopping at the maximum scoring extension. Five traditional BLAST programs are BLASTN, BLASTP, BLASTX, TBLASTN, TBLASTX, as shown in Table 1[66].

New versions of BLAST have become available, e.g. WU-BLAST[67, 68] which provide gapped alignments, PSI-BLAST[69] which is more sensitive in picking up distant evolutionary relationships, mpiBLAST[70] which is an open-source parallel BLAST, G-BLAST which is a grid-based solution[71] and FSA-BLAST[72, 73] which has algorithmic improvements.

2.2.3.4.2. FASTA

The FASTA package[46] is another widely used heuristic sequence algorithm. Originally, FASTA was introduced as FASTP and was designed for protein sequence similarity searching. The current FASTA package contains programs for protein-protein, DNA-DNA, protein-translated DNA (with frameshifts), and ordered or unordered peptide searches. It uses a four-step approach to find local high scoring alignments, starting from exact short word matches, through maximal scoring ungapped extensions. There is a trade off between speed and sensitivity in the choice of parameter *ktup*: the higher the value of *ktup*, the faster the algorithm will run, albeit with less accuracy (more significant misses).

The first step identifies the regions of highest density in each sequence comparison. By using a look up table, all identically matching words of length *ktup* between any two

sequences are located and regions with many mutually supporting word matches are identified. The default value of *ktup* is 1 or 2 for protein sequences and 4 or 6 for DNA sequences. The second step extends the exact word matches to find maximal scoring ungapped regions. The third step checks if any if these ungapped regions can be joined by a gapped region, allowing for gap costs. The final step realigns the highest scoring candidate matches in a database search using a dynamic programming algorithm. This step, however, is limited to a subregion of the dynamic programming matrix forming a band around the potential heuristic match.

2.3. PARALLEL COMPUTATION MODEL AND PARALLEL ARCHITECTURES

Traditionally, software has been written for serial computation to be run on a single computer having a single Central Processing Unit (CPU). This type of programming is called Sequential Programming, in which the problem is broken into a discrete series of instructions that are executed one after another and only one instruction may execute at any moment in time. Figure 7 illustrates how sequential programming execution is processed by the CPU.

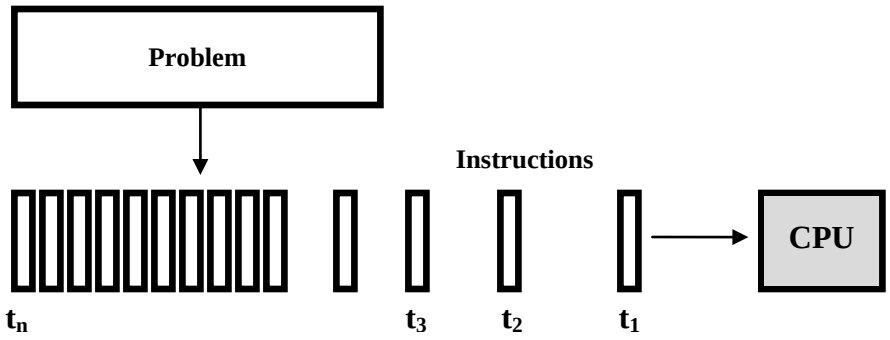

Figure 7. Sequential programming execution

In the simplest sense, parallel programming is the simultaneous use of multiple computing resources/CPUs to solve a computational problem. The problem is broken into discrete parts that can be solved concurrently. Each part is further broken down to a series of instructions, in which they are executed simultaneously on different CPUs. In other words, parallel programming focuses on partitioning the overall problem into separate tasks, allocating tasks to processors and synchronizing the tasks to get meaningful results. Figure 8 illustrates how a parallel program is executed by multiple CPUs.

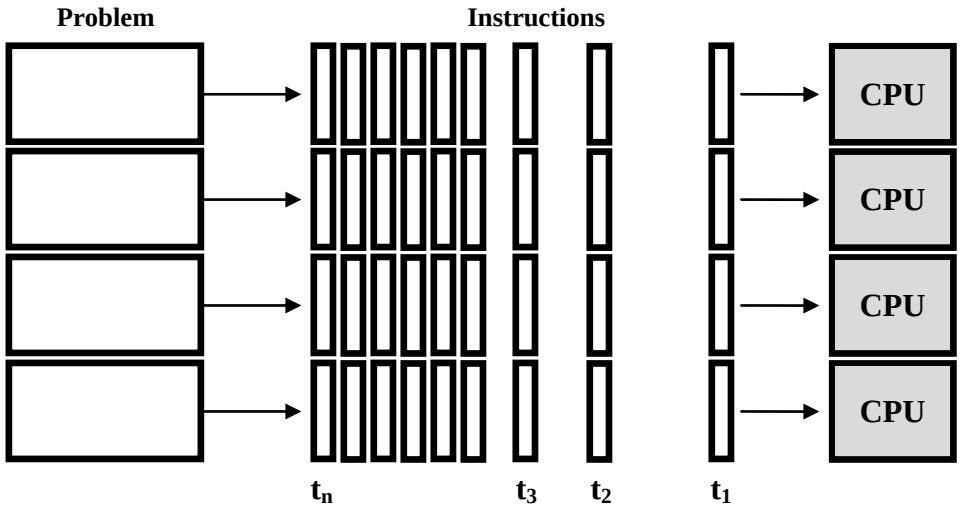

Figure 8. Parallel programming execution

The main advantage of parallel programming lies with its overall ability to reduce the execution time required needed to obtain the solution, as well as the possibility of solving larger problems. Another advantage also includes the possibility of using non-local resources on a local area network, or even the internet when local compute resources are scarce. On the other hand, the limitation of parallel programming lies with the overhead

communication time needed for synchronization and transferring of data between processors.

2.3.1. TERMINOLOGY

Below are some frequently used terms in parallel computing[74]:

2.3.1.1. Speed-up

The run time of the sequential program divided by run time of the parallel program. Speed-up *s* can be expressed as the following equation:

$$s = \frac{t_s}{t_p}$$

Equation 7. Speed-up equation

Where:

t_s is the run time of the sequential program

t_p is the run time of the parallel program

2.3.1.2. Parallel Overhead

The extra work associated with parallel version compared to its sequential code, mostly the extra CPU time and memory space requirements from synchronization, data communications, parallel environment creation and cancellation, etc.

2.3.1.3. Synchronization

The coordination of simultaneous tasks to ensure correctness and avoid unexpected race conditions.

2.3.1.4. Efficiency

The execution time using a single processor divided by the quantity of the execution time using a multiprocessor and the number of processors. Efficiency *e* can be expressed as the following equation:

$$e = \frac{t_s}{n \cdot t_p}$$

Equation 8. Efficiency equation

Where:

t_s is the run time of the sequential program

t_p is the run time of the parallel program

n is the number of processors used

Combining equation 7 and 8, the efficiency equation can be simplified as follows:

$$e = \frac{s}{n}$$

Equation 9. Simplified efficiency equation

2.3.1.5. Scalability

A parallel system's ability to gain proportionate increase in parallel speedup with the addition of more processors.

2.3.1.6. Task

A logically high level, discrete, independent section of computational work. A task is typically executed by a processor as a program

2.3.2. VON NEUMANN ARCHITECTURE

For over 40 years, virtually all computers have followed a common machine model known as the von Neumann architecture[75]. A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory. Figure 9 shows the block diagram of the von Neumann architecture.

Characteristics of von Neumann architectures:

- Memory is used to store both program and data instructions
- Program instructions are coded data which tell the computer to do something
- Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then sequentially performs them.

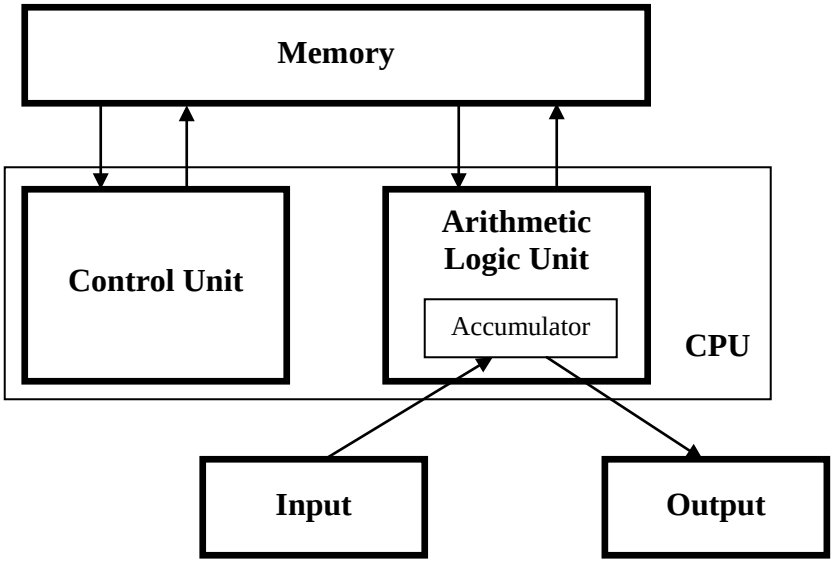

Figure 9. Block diagram of the von Neumann architecture

2.3.3. FLYNN'S CLASSICAL TAXONOMY

Flynn's Taxonomy is one of the best-known classification schemes in parallel computing[76]. It classifies multi-processor computer architectures based on the two independent dimensions of Instruction and Data axes. Each of these dimensions can have only one of two possible states: Single or Multiple. The focus is on the multiplicity of hardware used to manipulate the instruction and data streams[77, 78]. Flynn's Taxonomy is illustrated in Table 2.

Table 2. Flynn's Taxonomy

	Single Instruction	**Multiple Instruction**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

2.3.3.1. Single Instruction, Single Data (SISD)

SISD refers to computers with a single instruction stream and a single data stream, as shown in Figure 10. Single instruction means only one instruction stream is being acted on by the CPU during any one clock cycle, while single data means only one data stream is being used as input during any one clock cycle. SISD has a deterministic execution. Examples of SISD are uniprocessors and single CPU workstations and mainframes.

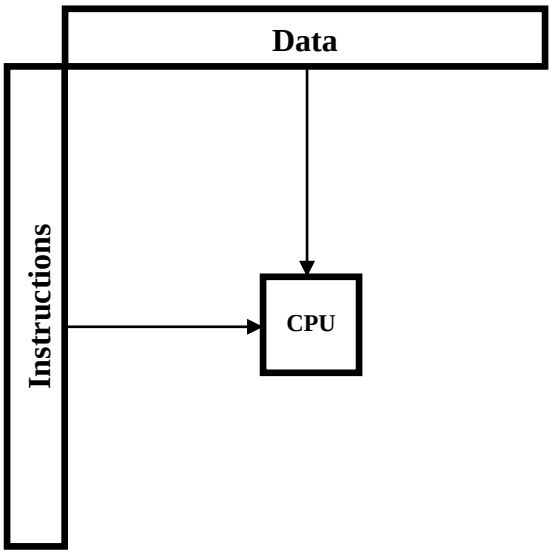

Figure 10. Single Instruction, Single Data (SISD)

2.3.3.2. Single Instruction, Multiple Data (SIMD)

As illustrated in Figure 11, SIMD refers to computers with a single instruction stream but multiple data streams. In other words, all processing units execute the same instruction at any given clock cycle but each processing unit can operate on a different data element. This type of architecture typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units. SIMD is best suited for specialized problems characterized by a high degree of regularity. Examples of SIMD are processor arrays and pipelined vector processors.

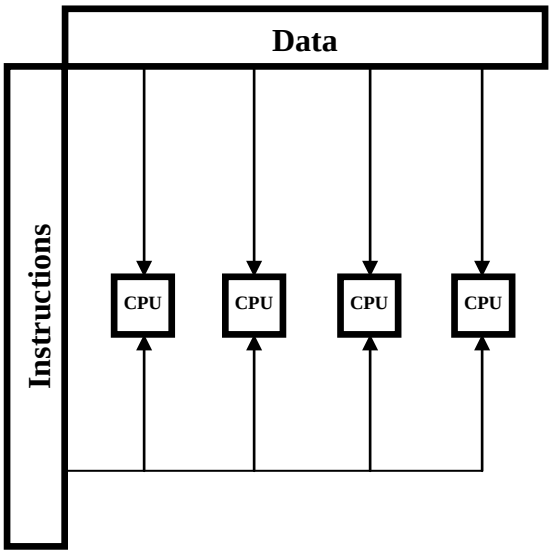

Figure 11. Single Instruction, Multiple Data (SIMD)

2.3.3.3. Multiple Instruction, Single Data (MISD)

MISD refers to computers with a multiple instruction streams but only a single data stream, as shown in Figure 12.

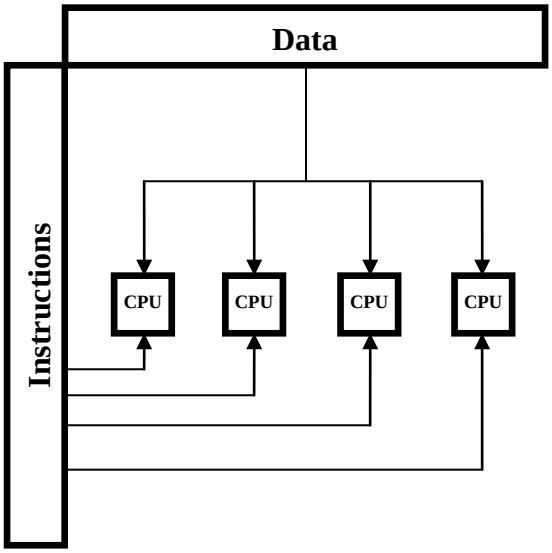

Figure 12. Multiple Instructions, Single Data (MISD)

MISD computer is a pipeline of multiple independently executing functional units operating on a single stream of data, forwarding results from one functional unit to the

next. Some conceivable application uses of MISD are multiple frequency filters operating on a single signal stream and multiple cryptography algorithms attempting to crack a single coded message. Systolic arrays are examples of MISD[76].

2.3.3.4. Multiple Instruction, Multiple Data (MIMD)

As illustrated in Figure 13, MIMD refers to computers with a multiple instruction streams and multiple data streams. In other words, every processor may be working with a different data stream and execution can be synchronous or asynchronous, deterministic or non-deterministic. The TOP500 table, which shows the 500 most powerful commercially available computer systems, indicates that as of 2009, the entire TOP500 supercomputers are based on MIMD architecture [79]. Examples of MIMD are supercomputers and grids.

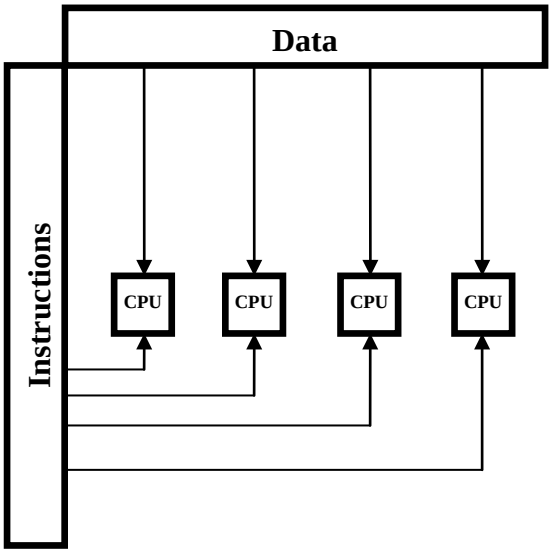

Figure 13. Multiple Instruction, Multiple Data (MIMD)

2.4. ACCELERATOR TECHNOLOGIES IN HIGH PERFORMANCE COMPUTING

The recent emergence of accelerator technologies such as Field-Programmable Gate Arrays (FPGAs), Graphics Processing Unit (GPUs) and multi-core processors have made it possible to achieve an excellent improvement in execution time for many bioinformatics applications, compared to current general-purpose platforms. We review those accelerator technologies in the following section.

2.4.1. VLSI

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. Currently, the term is no longer as common as it once was, as chips have increased in complexity into billions of transistors. Early VLSI implementation of bioinformatics application include BioScan[80], Biological Information Signal Processor (BISP)[81] and Systolic Accelerator for Molecular Biological Application (SAMBA)[82].

BioScan[80] accelerates the identification of similar segments for DNA or protein sequences without allowing gap. It contains a total number of 12,992 processors consisting of 16 chips of 812 1-bit processors each. The database scanning has a limited query sequence of 12,992 characters. BioScan does not support dynamic programming algorithms.

BISP[81] implements a modified version of the Smith-Waterman algorithm and allows many parameters to be set. It consists of 256 16-bit processors and a programmable processor Motorola 68020, making possible the computation of unlimited sequence length.

SAMBA[82] implements a parameterized Smith-Waterman algorithm. By setting different parameters, local or global comparisons can be performed, with or without gap penalty. The complete SAMBA system consists of a workstation, a systolic array of 128 full custom hardwired 12-bit processors, and an FPGA-based interface i.e. PeRLe-1 board.

In general, the early VLSI implementations provide a respectable speed-up for the state of technology at that time. However, they are dwarfed by the implementations on current accelerator technologies such as FPGA, GPU and Cell/BE. Another drawback to the early VLSI implementations is that the core of the system relies on Application Specific Integrated Circuit (ASIC) component, in which the chip is devoted to a single function (or a restricted class of functions). Once designed and fabricated, it cannot be modified and is not flexible to program.

2.4.2. FPGA

FPGA is a semiconductor device that can be configured by the customer or designer after manufacturing. To define the behaviour of the FPGA, the user provides a hardware description language (HDL), such as VHDL and Verilog HDL, or a schematic implementation. The HDL form might be easier to work with when handling large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. On the other hand, schematic entry can allow for easier

visualization of an implementation. They can be used to implement any logical function that an ASIC could perform, but the ability to update the functionality after manufacturing offers advantages for many applications.

The underlying architecture of the FPGA is well-suited for parallel processing. FPGAs contain programmable logic components called *logic blocks*, and a hierarchy of reconfigurable interconnects that allow the blocks to be *wired together*, somewhat similar to a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like *AND* and *XOR*. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

Although FPGAs are flexible, their configuration has to be changed for each single algorithm. Thus, making it more complicated in general compared to writing codes for programmable architectures. Examples of bioinformatics application on FPGAs include [83-88].

2.4.3. GPU

Programmable GPUs have received attention from the scientific computing community since their introduction on the market in 2000[89]. Architecturally, modern GPUs implement what is referred to as a streaming processor[90, 91]. This architecture gains its speed by devoting significantly more chip real estate to the computational engine than a conventional CPU. Furthermore, its attractive price to performance ratio and the fact that GPUs are now commodity items found in almost all computers makes it an appealing alternative for high performance computing. Examples of bioinformatics application on GPUs include [87, 92-97].

However, the traditional general-purpose computing on the GPU (GPGPU) development is based on graphics function library, for example OpenGL and Direct 3D, which makes the GPU used only by the professional people familiar with graphics API, and brings many inconveniences to the common users. The two major GPU vendors, NVIDIA and AMD, recently announced their new developing platforms Compute Unified Device Architecture (CUDA)[98] and Close to the Metal (CTM)[99] , respectively. Unlike previous GPU programming models, these are proprietary approaches designed to allow a direct access to their specific graphics hardware. Therefore, there is no compatibility between the two platforms. CUDA is an extension of the C programming language; CTM is a virtual machine running proprietary assembler code. However, both platforms overcome some important restrictions on previous GPGPU approaches, in particular those set by the traditional graphics pipeline and the relative programming interfaces like OpenGL and Direct3D.

CUDA is an extension of C/C++ which enables users to write scalable multi-threaded programs for CUDA-enabled GPUs. CUDA programs can be executed on GPUs with NVIDIA's Tesla unified computing architecture[100]. The emergence of CUDA allows a C-like development environment to programmers as a C compiler is used to compile programs, and the shader languages are replaced with C language and some CUDA extended libraries. This change means that programmers do not need to map programs into graphics APIs, making GPGPU program development more flexible and efficient.

CUDA programs contain a sequential part, called a *kernel*. The *kernel* is written in conventional scalar C-code. It represents the operations to be performed by a single thread and is invoked as a set of concurrently executing threads. These threads are

organized in a hierarchy consisting of so-called thread blocks and grids. A *thread block* is a set of concurrent threads and a *grid* is a set of independent thread blocks. Each thread has an associated unique ID (*threadIdx*, *blockIdx*) ∈ {0,..., *dimBlock*-1} × {0,..., *dimGrid*-1}. This pair indicates the ID within its thread block (threadIdx) and the ID of the thread block within the grid (*blockIdx*). Similar to MPI processes, CUDA provides each thread access to its unique ID through corresponding variables. The total size of a grid (*dimGrid*) and a thread block (*dimBlock*) is explicitly specified in the kernel function-call: kernel<<<*dimGrid*, *dimBlock*, *other configurations*>>> (parameter list);

The hierarchical organization into blocks and grids has implications for thread communication and synchronization. Threads within a thread block can communicate through a per-block shared memory (PBSM) and may synchronize using barriers. However, threads located in different blocks cannot communicate or synchronize directly. Besides the PBSM, there are four other types of memory: per-thread private local memory, global memory for data shared by all threads, texture memory and constant memory. Texture memory and constant memory can be regarded as fast read-only caches.

The Tesla architecture supports CUDA applications using a scalable processor array. The array consists of a number of streaming multiprocessors (SMs). Each SM contains eight scalar processors (SPs), which share a PBSM of size 16 KB. All threads of a thread block are executed concurrently on a single SM. The SM executes threads in small groups of 32, called warps, in single-instruction multiple-thread (SIMT) fashion. Thus, parallel performance is generally penalized by data-dependent conditional branches and improves if all threads in a warp follow the same execution path.

An important aspect of CUDA programming is the management of the memory spaces that have different characteristics and performances, as explained below[97].

- *Readable and writable global memory* is relatively large (typically 1 GB), but has high latency, low bandwidth, and is not cached. The effective bandwidth of global memory depends heavily on the memory access pattern, e.g. coalesced access generally improves bandwidth.
- *Readable and writable per-thread local memory* is of limited size (16 KB per thread) and is not cached. Access to local memory is as expensive as access to global memory and is always coalesced.
- *Read-only constant memory* is of limited size (totally 64 KB) and cached. The reading cost scales with the number of different addresses read by all threads. Reading from constant memory can be as fast as reading from a register (e.g. if all threads of a half-warp read the same address).
- *Read-only texture memory* is large (depending on the size of global memory) and is cached. Texture memory can be read from kernels using texture fetching device functions. Reading from texture memory is generally (not absolutely) faster than reading from global or local memory.
- *Readable and writable per-block shared memory* is fast on-chip memory of limited size (16 KB per block). Shared memory can only be accessed by all threads in a thread block. Shared memory is divided into equally-sized banks that can be accessed simultaneously by each thread. Accessing the shared memory is as fast as accessing a register as long as there are no bank conflicts.

- *Readable and writable per-thread registers* are the fastest memory to access but is of very limited size.

2.4.4. MULTI-CORE

Multi-core technology was first discussed in 1989[25]. Conceptually, multi-core architecture refers to a single processor package containing two or more processor execution cores or computational engines that deliver fully parallel execution of multiple software threads. The operating system treats each of its execution cores as a discrete processor, with all associated execution resources.

One of the ideas behind the movement to multi-core is parallelism. It is one of the best ways to address the issue of power while maintaining performance where higher data throughput may be achieved with lower voltage and frequency. The result is a larger transistor count, but overall lower power dissipation and power density. Instead of classifying based upon speed, one could classify products based upon the number of working cores or overall data throughput. The integration of multiple cores on a chip also allows lower interconnect latency and therefore higher bandwidth between cores than their discrete counterparts. Hence, microprocessor designers and manufacturers have turned to building chip multi-processors [26-29].

New chip architectures built for scaling out instead of scaling up will offer enhanced performance, reduced power consumption and more efficient simultaneous processing of multiple tasks. Multi-core systems embrace the scale out approach to performance. This architecture in essence reflects a *divide and conquer* strategy. By splitting the computational work performed by a single core in traditional microprocessors and among

multiple execution cores, a multi-core processor can perform more work within a given clock cycle. In other words, multi-core processors are able deliver higher performance and greater efficiency without the heat problems and other disadvantages experienced by single core processors run at higher frequencies to squeeze out more performance. By multiplying the number of cores in the processor, it is possible to tremendously increase computing resources, higher multithreaded throughput, and the benefits of parallel computing.

Multi-core architecture may take on a number of forms, namely homogeneous multi-core, heterogeneous multi-core and cluster of multi-core. Examples of bioinformatics application on multi-cores include [87, 101-107].

2.4.4.1. Homogeneous Multi-core

Homogeneous multi-core processor, also known as symmetric multi-core, is a processor which has multiple execution cores that are all exactly the same. Every single core has the same architecture and the same capabilities. An example of a homogeneous multi-core system is the Intel® Core™ i7 Processor[108]. Homogeneous multi-core processor usually uses a shared memory. In the case of Intel® Core™ i7 Processor, all the cores shares the L3 cache.

2.4.4.2. Heterogeneous Multi-core

Heterogeneous multi-core processor, also known as asymmetric multi-core, is a processor which has multiple execution cores, but the cores might be of different implementations. Each core will have different capabilities. An example of a heterogeneous multi-core system is the Cell Broadband Engine (Cell/BE)[109], which will be discussed in detail in

further chapters. Heterogeneous multi-core processor usually does not utilize shared memory. In the case of Cell/BE, the PPE has its own L1 and L2 cache, while the SPEs have their own respective Local Storage.

Recent research in heterogeneous multi-core processors has identified significant advantages over homogeneous multi-core processors in terms of power and throughput and in addressing the effects of Amdahl's law on the performance of parallel applications[110].

2.4.4.3. Cluster of Multi-core

A cluster of multi-core is a group of tightly coupled multi-core processors that work together closely so that in many respects they can be viewed as though they are a single entity. The components of a cluster are commonly, but not always, connected to each other through fast local area networks. Clusters are usually deployed to improve performance and/or availability over that provided by a single computer. An example of a cluster of multi-core system is a cluster of Playstation3[111].

3. CELL BROADBAND ENGINE

This chapter discusses the Cell/BE, its architecture, how it overcomes the three wall limitations, interprocessor communication and how to develop applications on the Cell/BE. Lastly, we categories and analyze programming techniques which are tailored for the Cell/BE.

3.1. INTRODUCTION

The Cell Broadband Engine[109], or often called as Cell/BE, is a single-chip heterogeneous multi-core processor which is developed by Sony, Toshiba and IBM. Although originally designed as a processor for Sony PlayStation3, Cell/BE has a general-purpose architecture, offering a unique assembly of thread-level and data-level parallelization options. It is operating at the upper range of existing processor frequencies (3.2 GHz for current models). Apart from that, the power consumption is also comparable to that of mobile processors.

3.2. CELL/BE ARCHITECTURE

The Cell/BE combines an IBM PowerPC Processor Element (PPE) and eight Synergistic Processor Elements (SPEs)[112]. An integrated high-bandwidth bus called the Element Interconnect Bus (EIB) connects the processors and their ports to external memory and I/O devices. The block diagram of the Cell/BE architecture is shown in Figure 14.

One type of processors in the Cell/BE is the PPE, which is a 64-bit Power Architecture core and contains a 64-bit general purpose register set (GPR), a 64-bit floating point register set (FPR), and a 128-bit Altivec register set. It is fully compliant with the 64-bit

Power Architecture specification and can run 32-bit and 64-bit operating systems and applications.

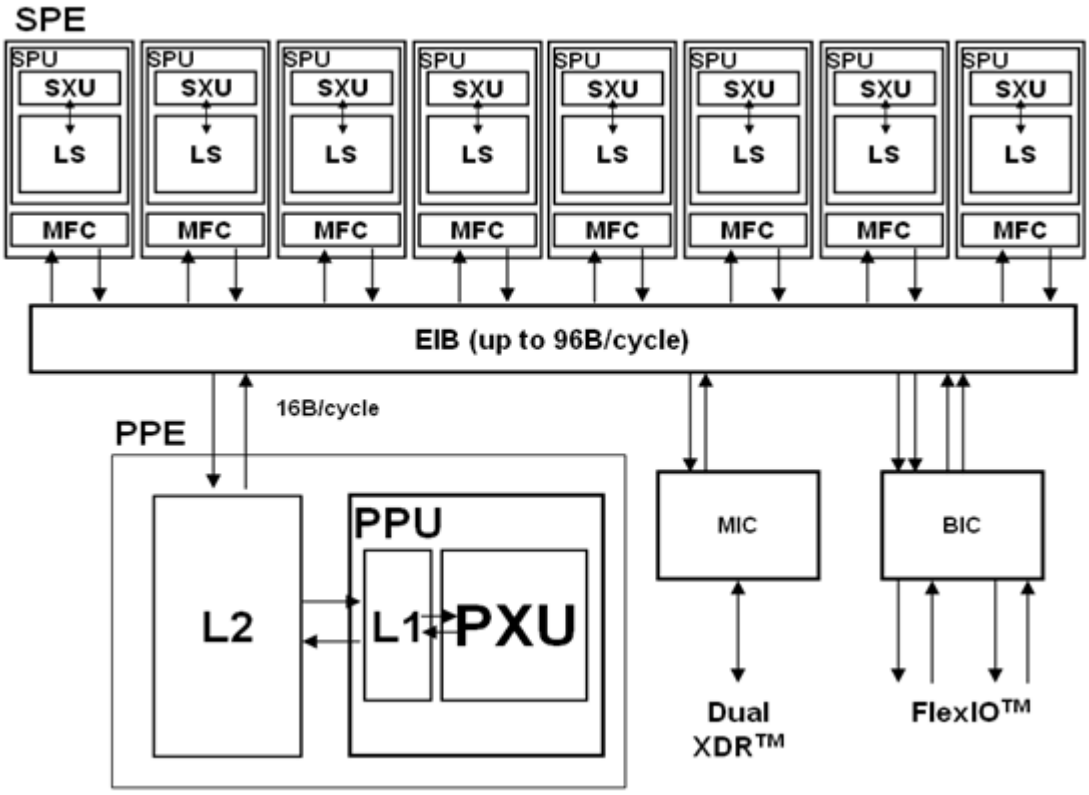

Figure 14. Block diagram of the Cell Broadband Engine Architecture

The other type is the SPEs, which on the other hand, are independent processors. Each SPE is able to run its own individual application programs. Each SPE consists of a processor implementationed for streaming workloads, a local memory, and a globally coherent DMA engine. The EIB is a 4-ring structure, and can transmit 96 bytes per cycle, for a bandwidth of 204.8 Gigabytes/second. The EIB can support more than 100 outstanding DMA requests.

From an architectural standpoint, parallelism exploitations at multiple levels on the Cell/BE are possible. Each chip has eight SPEs with two-way instruction-level

parallelism on each SPE. Furthermore, the SPE supports both scalar as well as SIMD computations[113]. Hence, it has a high peak performance because the SPE is simpler and more efficient than general purpose processors in terms of the micro and memory architecture[114]. The Cell/BE operates on a shared, coherent memory. In this respect, it extends current trends in PC and server processors. The most distinguishing feature of the Cell/BE lies within the variety of the processors it has, i.e. the PPE and the SPEs. Heterogeneous multi-core systems can lead to decreased performance if both the operating system and application are unaware of the heterogeneity[115]. However, intelligent scheduling processes show the potential for power savings and speedup on a heterogeneous multi-core system[116]. Further work showed that heterogeneous multi-core systems implementation targeting different cores to specific application classes can increase performance over that obtained by combining general-purpose cores[117]. The PPE is designed to run the operating system and, in many cases, the top-level control thread of an application, while the SPEs is optimized for compute-intensive applications, hence, providing the bulk of the application performance.

The SPE implements a Cell-specific set of SIMD instructions[118]. All single precision floating point operations on the SPU are fully pipelined, and the SPU can issue one single-precision floating point operation per cycle. Double precision floating point operations are partially pipelined and two double-precision floating point operations can be issued every six cycles. With all eight SPUs active and fully pipelined double precision FP operation, the Cell/BE is capable of a peak performance of 21.03 Gflops. In single precision FP operation, the Cell/BE is capable of a peak performance of 230.4 Gflops [119].

The SPE can access RAM through direct memory access (DMA) requests. The DMA transfers are handled by the MFC. All programs running on an SPE use the MFC to move data and instructions between local storage and main memory. Data transferred between local storage and main memory must be 128-bit aligned. The size of each DMA transfer can be at most 16 KB. DMA-lists can be used for transferring large amounts of data (more than 16 KB). A list can have up to 2,048 DMA requests, each for up to 16 KB. The MFC supports only DMA transfer sizes that are 1,2,4,8 or multiples of 16 bytes long. Kistler et al.[120] analyze the communication network of the Cell/B.E. processor and state that applications that rely heavily on random scatter and or gather accesses to main memory can take advantage of the high communication bandwidth and low latency.

3.3. OVERCOMING *THE THREE WALL* LIMITATIONS

The Cell/BE processor also overcomes three important limitations of contemporary microprocessor performance, i.e. power, memory and frequency limitations[114, 121]:

3.3.1. OVERCOMING THE POWER WALL

Since microprocessor performance is limited by achievable power dissipation rather than by the number of available resources (transistors and wires), the only way to significantly increase the performance is to improve the power efficiency at about the same rate as the performance increase. One way to increase power efficiency is to differentiate between processors optimized to run an operating system and control-intensive code, and processors optimized to run compute intensive applications. The Cell/BE does this by

providing a general-purpose PPE to run the operating system and other control-plane code, and eight SPEs specialized for computing data-plane applications.

3.3.2. OVERCOMING THE MEMORY WALL

Performance is often dominated by the activity of moving data between the processor and the main storage. Hence, movement of data must be managed explicitly, even with the existence of hardware cache mechanisms. The Cell Broadband Engine's SPEs use two mechanisms to deal with long main-memory latencies. The first mechanism is a 3-level memory structure consisting of the main storage, local stores in each SPE, and large register files in each SPE. The second mechanism is the availability of asynchronous DMA transfers between main storage and local stores. These features allow programmers to be able to schedule simultaneous data and code transfers to cover long latencies effectively. Because of this organization, the Cell Broadband Engine can support 128 simultaneous transfers between the eight SPE local stores and main storage. This surpasses the number of simultaneous transfers on conventional processors by a factor of almost twenty.

3.3.3. OVERCOMING THE FREQUENCY WALL

Conventional processors require increasingly deeper instruction pipelines to achieve higher operating frequencies. This technique has reached a point of diminishing returns and even negative returns if power is taken into account. The Cell Broadband Engine specializes the PPE and the SPEs for control and compute-intensive tasks, respectively. Hence, allowing both the PPE and the SPEs to be designed for high frequency without

excessive overhead. The PPE achieves efficiency primarily by executing two threads simultaneously rather than by optimizing single-thread performance. Each SPE achieves efficiency by using a large register file, which supports many simultaneous in-flight instructions without the overhead of register-renaming or out-of order processing. Each SPE also achieves efficiency by using asynchronous DMA transfers, which support many concurrent memory operations without speculative overheads.

3.4. INTERPROCESSOR COMMUNICATION

Although it is a multiprocessor system on a chip, the Cell/BE processor is not a traditional shared-memory multiprocessor. One of the major characteristics is that an SPE can execute programs and directly load and store data only from and to its private Local Storage (LS). Since SPEs lack shared memory, they must communicate explicitly with the PPE or other SPEs using three primary communication mechanisms: DMA transfers, mailbox messages, and signal-notification messages[122]. All three communication mechanisms are implemented and controlled by the SPE's MFC.

3.4.1. DMA TRANSFER

DMA transfers are the most important means of communication on the Cell/BE processor, facilitating both bulk data transfers and synchronization. The capabilities of DMA transfers are summarized below:

- DMA transfers enable exchange of data between the main memory and the local stores of the SPEs, as well as transfers from one local store to another.

- The messages can be of size 1, 2, 4, 8, and 16 bytes, and multiples of 16 bytes up to 16KB. Source and destination addresses of messages 16 bytes and larger have to be 16 bytes aligned, and addresses of messages shorter than 16 bytes require the same alignment as the message size. Additionally, messages of subvector sizes (less than 16 bytes) have to have the same alignment of source and destination addresses within the vector.
- Messages larger than 16KB can only be achieved by combining multiple DMA transfers. DMA lists are a convenient facility to achieve this goal, as well as to implement strided memory access. A DMA list can combine up to 2048 DMA transfers.
- DMA transfers are most efficient if they transfer at least one cache line and if they are aligned to the size of a cache line, which is 128 bytes.
- By default, DMA messages are not ordered. Ordering of DMAs can be enforced by the use of *barriers* and *fences*. A *barrier* orders a message with respect to messages issued before as well as after a given message. A *fence* orders a message only with respect to messages issued before the given message
- DMA transfers are non-blocking in their very nature. While DMAs are in progress, the SPE should be doing some useful work and only check for DMA completion, when it comes to processing of the transferred data.
- DMA engines are parts of the SPEs. Each SPE can queue up to 16 requests in its own DMA queue. Each DMA engine also has a proxy DMA queue, which can be accessed by the PPE and other SPEs. The proxy queue can hold up to eight requests.

Both the SPEs and the PPE are capable of initiating DMAs, but the SPE-initiated DMAs are more efficient and should be given preference over the PPE-initiated DMAs. Nevertheless, if the need arises to use the PPE-initiated DMAs, it can be accomplished by means of the MFC SPE proxy command functions.

Although each single SPE has a theoretical bandwidth of 25.6 GB/s, which is equal to the peak bandwidth of the main memory, a single SPE will have a hard time saturating this bandwidth. In order to get good utilization of the bus, one should initiate many requests from many SPEs, and also restrain from ordering the messages, if possible, to give the arbiter the most room for traffic optimization.

One of the important aspects of the Cell/BE communication system is the efficiency of local store to local store communication[123]. Local store to local store communication may prove invaluable not only for bulk data transfers, but also for synchronization between SPEs.

3.4.2. MAILBOXES

Mailboxes support the sending of short, 32-bit messages from the PPE to the SPEs and between the SPEs. The mailboxes are *First-In-First-Out* (FIFO) queues, meaning the messages are processed in the order of their issue. Each SPE has a four-entry mailbox for receiving incoming messages from the PPE and other SPEs, and two one-entry mailboxes for sending outgoing messages to the PPE and other SPEs - one of which serves the purpose of raising an interrupt on the receiving device.

Mailbox operations have blocking nature on the SPE. An attempt to write to a full outbound mailbox will stall until the mailbox is cleared by a PPE read. Similarly, an attempt to read from an empty inbound mailbox will stall until the PPE writes to the mailbox. The same does not apply to the PPE. Neither an attempt to write to a full mailbox nor an attempt to read an empty mailbox will stall the PPE. Mailboxes are useful to communicate short messages, such as completion flags or progress status. They can also serve the purpose of communicating short data, such as storage addresses and function parameters. The blocking nature of the mailboxes on the SPE side makes them perfect for the PPE to initiate actions on the SPEs. However, they are not suitable to acknowledge SPE completion of operations to the PPE.

Although mailbox message values are intended to communicate messages up to 32 bits in length, such as buffer completion flags or program status, they can also be used for any short-data transfer purpose, such as sending of storage addresses, function parameters, command parameters, and state-machine parameters.

3.4.3. SIGNAL NOTIFICATION CHANNELS (SIGNALS)

SPE signal-notification channels are connected to inbound registers (into the SPE). A signal is a short message of up to 32 bits long from the PPE, another SPE, or another system device. An example of this is buffer-completion synchronization flag. An SPE has two 32-bit signal-notification registers, each of which has a corresponding MMIO register that can be written with signal-notification data. They can be configured for one-to-one signaling or many-to-one signaling. SPE software can use polling or blocking

when waiting for a signal to appear, or it can set up interrupts to catch signals as they appear asynchronously.

A signal-notification message is sent to the SPE by writing to the main storage address of an MMIO register in the SPU's MFC. The signal is latched in the MMIO register, and the SPU executes a read-channel (`rdch`) instruction to get the signal value. An SPU can send a signal-notification message to another SPU with its special send-signal instructions (for example, `sndsig`). An SPE read of one of its two signal-notification channels clears the channel. However, a PPE MMIO read does not clear the channel.

One SPE can send a signal-notification message to another SPE using one of three special MFC commands: `sndsig`, `sndsigf`, and `sndsigb`. All of these commands are implemented in the same manner as a DMA `put` command, with the effective address of an MMIO register as the destination.

Like mailboxes, signal-notification channels are useful when the SPE places computational results in main storage via DMA. After requesting the DMA transfer, the SPE waits for the DMA transfer to complete and then sends a signal to notify the PPE that its computation is complete. In this case, waiting for the DMA transfer to complete only ensures that the SPE's LS buffers are available for reuse and does not guarantee that data has been coherently written to main storage.

Although signal notifications and mailbox messages look similar, there are important differences. Table 3 summarizes the differences between communication using a mailbox and using a signal.

Table 3. Comparison of mailboxes and signals

Attribute	Mailboxes	Signals
Direction	One inbound, two outbound, all accessible through channel interface.	Two inbound accessible through channel interface, but can send signal using MFC send-signal commands.
Interrupts	One mailbox can interrupt PPE Two mailbox-available Event interrupts	Two signal-notification Event interrupts.
Message Accumulation	No.	Yes: over-write mode (one-to-one), logical OR mode (many-to-one).
Unique SPU Commands	No; programs use channel reads and writes.	Yes, *sndsig*, *sndsigf*, and *sndsigb* for sending signals to other units.
Destructive Read	Reading a mailbox consumes an entry.	Reading a channel resets all 32 bits to '0'.
Channel Count	Indicates number of available entries.	Indicates waiting signal.
Number	Three mailboxes: 4-deep incoming, 1-deep outgoing, 1-deep outgoing with interrupt.	Two signal registers.

3.5. DEVELOPING APPLICATIONS FOR THE CELL BROADBAND ENGINE

Writing efficient and scalable code for the Cell/BE is, in many ways, different than programming most of the common modern architectures. The main differences come from the fact that, on the Cell architecture, the user has full control over the processor

behavior and all the hardware details are exposed to the programmer. The following general programming rules should be followed in order to exploit the full potential of the Cell/BE[123].

3.5.1. VECTORIZATION

The SPEs are vector units. In a code that is not vectorized, every scalar operation will be transformed to a vector operation which results in a considerable performance loss. Hence, vectorizing the code will ensure a gain in performance.

3.5.2. DATA ALIGNMENT

Since the local storage on the SPEs is relatively small, most of the operations will require a continuous streaming of data from the main memory to the SPEs local memory. As a result, non optimized memory transfers will deeply impact the performance. In order to achieve the best transfer rates, data accesses must be aligned both on the main memory and the SPEs local memories. Alignment will provide a better exploitation of the memory storage and a better performance of DMA transfer.

3.5.3. DOUBLE-BUFFERING

As explained in the previous point, data is continuously streamed from main memory to SPEs. The cost of all this communication is thus, considerable. Moreover each single message has to be relatively small in size since local memories have limited storage space; this means that a high number of DMA transfers will be performed in a single operation, each of which will add the (fixed) cost of the DMA latency to the

communication phase. In order to hide the cost of the latencies and memory transfers, DMA transfers can be overlapped with SPE local computations using double buffering. If these local computations are more expensive than a single data transfer, the communication phase can be completely hidden.

3.5.4. DATA REUSE

To reduce the number of memory transfers, it is important to arrange the instructions in order to maximize the reuse of data once it has been brought into the SPEs local memories. Explicit unrolling provides considerable improvements in performance due to the high number of registers on the SPEs and to the simplicity of SPEs architecture.

3.5.5. BRANCH MINIMIZATION

SPEs can only do static branch prediction. Therefore, reducing the number of branches in the code usually provides performance improvements, since these prediction schemes are rather inefficient on programs that have a complex execution flow.

3.6. PROGRAMMING TECHNIQUES FOR THE CELL/BE

Programming is the process of writing, testing, and maintaining the source code of computer programs. The choice of programming techniques is to achieve an elegant, efficient, and maintainable software program that exhibits the desired behavior. The process of writing source code requires expertise in many different subjects, including knowledge of the application domain, compiler and the target processor. On any processor, coding optimizations are achieved by exploiting the unique features of the

hardware. In the case of the Cell/BE, the large number of SPEs, their large register file, and their ability to hide main-storage latency with concurrent computation and DMA transfers support many interesting programming models. With the computational efficiency of the SPEs, software developers can create programs that manage dataflow as opposed to leaving dataflow to a compiler or to post optimization process.

Many of the unique features of the SPE are handled by the compiler, although programmers looking for the best performance can take advantage of the features independently of the compiler. It is almost never necessary to optimize the benefits but programming the SPE in assembly language as C intrinsics provides a convenient way to program the efficient movement and buffering of data.

Table 4. Classification of Cell/BE applications into programming techniques

Technique	Applications	Reference
Function-Offload Model	RAxML-Cell	[124]
Computation-Acceleration Model	Smith-Waterman (short sequences)	[106]
	ClustalW	[106]
	Real-time wavelet decomposition for HDTV video images	[125]
	Ray Tracing	[126]
	Smoothed Particle Hydrodynamics	[127]
Streaming Model	Smith-Waterman (long sequences)	[102]
	CBESW	[101]
	BLAST	[104]
	Pairwise Distance Matrix computation	[103]
	Euler particle-system simulation	[122]
	Volume Ray Casting	[128]

Table 4 shows a classification of applications, that have been recently developed on the Cell/BE, into programming techniques. These techniques and applications are explained in more detail below.

3.6.1. FUNCTION-OFFLOAD MODEL

The function-offload mode, also called the Remote Procedure Call (RPC) Model, is the fastest way to effectively use the Cell/BE with an already existing application. It specifically notes the use of program stubs via the Interface Description Language (IDL).

PPE code:

```
Start main application
Invoke RPC call using IDL interface
Call SPE procedure
Wait/synchronize
Continue main application
End
```

SPE code:

```
Start
Receives RPC call
Run SPE procedure
Return result to PPE
End
```

Figure 15. Pseudocode of the Function Offload Model

In this model, the main application runs on the PPE and calls selected performance-critical procedures to run on one or more SPEs, which are used as accelerators. An interface description in the form file with an extension of .idl is required. The model allows a PPE program to call a procedure located on an SPE as if it were calling a local

procedure on the PPE. This provides an easy way for programmers to use the asynchronous parallelism of the SPEs without having to understand the low-level workings of the MFC DMA layer. However, it is essential to identify which procedures should be executed on the PPE and which on the SPEs. The PPE and SPE source modules must be compiled separately, by different compilers. The pseudocode of the function offload model is shown in Figure 15.

An example of an application that uses the function-offload model is the RAxML. RAxML[124] (Randomized Axelerated Maximum Likelihood) is a bioinformatics program for large-scale ML-based (Maximum Likelihood[129]) inference of phylogenetic (evolutionary) trees using multiple alignments of DNA or amino acid sequences. The MPI version of the RAxML was ported to the PPE and both loop-level parallelization of tasks across SPEs and a scheduler which multiplexes more than two MPI processes on the PPE using an event-driven model were introduced to expose more task-level parallelism. The most time-consuming functions of each MPI process, namely newview, evaluate and makenewz, were offloaded to the SPEs. The SPE codes are optimized using vectorization of computation, a specialized casting transformation coupled with vectorization of control statements, and communication optimizations. Besides the fact that each function can be executed faster on an SPE, having all three functions offloaded to an SPE significantly reduces the amount of PPE-SPE communication. An SPE thread is spawned at the beginning of each MPI process. The thread executes the offloaded function upon receiving a signal from the PPE and returns the result back to the PPE upon completion. To avoid excessive overhead from repeated spawning and joining of threads, threads remain bound on SPEs and perform a busy-wait

for the PPE signal to start executing a function. Furthermore, the codes of all three offloaded functions are loaded on each SPE, such that each thread can execute any of the functions on demand, including nested combinations of these functions.

The performance of the Cell/BE implementation of RAxML was compared to the MPI implementation of RAxML on two architectures:

- A 32-bit Intel Pentium 4 Xeon with Hyperthreading technology (2-way SMT), running at 2GHz, with 8KB L1-D cache, 512KB L2 cache, and 1MB L3 cache.
- A 64-bit IBM Power5 processor, a quad-thread, dual-core processor with dual SMT cores running at 1.65 GHz, 32KB of L1-D and L1-I cache, 1.92 MB of L2 cache, and 36 MB of L3 cache.

The Cell/BE processor clearly outperforms the Intel Xeon by more than a factor of two, while Cell/BE performs 9%-10% better compared to the IBM Power5. The computation uses double precision floating point arithmetic, which is not optimized for Cell SPE pipelines. Hence, the use of single-precision arithmetic would further widen the performance margin between Cell and the IBM Power5.

3.6.2. COMPUTATION-ACCELERATION MODEL

The Computation-Acceleration Model is an SPE-centric model that provides a smaller-grained and more integrated use of SPEs than the function-offload model. The Computation-Acceleration Model speeds up applications that use computation-intensive mathematical functions without requiring a significant rewrite of the applications. Most computation-intensive sections of the application run on SPEs. The PPE acts as a control and system service facility. Multiple SPEs work in parallel. The work is partitioned

manually by the programmer, or automatically by the compilers. The SPEs must efficiently schedule MFC DMA commands that move instructions and data. This model either uses shared memory to communicate among SPEs, or it uses a message-passing model. The pseudocode of the computer-acceleration model is shown in Figure 16.

PPE code:

```
Start main application
Construct a context and thread for each SPEs
Create SPE threads with the context as parameter
Wait for all SPEs to complete
Continue main application if required
End
```

SPE code:

```
Define local storage and buffer
Fetch SPE context through DMA
Fetch data through DMA
While there are task to be executed
      Do necessary computations
Return result to PPE
End
```

Figure 16. Pseudocode of the Computation-Acceleration Model

Examples of applications that use this technique on the Cell/BE are Smith-Waterman algorithm for short sequences[106], ClustalW[106], ray tracing[126], smooth particle hydrodynamics[127] and real-time wavelet decomposition for HDTV video images[125]. The Smith-Waterman algorithm[33] finds the optimal local alignment of two sequences by means of dynamic programming. It compares two sequences by computing a distance that represents the minimal cost of transforming one subsequence into another. Two basic

operations are used in the transformation, i.e. insertion/deletion and substitution. The distance between the subsequences is measured as the smallest number of operations required to change one subsequence into another. The Smith-Waterman implementation by Sachdeva et. al.[106] is capable of executing a pairwise alignment of 8 pairs of sequences, using one SPE for each pairwise alignment. Their implementation requires both sequences to fit entirely in the SPE local store of 256 KB, which limits the sequence length to 2048. The alignment scores were pre-computed on the PPE, and then DMA-transferred to the SPEs together with the query and the library sequences. Other parameters such as the alignment matrix and the gap penalties are also included in the context for every SPE. The Smith-Waterman kernel, which is based on the FASTA package by Eric Lindahl, was then executed in each SPE. As for the load balancing, a simple round-robin strategy was implemented, in which the sequences in the query library are allocated to the SPEs based on the sequence numbers and the SPE number. The implementation on the Cell/BE running on 8 SPEs performs 6.2 times and 4.7 times faster compared to implementations on Opteron with SSE2 code and PowerPC G5 with Altivec code, respectively.

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 a global alignment algorithm. A cluster analysis is then performed on each of the scores from the pairwise alignment to generate a hierarchy for alignment. Finally, the alignment is built step by step, adding one sequence at a time, according to the guide tree. The ClustalW implementation by Sachdeva et. al.[106] is focused on running the *paralign* function, which performs the task of comparing all input sequences against each other, on the

SPEs, with the rest of the code executing on the PPE. The paralign function performs a total of $n(n-1)/2$ alignments for n sequences, consuming about 60-80% of execution time. 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 implementation ported the IBM Life Science version with vectorized *forward_pass* for the SPE code. The computation starts with the PPE creating the SPE threads and passes the maximum sequence size through a mailbox message. The SPEs allocate memory only once in the entire computation based on the maximum size, and then wait for the PPE to send a message for them to pull in the context data using DMA transfer and begin the computation. Work load is assigned to the SPEs using a simple round-robin strategy: each SPE is assigned a number from 0 to 7, and SPE k is responsible for comparing sequence number i against all sequences $(i + 1)$ to n if $(i \ mod \ 8 = k)$. For storing of the output values, the SPEs are also passed a pointer to an array of structures, which are 16-byte aligned, in which they can store the output of the forward pass function executed for two sequences. The ClustalW code, executing on the PPE side, then uses the output for the forward pass function to generate the guide tree from the scores received from the SPE, and to compute the final alignment. The implementation on the Cell/BE running on 8 SPEs performs up to 1.26 times faster when compared to a SSE-vectorized implementation on PowerPC G5.

Ray tracing is a general technique from geometrical optics of modeling the path taken by light by following rays of light as they interact with optical surfaces. An implementation of ray tracing in the Cell/BE, which using the BHV traversal scheme proposed in [130], is described in [126]. Each SPE independently runs a full ray tracer, and parallelization is

achieved by SPEs working on different pixels. The implementation starts with subdividing the image plane into a set of image tiles. From this shared task queue, each SPE dynamically fetches a new tile, and renders it. An integer variable, specifying the ID of the next tile to be rendered, is allocated in system memory to ensure synchronization of the accesses to the task queue. This variable is visible among all SPEs, and each time an SPE queries the value of the variable, it performs an atomic fetch-and-increment. This atomic update mechanism allows the SPEs to work fully independently from both other SPEs and PPE, requiring no communication among those units. The only explicit synchronization is at the end of each frame, where the PPE waits to receive an ’end frame’ signal from each SPE. The ray tracing implementation on dual Cell/BE with 16 SPEs is evaluated to be 7.1-15.3 times faster than a 2.4 GHz AMD Opteron-based system. Extrapolating the performance that would be achievable on a 3.2GHz Cell with 7 SPEs as used in a Playstation 3 yield a speed up of 4.8-9.6 times that of an Opteron CPU.

Smoothed Particle Hydrodynamics (SPH) is a method used mainly to simulate complex materials, such as water. The particles can be seen as interpolation points, approximating local field quantities. It has over time been applied to numerous problems, such as, elasticity and fracture modeling [131], hair-hair interactions [132], and simulation of incompressible fluids [131]. The SPH implementation on the Cell/BE[127] starts with creating the hash table of every particle on the PPE, serially. In the meantime, the SPEs pre-calculate the hash values for the neighboring cells to find interacting particles. The interaction list is created by iterating over the hash buckets, calculating distances between particles. Because of hash value collisions, it is possible that duplicate particle interactions are found. Therefore, in addition to checking distances between two particles,

their grid cell relation must be tested. This is done using *workblocks* of *N* buckets, which are processed in parallel on the SPEs. It ignores the symmetrical property of the inter-particle forces in order to maximize data locality and to allow asynchronous execution of the SPE threads. Afterwards, the SPE threads then process particles in batches from the *workpool* to compute the time integration and collision handling for velocity reflection and position projection. The results are then sent back to the PPE. The SPH implementation on the Cell/BE with 8 SPEs performs 9.8 faster compared to a scalar implementation on a 2.0GHz PPC 970 processor, reaching 39.8 Hz update frequency with 15 avg. neighbors / particle.

Wavelet decomposition is one of the essential methods for compressing or decompressing high resolution images. The real-time wavelet decomposition for HDTV video images [125] implementation starts with the PPE reading the image file, dividing it into 8 pieces and then sending the context about the divided image pieces, e.g. the size, address in main memory, etc to each of the 8 SPEs. Each SPE receives the context and obtains the divided image from the main memory. The 1D Fast Wavelet Transform (FWT) computations in each SPE are done using the SIMD instructions to exploit the data parallelism. Thus each partial image assigned to an SPE is formed into appropriate matrix size for the SIMD instruction by transposing the elements. The image data can be processed 4 pixels at a time with SIMD instructions. After the 1D FWT is completed, the data from each SPE is sent back to the PPE and stored in the main memory. After all of the SPEs finish the process, the whole image as decomposed along the x coordinates is ready. The same process is repeated for the decomposed image on the other dimension (along the x coordinates), and the 2D wavelet decomposition is thus performed.

3.6.3. STREAMING MODEL

In the Streaming Model, each SPE, in either a serial or parallel pipeline, computes data that streams through. The PPE acts as a stream controller, and the SPEs act as stream-data processors. For the SPEs, on-chip load and store bandwidth exceeds off-chip DMA-transfer bandwidth by an order of magnitude. If each SPE has an equivalent amount of work, this model can be an efficient way to use the Cell Broadband Engine because data remains inside the Cell Broadband Engine as long as possible. The PPE and SPEs support message-passing between the PPE, the processing SPE, and other SPEs. The pseudocode of the streaming model is shown in Figure 17.

PPE code:

```
Start main application
Construct a context and thread for each SPEs
Create SPE threads with the context as parameter
Wait for all SPEs to complete
Continue main application if required
End
```

SPE code:

```
Define local storage and buffer
Fetch SPE context through DMA
While there are task to be executed
      Fetch data through DMA
      Do necessary computations
      Do SPE-PPE/SPE-SPE communication if necessary
End
```

Figure 17. Pseudocode of the Streaming Model

The techniques used to implement the alignment of long DNA sequences[102], CBESW[101], BLAST[104] and pairwise distance matrix computation[103] are discussed further in Chapter 4, 5, 6 and 7, respectively.

The Euler particle-system simulation described in [122] contains a computational kernel that streams packets of data through the kernel for each step in time. Using DMA transfers for PPE-SPE communication, the SPEs fetch the context. For each step in time for the block of particles, the SPEs fetch their respective data (position, velocity and inverse mass) by means of DMA transfer. Once it is completed, the SPEs perform the Euler computation and put back the position and velocity data back into system memory.

The volume ray casting implementation[128] introduces streaming model based schemes and techniques to efficiently implement acceleration techniques for ray casting on Cell/BE. In addition to ensuring effective SIMD utilization, their method provides two key benefits: there is no cost for empty space skipping and there is no memory bottleneck on moving volumetric data for processing. Furthermore, experimental results show that we can interactively render practical datasets on a single Cell/BE processor.

4. ALIGNING LONG DNA SEQUENCE ON THE CELL BROADBAND ENGINE

This chapter elaborates our implementation of a novel, efficient and scalable parallel algorithm for very long DNA sequence alignments on a heterogeneous multi-core system, the Cell Broadband Engine. The two types of parallelization utilized in the implementation, i.e. the wavefront and the SIMD vectorization are discussed. Lastly, performance comparisons to other architectures such as GPU and FPGA are provided.

4.1. INTRODUCTION

Sequence alignment is an essential tool to determine the degree of similarity between nucleotide or amino acid sequences which is assumed to have same ancestral relationships. The optimal local alignment of a pair of sequences can be computed by the dynamic programming (DP) based Smith-Waterman algorithm[33]. However, this approach is very expensive in terms of time and memory cost. One technique to speedup this time consuming task is to introduce heuristics in the search algorithm, e.g. BLAST [45]. The drawback of this approach is that the more efficient the heuristics, the worse is the result. In other words, these algorithms sacrifice sensitivity for speed. Hence, more distant sequence relationship may not be detected.

Another popular approach to reduce computational time without sacrificing the performance is to use High Performance Computing. Examples of parallel architectures that have been evaluated for this problem include FPGAs[86], GPUs[93] and SIMD arrays[133]. In this chapter, we investigate how the Cell Broadband Engine can be used

as a computational platform to accelerate sequence alignment for very long DNA sequences.

4.2. SMITH-WATERMAN ALGORITHM

The Smith-Waterman algorithm is used to determine the optimal local alignment between two nucleotide or protein sequences. The algorithm compares two sequences by computing the similarity score by means of dynamic programming (DP). Two elementary operations are used: substitution and insertion/deletion (also called a gap operation). The original algorithm was proposed by Smith and Waterman[33] with a complexity of $O(m^2n)$ and was improved by Gotoh[65] to run at $O(mn)$. The Smith-Waterman Algorithm as a local alignment has been explained briefly in section 2.2.3.2.

Consider two strings S_1 and S_2 with length m and n, respectively. The Smith-Waterman algorithm computes the similarity value $M(i, j)$ of two sequences ending at position i and j of the two sequences S_1 and S_2, respectively. For affine gap penalties, i.e. $\alpha \neq \beta$, the computation of $M(i, j)$, for $1 \le i \le m$, $1 \le j \le n$, is given as shown in Equation 10:

$$M(i, j) = \max \{M(i-1, j-1)+sbt(S1[i], S2[j]), E(i, j), F(i, j), 0\}$$
$$E(i, j) = \max \{M(i, j-1) - \alpha, E(i, j-1) - \beta\},$$
$$F(i, j) = \max \{M(i-1, j) - \alpha, F(i-1, j) - \beta\},$$

Equation 10. Smith-Waterman equation for affine gap penalties

where sbt is a character substitution cost table, α is the cost of the first gap; β is the cost of the following gaps. For linear gap penalties, i.e. $\alpha = \beta$, the above recurrence relations can be simplified, as shown in Equation 11:

$$M(i, j) = \max\{M(i-1, j-1) + sbt(S1[i], S2[j]), M(i, j-1) - \alpha, M(i-1, j) - \alpha, 0\}$$

Equation 11. Smith-Waterman equation for linear gap penalties

Initialization values are given as the following: for $0 \le i \le m$, $0 \le j \le n$, $M(i, 0) = M(i, j) = E(i, 0) = F(i, j) = 0$. Each position of the matrix M is a similarity value. The maximum local alignment score is defined as the maximal value in matrix H. The two segments of $S1$ and $S2$ producing this value can be determined by a trace-back procedure. The three arrows in Figure 18 show the data dependencies in the alignment matrix: each cell depends on its left, upper, and upper-left neighbors.

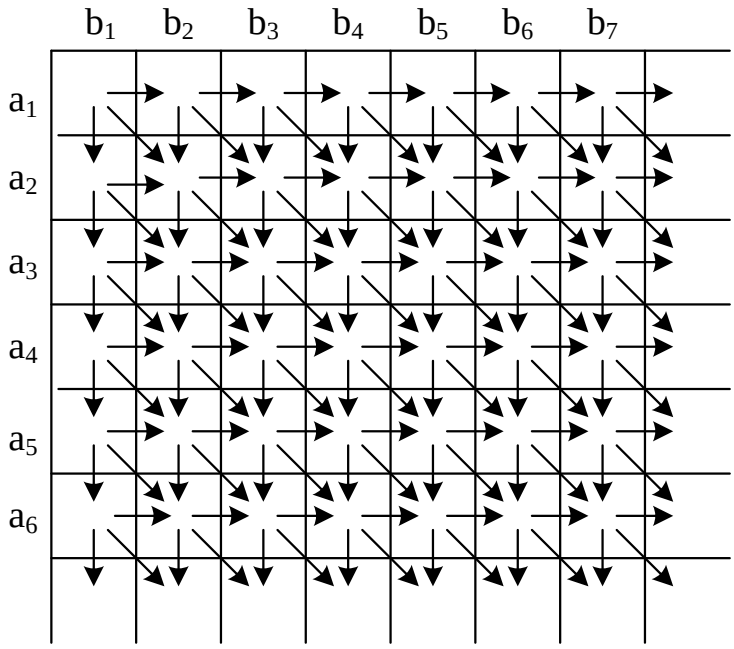

Figure 18. Data dependency in the SW algorithm alignment matrix

Figure 19 illustrates an example of computing the local alignment between two DNA sequences **CAGTTTCG** and **ACAGTCGAACG** using the Smith-Waterman algorithm. The matrix $M(i, j)$ is shown for the linear gap cost $\alpha = \beta = 1$, and a substitution cost of +2 if the characters are identical and -1 otherwise. The highest score in the matrix (+10) is the optimal score for the alignment. The trace-back procedure, shown in form of arrows, shows that the optimal local alignment is **CAGTTTCG** and **CAG – –TCG**.

		A	C	A	G	T	C	G	A	A	C	G
	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	+2	+1	0	0	+2	+1	0	0	+2	+1
A	0	+2	+1	+4	+3	+2	+1	+1	+3	+2	+1	+1
G	0	+1	+1	+3	+6	+5	+4	+3	+2	+2	+1	+3
T	0	0	0	+2	+5	+8	+7	+6	+5	+4	+3	+2
T	0	0	0	+1	+4	+7	+7	+6	+5	+4	+3	+2
T	0	0	0	0	+3	+6	+6	+6	+5	+4	+3	+2
C	0	0	+2	+1	+2	+5	+8	+7	+6	+5	+6	+5
G	0	0	+1	+1	+3	+4	+7	+10	+9	+8	+7	+8

Figure 19. Sequence alignment of CAGTTTCG and ACAGTCGAACG

4.3. WAVEFRONT PARALLELIZATION

Our parallel algorithm employs a static load balancing strategy, which means that the work load is known at the start and distributed equally across processes and processors. The algorithm starts by reading the input dataset. The PPE then preprocesses the set of input sequences such that all the SPEs will have their respective sequence parts in their local memory. Consider two sequences, *S1* and *S2* of length *m* and *n* respectively. Assume that *p* SPEs, P_1, ..., P_p, are used for the computation. *S1* is broadcast to all SPEs, while *S2* is divided into *p* pieces, of size *n/p*, and each SPE P_i, $1 \leq i \leq p$, receives the i-th piece of *S2*. Each SPE has to compute a non-overlapping *m* x *n*/*p* submatrix of the whole *m* x *n* DP matrix. This computation is performed in $q + p - 1$ rounds, where $q = m/r$ and *r* denotes the number of consecutive rows calculated in one round. Hence, each round

computes an *r* x *n/p* submatrix in a number of SPEs in parallel in each round. The scheduling scheme follows a wavefront pattern and is illustrated in Figure 20.

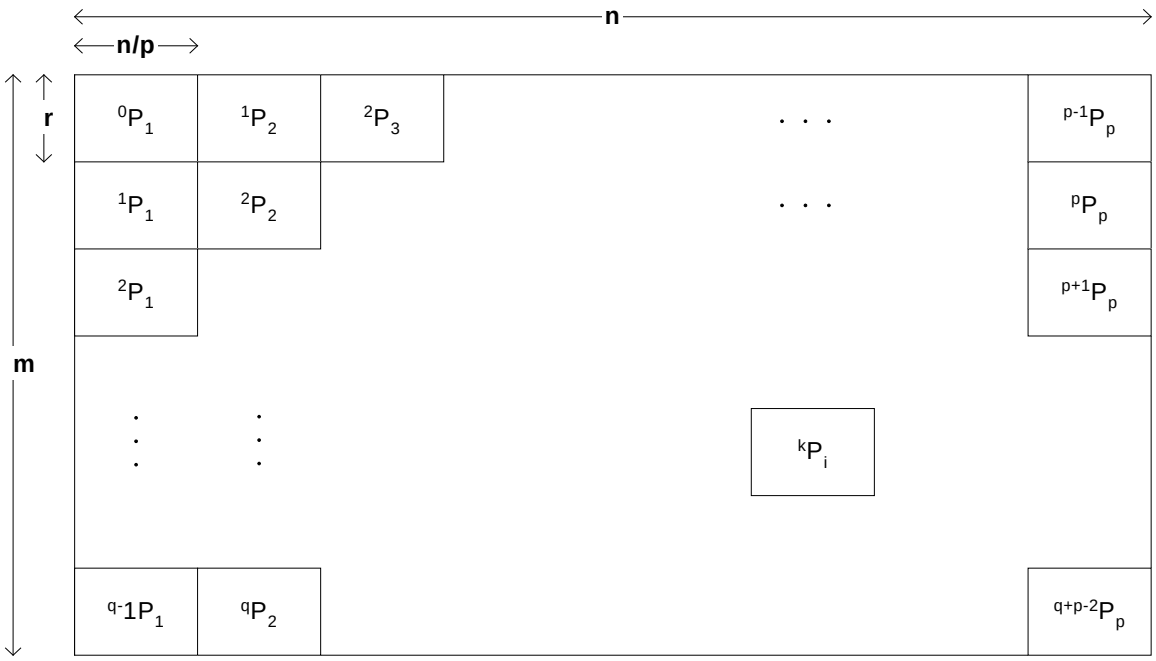

Figure 20. Block diagram of the wavefront algorithm

The notation $^{k}P_{i}$ denotes the sub-matrix computed by the SPE P_i at round k. Thus, at the start, P_1 starts computing $^{0}P_1$ at round 0. Then, P_1 and P_2 computes $^{1}P_1$ and $^{1}P_2$, respectively at round 1; P_1, P_2 and P_3 computes $^{2}P_1$, $^{2}P_2$ and $^{2}P_3$, respectively at round 2, and so on. Due to the limitation of the local storage of SPE of 256 KB for both the program and the data, we implemented a linear space algorithm. Hence, in each $^{k}P_i$, the similarity value $M(i, j)$ at position i and $_j$ is then computed by according to Equation 12:

$$M(i, j) = \max \{M(j-1)+sbt(S1[i], S2[j]),E, F(j),0\}$$
$$E(i, j) = \max \{M(j - 1) - \alpha, E \},$$
$$F(i, j) = \max \{M(j) - \alpha, F(j) - \beta\}$$

Equation 12. Modified Smith-Waterman equation for the Cell/BE mapping

After computing the k^{th} part of the ${}^{k}P_i$, SPE P_i sends the elements of the rightmost column of ${}^{k}P_i$ to SPE P_{i+1}. Using these information, SPE P_{i+1} can compute the ${}^{k+1}P_{i+1}$. After $q+p-1$ rounds, SPE P_p receives its necessary information from P_{p-1} and computes ${}^{q+p-2}P_p$ and finishes the entire alignment. During the entire computation, each SPE updates and stores its maximum local score. At the end of the computation, each SPE sends its maximum local score to the PPE through the mailbox function. The PPE uses *spe_stat_out_mbox* function to fetch the status of the SPU outbound mailbox for each SPE thread and read each maximum local score. Using those scores, the PPE then determines the global optimal score.

4.4. SIMD PARALLELIZATION

In order to further exploit the capabilities of the Cell Broadband Engine, our parallel implementation has been modified using Single Instruction Multiple Data (SIMD) registers of the SPEs for further optimization using the concepts of the vectorization strategy for Smith-Waterman comparison done by Wozniak[134].

Due to the additional memory requirement for this method as well as the local storage memory limitation of the SPEs, each SPE can only compute a submatrix of size 128x128 in each round. Hence, with 8 SPEs, we can compute an overall DP matrix of size 2048x1024. This length, however, is quite short for real life application. Hence, we have extended the algorithm such that it can compute alignment of longer sequences. In this new approach, the computation is split into blocks of size 2048x1024. Each block is computed using 8 SPEs, in which the larger 2048x1024 block is divided into smaller blocks computation of size 128x128. Once a 2048x1024 block has been computed, the local maximum is then sent to the PPE through the mailbox function and the right-most

column of this block is saved. The next 2048x1024 block is then offloaded to the SPEs to be computed. Due the nature of the Smith-Waterman algorithm, the block directly below the current block will be chosen as next block (vertical priority). Once all the blocks in the current vertical column has been computed successfully, the concatenation of the right-most column of the vertical blocks are sent and processed to compute the next batch of blocks.

Pseudocode of the SIMD parallelization scheduling is illustrated in Figure 21. At the end of all block computations, the maximum of the local maximums collected by the PPE is determined as the global optimal score.

Input:

num: Number of SPEs used, *S1* and *S2*: Sequences *S1* and *S2* with lengths *m* and *n*, respectively

Output:

S_{max}: Global maximum score for the optimum local alignment of *S1* and *S2*

SPE Pseudocode:

```
Initialize;
   While (outerloop<(n/1024)){
      Fetch the right-most column of P_num of the previous iteration from
      PPE through DMA transfer;

      Fetch part of S2 of the corresponding block from the PPE through
      DMA transfer;

      innerloop=0;

      While (innerloop<(m/2048)){
         Fetch part of S1 of the corresponding block from the PPE
         through DMA transfer;
```

```
        While (count<2048){
           if (i>0){
                 Receive signal and data from P_{i-1};
           }
           Compute sub-block for size 128x128;
           if (i<num){
                 Send signal and data to P_{i+1};
           }
           count+=128;
        }
        innerloop++;
        Send the local maximum to PPE through mailbox;
     }
     outerloop++;
     Send the right-most column of Pnum to PPE through DMA transfer;
  }
End;
```

Figure 21. Pseudocode of the SIMD parallelization scheduling

Throughout the entire computation, data is sent using direct SPE to SPE communication in order to avoid the latency of communicating through shared memory. Thus, synchronizing the communication between SPEs is crucial. Our implementation uses the MFC sendsignal command (*mfc_sndsig*) for the means of synchronization. The *mfc_sndsig* requires the effective address of the target SPE signal-notification channel as well as a 32-bit signal value. The command increments the channel count of the target SPEs signal-notification channel by one. The SPE verifies that the previous value has been read by performing an MFC get command from the effective address of the target SPE signal-notification register and ensuring that it has been reset by a channel read on the target SPE. The target SPE uses a read-channel instruction on the signal notification channel of interest to receive the 32-bit signal value. This read-channel instruction will

return immediately, reset any set bits in the signal-notification register, and reset the channel count if the associated signal-notification register has a waiting unread signal value. Otherwise, the read-channel instruction will cause the SPU to stall until a write to the signal-notification register happens.

4.5. PERFORMANCE EVALUATION

In this section, we analyze the performance of our parallel algorithm for varying number of SPEs and varying sequence lengths using artificial DNA data sets. The experiment has been conducted on the IBM Full System Simulator for the Cell Broadband Engine[135], which is a generalized simulator that can be configured to simulate a broad range of full system configurations. The simulator supports full functional simulation and is able to simulate and capture many levels of operational details on instruction execution, cache and memory subsystems, communications, and other important system functions. Furthermore, it supports cycle-accurate simulation, which not only models functional accuracy but also timing. It considers internal execution and timing policies as well as the mechanisms of system components, such as arbiters, queues, and pipelines.

The performance statistics measured from the simulator for the parallel algorithm are then converted to the following measurements: computational time, speed-up, cell updates per second (CUPS), and the parallel efficiency, as shown in Figure 22-25 respectively. The term $l(r)$ describes that the aligned sequences of length l, and r rows are being sent from one SPE to another at one time.

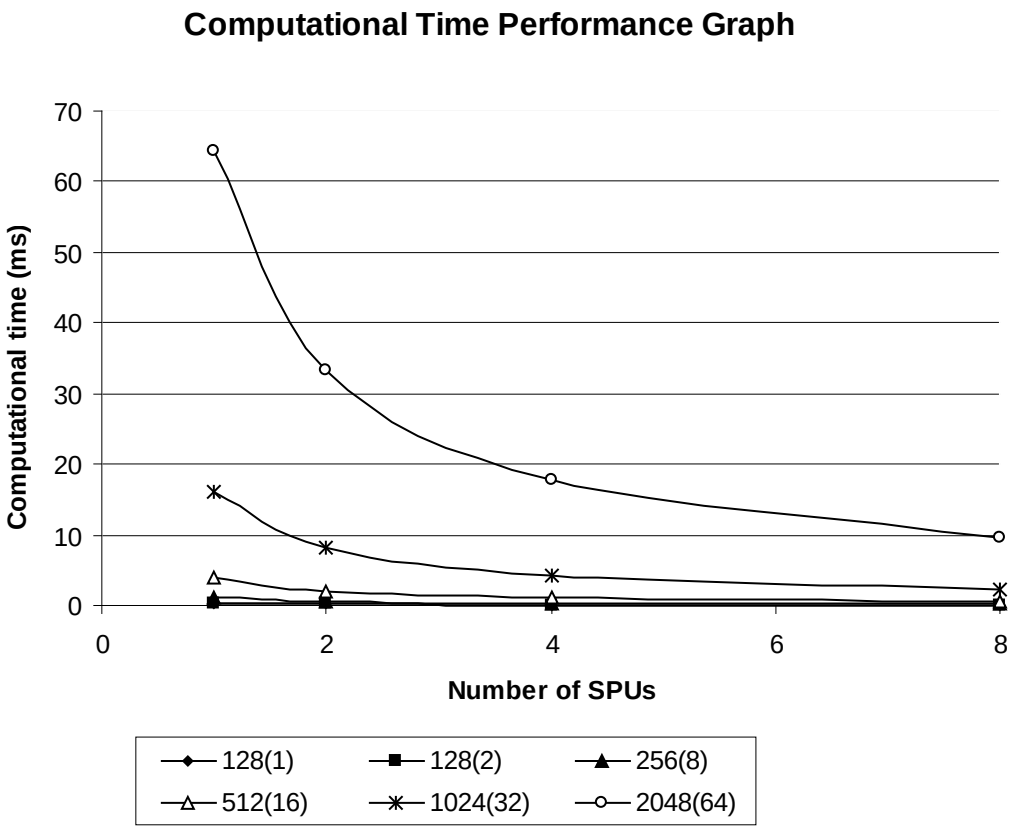

Figure 22. Computational graph of the performance evaluation results

Figure 22 shows the computational time of our parallel algorithm on the abovementioned datasets. By using 8 SPEs, our parallel algorithm managed to reduce the computational time of aligning sequences of length 2048 from 64.34 milliseconds to 9.47 milliseconds by sending 64 rows at a time.

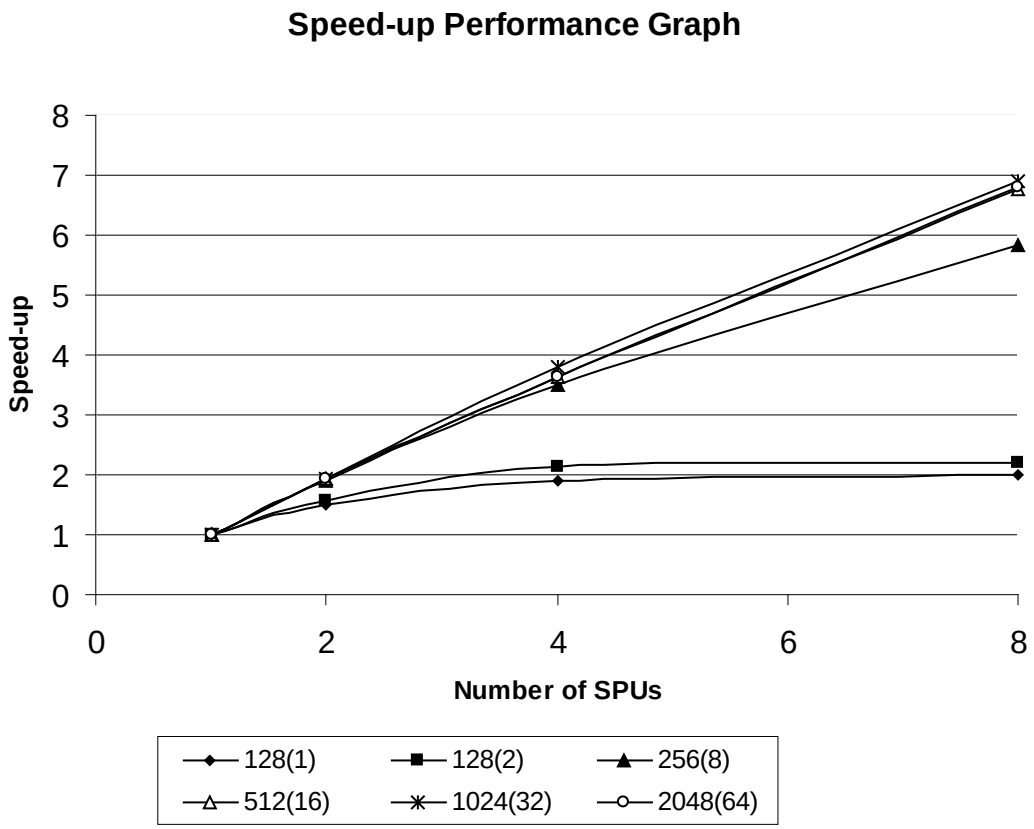

Figure 23. Speed-up graph of the performance evaluation results

The speed-up of our parallel algorithm is shown in Figure 23. By using 8 SPEs, we managed to achieve a speed-up of up to 6.91 for aligning sequences of length 2048 by sending 64 rows at a time.

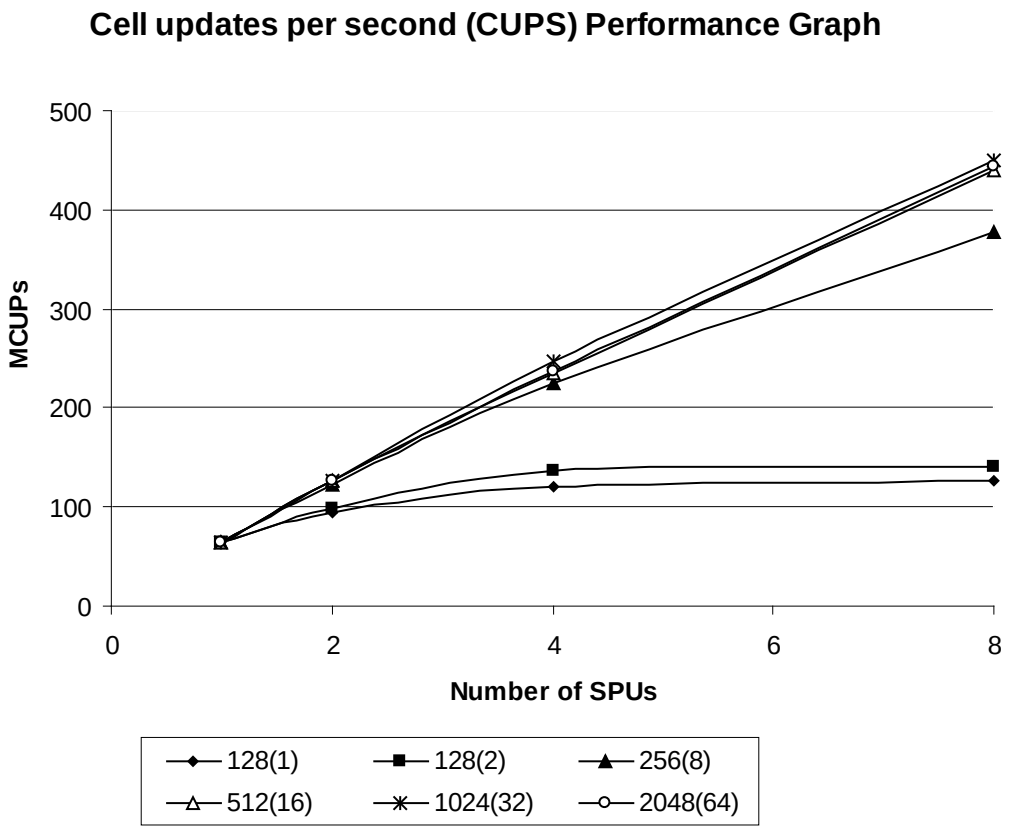

Figure 24. CUPS graph of the performance evaluation results

Figure 24 shows the performance of our algorithm in terms of cell updates per second. By using 8 SPEs, our algorithm managed to achieve a speed-up of up to 450 MCUPS for sequence alignment of length 2048 by sending 64 rows at a time.

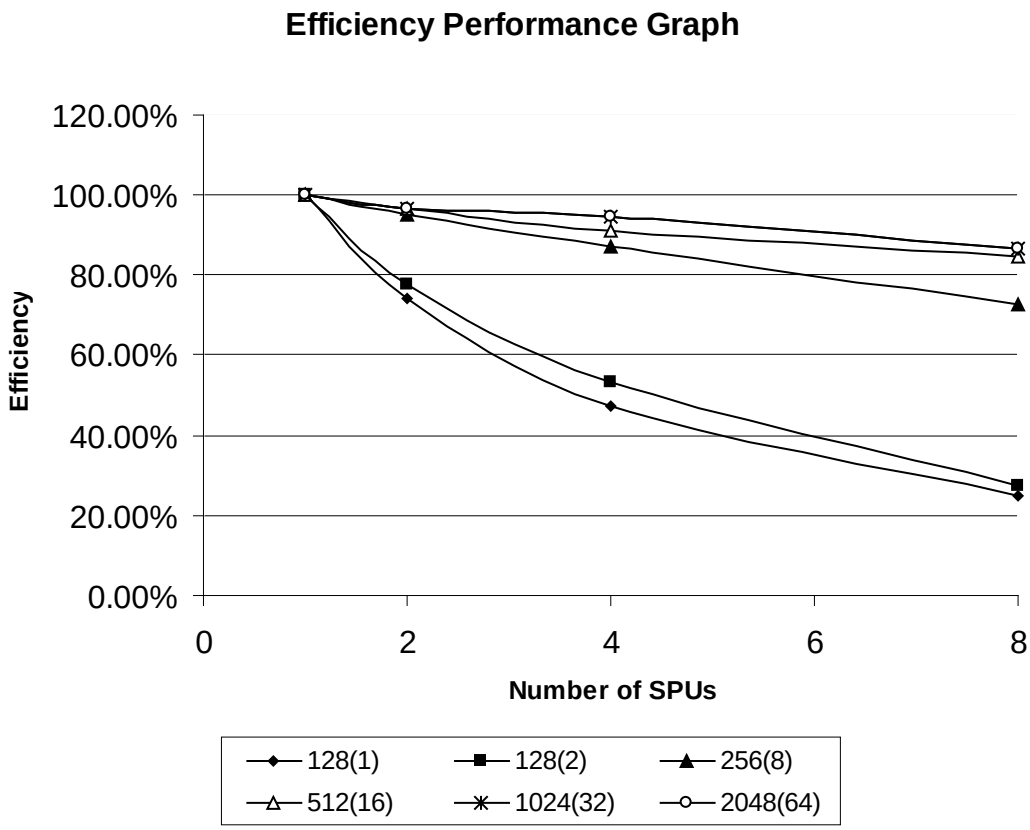

Figure 25. Efficiency graph of the performance evaluation results

Our algorithm also shows a good scalability as it achieves high efficiency, especially for datasets of longer sequences, as can be seen in Figure 25. Sequence alignment of length 2048 which sends 64 rows at a time provides 86.4% efficiency.

For the SIMD parallelization, the performance statistics obtained from the simulator are converted to computational time, and cell updates per second (CUPS). The usage of larger blocks allows the alignment of longer sequences. In the experiment results, we have aligned sequences of length up to 8192. However, 8192 is not a length restriction of our algorithm but a limit imposed by the IBM Full System Simulator simulation time.

Table 5. Performance evaluation results of the SIMD parallelization

Size	Computational Time (ms)	CUPS
2048 x 1024	0.86	2,448.65
2048 x 2048	1.49	2,808.58
4096 x 4096	5.31	3,158.31
8192 x 8192	21.34	3,169.22

As shown in Table 5, our implementation achieves a performance of up to 3,160 MCUPS. Thus, our implementation is 4-5 times faster than the Smith-Waterman implementation using GLSL on a GeForce 7900 GTX presented in [93]. The FPGA implementation using Verilog presented in [86] on a Virtex-II XC2V6000 is about 1.5 times faster than ours. Although FPGAs are flexible, their configuration has to be changed for each single algorithm, which is in general more complicated than writing code for programmable architectures such as the Cell/BE.

4.6. SUMMARY

We have presented a parallel algorithm for sequence alignment on a heterogeneous multi-core system using both SIMD vectorization and wavefront parallelism. Our implementation on the Cell/BE simulator shows almost linear speedup and reduces the computational time for sequences of 2048 to only 9.47 ms, achieving 450 MCUPS in the process. Furthermore, we have shown that by exploiting the SIMD feature of the Cell/BE, we are able to align longer sequences with excellent performance. In aligning two sequences of length 8192, our implementation achieves almost 3.2 GCUPS.

5. CBESW: IMPLEMENTATION OF THE SMITH-WATERMAN ALGORITHM ON THE PLAYSTATION®3

This chapter elaborates how the PlayStation® 3, powered by the Cell Broadband Engine, can be used as a computational platform to accelerate the Smith-Waterman algorithm for large protein datasets. Lastly, performance comparisons to other implementations such as SSEARCH, Striped Smith-Waterman, CUDA-SW and CUDASW++ are provided.

5.1. INTRODUCTION

The optimal local alignment of a pair of sequences can be computed by the dynamic programming (DP) based Smith-Waterman (SW) algorithm[33]. However, this approach is expensive in terms of time and memory cost. Furthermore, the exponential growth of available biological data[1, 136] means that the computational power needed is growing exponentially as well.

Previous works in improving the search time of the SW algorithm include the usage of SIMD multimedia extension of general-purpose CPUs as well as accessible accelerator technologies, such as FPGAs, GPUs and specialized processors. Implementation by Farrar[137] exploits the SSE2 SIMD multimedia extension of general-purpose CPUs. His implementation makes use of query profile[138] and utilizes vector registers, which are parallel to the query sequence and are accessed in a striped pattern. FPGA implementations [86, 139] tend to be very expensive and hard-to-program. Hence, they are not suitable for many users. Liu et al. [93] first reported the implementation of the

Smith-Waterman algorithm on graphics hardware using OpenGL. Although it achieves a high efficiency, programming in OpenGL requires specialized skills. Therefore, Manavski[92] re-implemented the SW algorithm on a GPU with the recently released C-based CUDA programming environment. Further SW implementations based on CUDA include [97, 140].

In this chapter, we demonstrate how the PlayStation®3 (PS3), powered by the Cell Broadband Engine[109], can be used to accelerate the Smith-Waterman algorithm.

5.2. SMITH-WATERMAN ALGORITHM

Our CBESW implementation uses the Smith-Waterman algorithm with affine gap penalties. The concept of the Smith-Waterman algorithm is described in Chapter 4.2. There are two basic approaches to vectorize the Smith-Waterman algorithm. All elements in the same minor diagonal of the DP matrix can be calculated independent of each other. Therefore, a possible vectorization approach is to compute the DP matrix in *minor diagonal order*[134], as elaborated in chapter 4. Another approach vectorizes the DP matrix computation in a *column-wise order*[137, 138]. By using vectors of elements parallel to the query sequence, the much-simplified dependency relationship and parallel loading of the vector scores from memory can be achieved, thus accelerating the DP matrix calculation.

We have decided to use the column-based approach for vectorization on the Cell/BE. processor since (1) the column-based approach outperforms the minor-diagonal approach on Intel SSE2 architectures and (2) since we only need to store one column of the DP

matrix instead of two diagonals for the minor diagonal method, the column-based approach requires less SPE memory.

5.3. IMPLEMENTATION

Details of the CBESW implementation are elaborated in this section. We explain the mapping of the algorithm to the Cell/BE, the query profile utilized to speed up the computation as well as the saturation arithmetic.

5.3.1. MAPPING TO THE CELL BROADBAND ENGINE

Our sequence alignment implementation uses affine gap penalties and utilizes the 128-bit wide SIMD vector registers of the SPEs for optimization. The vectorization strategy is based on a *column-based approach*[137, 138]. It also employs a static load balancing strategy, which means that the work load is known at the start and distributed equally across the SPEs. The code is written in C together with the Cell/BE SIMD Multimedia Extension Language intrinsics and SPU intrinsics for portability. DMA transfers and mailbox functions are used for communication purposes.

Table 6. List of SPU Low-Level Specific and Generic Intrinsics used

Category of Intrinsics	SPU Low-Level Specific and Generic Intrinsics used
Constant Formation Intrinsics.	`spu_splats`
Arithmetic Intrinsics	`spu_add` `spu_sub`
Compare, Branch and Halt Intrinsics	`spu_cmpgt`
Bits and Mask Intrinsics	`spu_sel`

Logical Intrinsics	`spu_or`
	`spu_and`
	`spu_nor`
	`spu_nand`

Shift and Rotate Intrinsics	`spu_slqwbyte`
	`spu_rlmaskqwbyte`
	`spu_rlmaska`

Scalar Intrinsics	`spu_extract`

A list of SPU Low-Level Specific and Generic Intrinsics used in our vectorized implementation, divided into categories, is shown in Table 6. Constant Formation Intrinsics, Arithmetic Intrinsics, Compare, Branch and Halt Intrinsics, Bits and Mask Intrinsics, Logical Intrinsics, Shift and Rotate Intrinsics and Scalar Intrinsics have been employed to access hardware features, which are not easily accessible from a high level language in order to obtain the best performance from the Cell/BE. More details about the syntax and semantics of these Intrinsics can be found in [141].

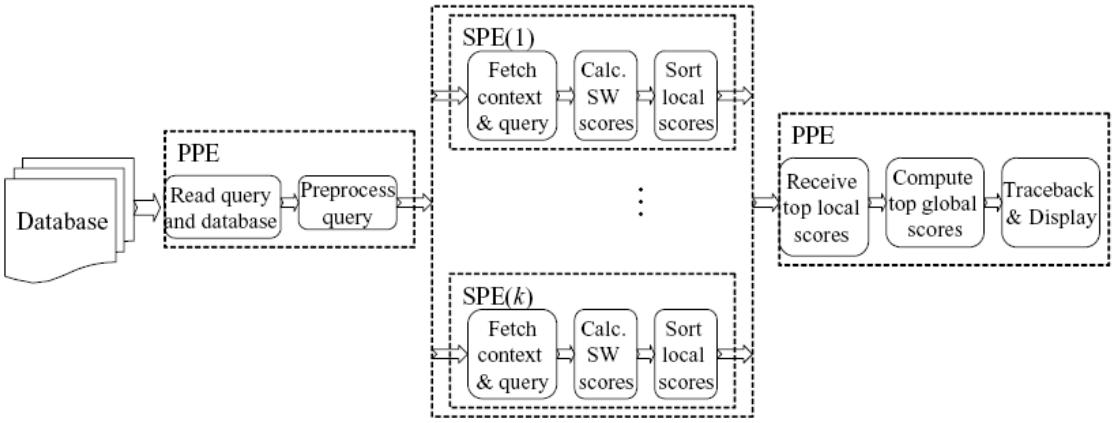

Figure 26. Mapping of the different stages of the CBESW implementation

Figure 26 illustrates the mapping of different stages of SW-based protein sequence database scanning application onto the Cell/BE. The PPE starts by reading the query and the database from the respective files and then pre-processes the query sequences such that they are suitable for vector operations. The pre-processed query sequence, together with some context data, is sent to each respective SPEs, which in turn will generate its own query profile. This process is done using DMA transfers, namely *mfc_get* and *mfc_put*.

Given a database *D* consisting of |*D*| sequences and *k* SPEs. Each SPE aligns the query sequence to the database sequences. Pseudocode of the mapping is illustrated in Figure 27. Scores obtained from those alignments are sorted locally in the SPEs and the *b* highest scores are sent to the PPE, where they are sorted once again to obtain the *b* overall highest scores.

Input:
Number of SPEs used *num*, Query sequence *Q*, Database sequences *D*.

Output:
Global maximum scores for the optimum local alignment of *Q* and *D*

SPE Pseudocode:

```
Start;
    Initialize;

    Fetch context data from PPE using DMA transfer;
    Fetch query sequence Q;
    Fetch the first database sequence Di, for each
      respective SPEs;

    While there is still work to be done{
      Compute score between Q and Di;
      Sort score;
       Fetch next database sequence Di+num;
    }

    Send highest scores to PPE using DMA transfer;
End;
```

Figure 27. Pseudocode of the SPE code for the Cell/BE mapping

Due to the fact that the SPEs only have 256 Kbytes of local memory, which have to store program code and data, memory allocation is crucial for the SPE. The current longest sequence in the Swiss-Prot database is 35,213 amino acids (accession number A2ASS6). In order to accommodate for longer protein sequence in the future, we allocate dynamic memory for the database sequences of up to 64,000 amino acids per sequence. Due to

these limitations, the maximum query sequence length allowed for our implementation is limited to 852.

5.3.2. QUERY PROFILE

In order to calculate $M(i,j)$ in the SW DP matrix, the value $sbt(S_1[i], S_2[j])$ needs to be added to $M(i–1, j–1)$. To avoid performing this table lookup for each element in the DP matrix, Rognes[138] and Farrar [137] suggested calculating a *query profile* parallel to the query sequence beforehand.

Assuming that $S_1, S_2 \in \Sigma^*$ and S_1 is the query sequence, the query profile is defined as a set $P = \{P_x \mid x \in \Sigma\}$ consisting of $|\Sigma|$ numerical strings of length l_1 each, where $l_1 = |S_1|$. Each string $P_x \in P$ consists of all substitution table values that are needed to compute a complete column j of the DP matrix for which $S_2[j] = x$. Pre-computing the query profile greatly reduces the amount of substitution table lookup in the SW DP matrix computation, since $|\Sigma|$ is usually much smaller than $|S_2|$.

The query profile can be calculated in a straightforward *sequential layout* [138] or in a more complex *striped layout* [137], as shown in Figure 28. The values in the query profile for sequential and striped layout are defined in Equation 13 and 14, respectively:

$$P_x[i] = sbt(S_1[i],x), \text{ for all } 1 \le i \le l_1,$$

Equation 13. Query profile equation for sequential layout

$$P_x[i] = sbt\left(S_1\left[\left(\left((i-1)\%p \right)t \right) + \left\lfloor \frac{i-1}{p} \right\rfloor + 1 \right], x \right) \text{ for all } 1 \le i \le l_1$$

Equation 14. Query profile equation for striped layout

where p is the number of segments and t is the segment length.

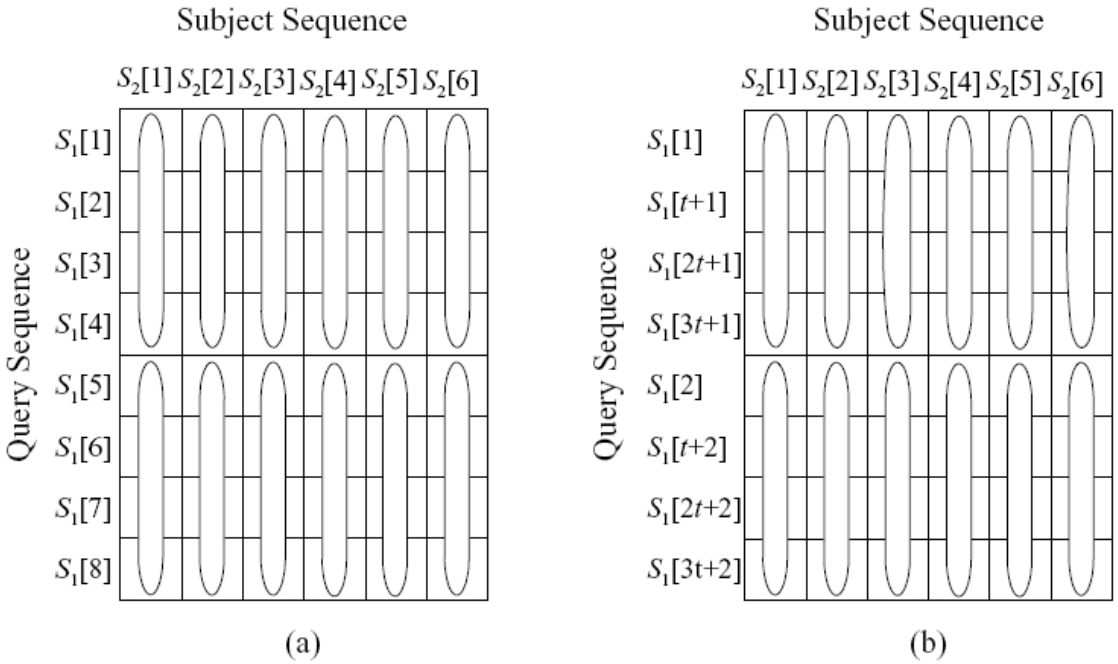

Figure 28. Query profile layout

In the striped layout, p corresponds to the number of elements that can be processed in a SIMD vector register (e.g. for 128-bit wide SIMD registers, $p = 8$ when using 16-bit precision). The length of each segment, t is defined in Equation 15.

$$t = \lceil (l_1 + p - 1)/p \rceil$$

Equation 15. Segment length equation used for the query profile calculation

Both approaches allow efficient vectorization on SSE2-compatible processors using the corresponding SIMD instruction set. Using the pre-calculated query profile, the computation of the DP matrix can be performed in column-wise order. Due to the simplified dependency relationship and parallel loading of the vector scores from memory, fast DP matrix calculations can be achieved. The advantage of the striped layout

compared to the sequential layout is that data dependencies between vector registers are moved outside the inner loop. For instance, when calculating vectors for the DP matrices *H* or *F* with the sequential layout, the last element in the previous vector has to be moved to the first element in the current vector. When using the striped query layout, this needs to be done just once in the outer loop when processing the next subject sequence character.

5.3.3. SATURATION ARITHMETHIC

The inner loop of the algorithm requires saturation arithmetic, namely saturated additions and saturated subtractions. The Cell/BE lacks the saturation arithmetic support, leaving the tasks to be handled by software instead of direct hardware support. In order to tackle this problem, we introduced two new functions, namely `spu_adds` and `spu_subs` to handle saturated additions and saturated subtractions, respectively.

5.4. PERFORMANCE EVALUATION

In this section, we analyze the performance of our parallel algorithm for various query sequence lengths using sequences from Swiss-Prot database. Searches for 18 query sequences with various lengths between 63 to 852 amino acids were performed. The accession numbers of the query sequences used are O29181, P03630, P02232, P05013, P14942, P00762, P53675, Q8ZGB4, P10318, P07327, P01008, P10635, P58229, P25705, P42357, P21177, Q38941 and O60341, respectively. All queries were run against Swiss-Prot release 55.2 comprising 130,497,792 amino acids in 362,782 sequence entries. The gap-open penalty used was 10 and the gap-extension penalty used was 2. The scoring

matrix used in the testing was BLOSUM45. All experiments were carried out on a standalone PlayStation® 3 machine, with Yellow Dog Linux 5.0 operating system and the Cell Software Development Kit (SDK) 2.0.

The performance statistics measured are then converted to the following measurements, i.e. computational time and Million Cell Updates Per Second (MCUPS). Given a query sequence of size Q and a database of size D, the MCUPS rating (million cell updates per second) is calculated by Equation 16.

$$\frac{|Q| \times |D| \times 10^6}{t}$$

Equation 16. MCUPS calculation equation

where

$|Q|$ = size of query sequence in amino acids

$|D|$ = size of database sequences in amino acids

t = run time (including input from file, initialization and result output)

Table 7 shows the performance evaluation of our implementation, in terms of computational time and MCUPS. All queries were run against Swiss-Prot release 55.2 comprising 130,497,792 amino acids in 362,782 sequence entries. Eighteen query sequences of length 63 to 852 amino acids were used. The gap-open penalty used is 10 and the gap-extension penalty used was 2. The BLOSUM45 scoring matrix was used. Our CBESW implementation scales well with the number of activated SPEs, as can be seen from the experiments using 2, 4 and 6 SPEs. By using all 6 SPEs available in the PS3, our parallel algorithm reaches a peak performance of 3,646.48 MCUPS for a query sequence of length 852 (accession number O60341).

Table 7. CBESW Performance Evaluation

Accession number	Query Sequence Length	CBESW 2 SPEs (seconds)	CBESW 4 SPEs (seconds)	CBESW 6 SPEs (seconds)	CBESW 6 SPEs (MCUPS)
O29181	63	24.56	20.83	18.45	445
P03630	127	38.14	24.67	19.05	869
P02232	143	40.46	25.83	19.17	973
P05013	189	44.59	26.94	19.60	1,258
P14942	222	47.96	27.96	20.12	1,439
P00762	246	49.77	28.33	20.24	1,586
P53765	255	50.37	28.60	20.43	1,628
Q8ZGB4	361	55.01	30.85	22.04	2,137
P10318	362	55.16	30.88	22.06	2,141
P07327	374	57.63	31.34	22.39	2,179
P01008	464	60.89	32.45	23.18	2,612
P10635	497	62.18	33.16	23.69	2,737
P58229	511	64.20	34.20	24.43	2,729
P25705	553	65.02	34.63	24.74	2,916
P42357	657	70.02	37.29	26.64	3,218
P21177	729	73.76	39.28	28.06	3,390
Q38941	850	80.15	42.62	30.45	3,642
O60341	852	80.25	42.68	30.49	3,646

We have compared the performance of our CBESW implementation with other publicly available implementations of SW-based protein database scanning, namely SSEARCH[142], Striped Smith-Waterman[137], CUDA[92] and CUDASW++ v1.0[97]. Performance comparison between our CBESW implementation with other implementations are in terms of MCUPS. All queries were run against Swiss-Prot release 55.2. The query sequences, as well as their respective Swiss Prot accession numbers, used in the different performance comparisons are shown in Table 8.

Table 8. List of query sequences used in different performance comparisons

Accession number	Query Sequence Length	SSEARCH	Striped SW	CUDA	CUDASW++
O29181	63	√	√	√	√
P03630	127	√	√	√	√
P02232	143			√	√
P05013	189			√	√
P14942	222			√	√
P00762	246			√	√
P53765	255	√	√	√	√
Q8ZGB4	361	√	√	√	√
P10318	362			√	√
P07327	374			√	√
P01008	464			√	√
P10635	497			√	√
P58229	511	√	√		√
P25705	553			√	√
P42357	657	√	√	√	√
P21177	729	√	√	√	√
Q38941	850	√	√	√	√
O60341	852	√	√	√	√

SSEARCH[142] is a SW implementation which is part of the FASTA[143] package. The SSEARCH performance is benchmarked on an Intel Core 2 Duo 2.4 GHz CPU with 1GB RAM. Both execution cores were used in the experiment. Nine query sequences with lengths of 63 to 852 amino acids and the BLOSUM45 scoring matrix were used. As shown in Figure 29, for a query sequence of length 852 (accession number O60341), SSEARCH achieves a performance of 121.91 MCUPS. Thus, our implementation is over 30 times faster.

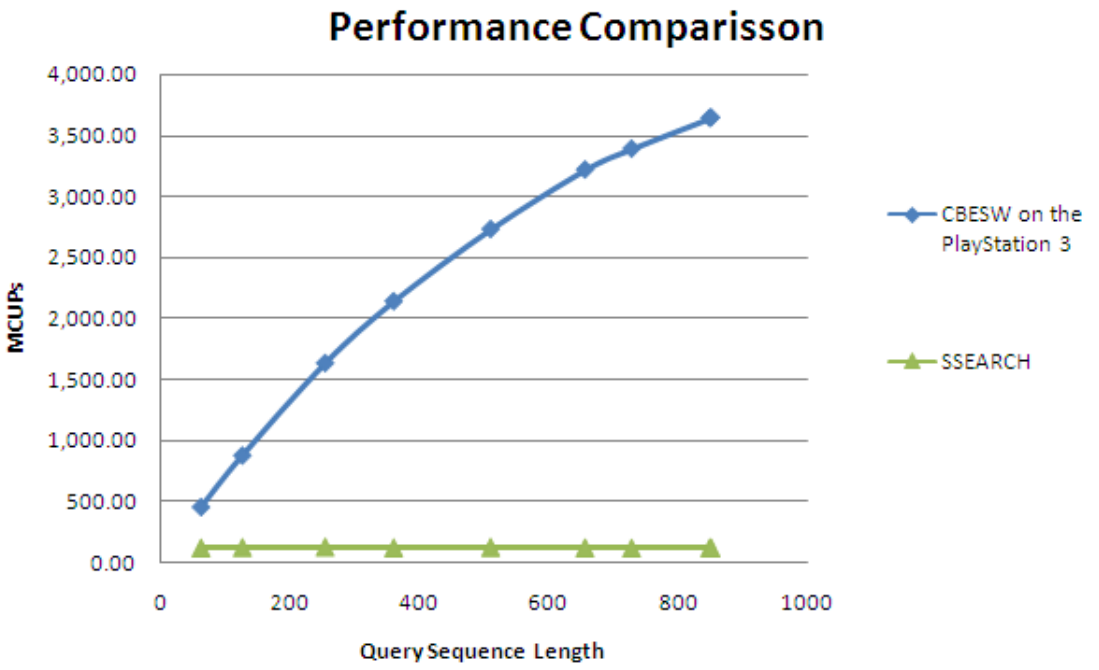

Figure 29. Performance comparison with the SSEARCH implementation

Figure 30 shows the performance comparison between PS3 and striped SW. Striped SW is also benchmarked on an Intel Core 2 Duo 2.4 GHz CPU with 1GB RAM. Both execution cores were used in the experiment. Nine query sequences with lengths of 63 to 852 amino acids and the BLOSUM45 scoring matrix were used. As can be seen from the figure, for query sequences with length > 255 amino acids, our PS3 implementation achieves a higher MCUPS performance compared to striped SW. The PS3 peak performance is 1.64 times faster than striped SW for the query sequence of length 852.

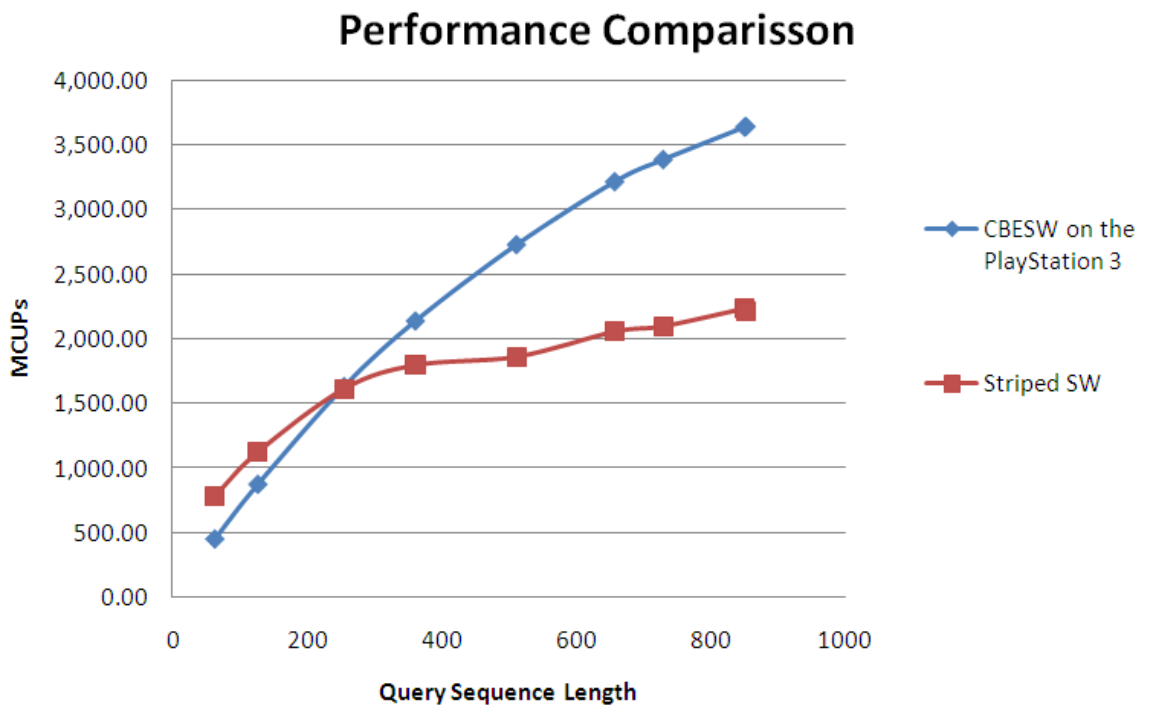

Figure 30. Performance comparison with the Striped Smith-Waterman implementation

The performance comparison between the PS3 implementation and CUDA-SW on a single NVIDIA GeForce 8800GTX is shown in Figure 31. The CUDA implementation experiment was conducted with a GeForce 8800GTX 512 MB installed in a PC with a Dual-Core AMD Opteron 2210 1.8GHz CPU, 2GB RAM running Fedora 6. Seventeen query sequences with lengths of 63 to 852 amino acids were used. The scoring matrix used for the CUDA implementation was BLOSUM 50. As can be seen from the figure, our implementation achieves a better MCUPS performance. The PS3 peak performance is 3 times faster compared to the peak performance CUDA implementation on a single NVIDIA GeForce 8800GTX.

Figure 31. Performance comparison with the CUDA implementation on a single NVIDIA GeForce 8800GTX

CUDASW++ implementation was benchmarked on a single NVIDIA Tesla C1060, consisting of 240 1.3 GHz streaming processor cores, installed in Intel Quad-Core i7-920 2.66GHz CPU, 12GB RAM running Linux Fedora 10. The performance comparison graph is shown in Figure 32. Eighteen query sequences with lengths of 63 to 852 amino acids were used. The scoring matrix used for the CUDA implementation was BLOSUM 50. In average, performance of the CUDASW++ is 4.38 faster compared to our CBESW implementation. As can be seen from the graph, the speed-up of CUDASW++ is more significant at short query length. The speed up obtained by the CUDASW++ compared to CBESW is expected, since Tesla C1060 is based on newer technology than PS3 by almost 4 years difference.

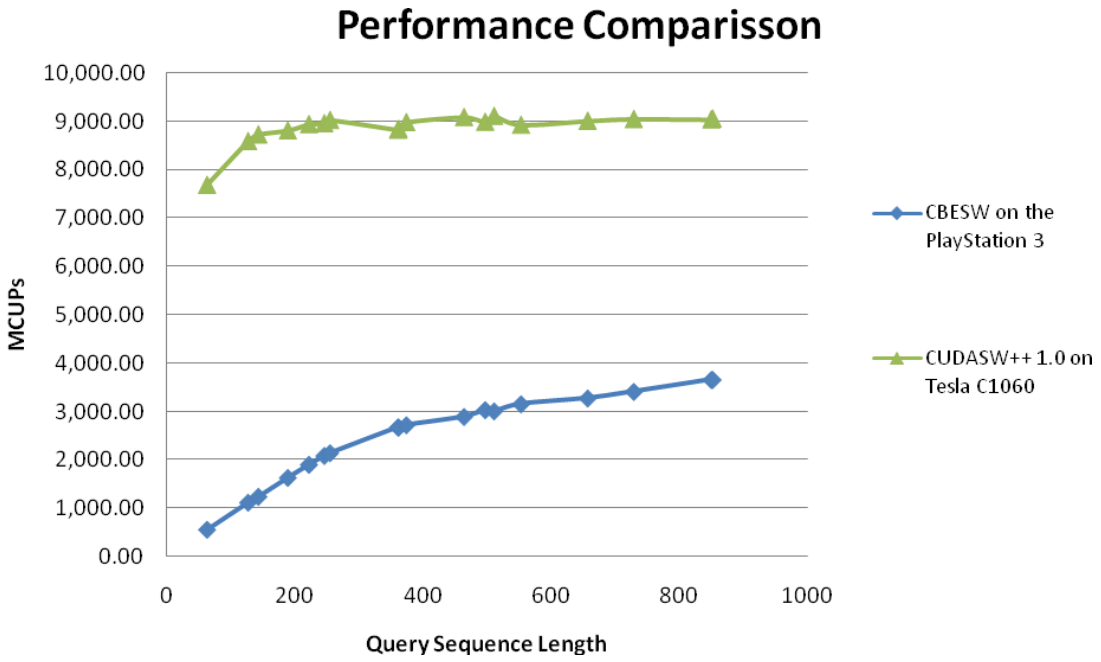

Figure 32. Performance comparison with the CUDASW++ implementation on a single NVIDIA Tesla C1060

5.5. SUMMARY

In this chapter, we have demonstrated that the PlayStation® 3, powered by the Cell Broadband Engine, can be effectively used to accelerate a biological sequence alignment application. In order to derive an efficient mapping onto this type of heterogeneous multi-core architecture, we have utilized SIMD vectorization and parallel data partitioning and communication techniques.

Our implementation achieves a peak performance of 3,646.48 MCUPS for a query sequence of length 852. Hence, the peak performance of our implementation is 30.1 times and 1.64 times faster than SSEARCH and striped SW, on an Intel Core 2 Duo 2.4 GHz. The CBESW peak performance is also 3 times faster compared to the peak performance CUDA implementation on a single NVIDIA GeForce 8800GTX. Comparison to

CUDASW++ on a single NVIDIA Tesla C1060, which is the one of the latest SW implementations with one of the most recent and powerful GPU, shows that it is 4.38 faster compared to our CBESW implementation.

The very rapid growth of biological sequence databases demands even more powerful high-performance solutions in the near future. Hence, our results are especially encouraging since high performance computer architectures are developing towards heterogeneous multi-core systems.

Due to the 256 KB memory limitation of the SPE local store, the maximum query sequence length in our current implementation is 852. One of the limiting factors is that the size of the query profile grows with the length of the query sequence. Part of our future work is therefore to tackle this limitation. A promising approach is to align subject sequences against separate chunks of the query profile. The complete query profile only needs to be stored once in the main memory instead of the local store of the SPE. This frees up more memory space for the SPEs and thus allows longer query sequences. Given a query sequence of length l, the query profile can be divided into n chunks in which each chunks contains a query profile of size l/n. The respective SPEs can then align a part of the chunk of the query profile it has and get the next chunk from outside memory via concurrent DMA transfer.

6. IMPLEMENTATION OF A HEURISTIC PROTEIN SEQUENCE DATABASE SCANNING ALGORITHM ON THE CELL/BE

This chapter discusses the implementation of a heuristic protein sequence database scanning algorithm, the BLASTP heuristic, on the Cell/BE. Furthermore, a new parallel communication pattern and a novel data structure utilized in the implementation are explained in detail. Lastly, performance comparisons of our Cell/BE BLASTP implementation on the Playstation®3 to the sequential FSA-BLASTP and NCBI-BLASTP implementations are presented.

6.1. INTRODUCTION

Scanning genomic sequence databases is a common and often repeated task in molecular biology. The scan operation consists of finding similarities between a particular query sequence and all sequences of a bank. There are two basic algorithmic approaches to perform this scanning i.e. exhaustive dynamic programming and heuristic algorithm. Heuristic algorithm in general produces the result more rapidly compared to the exhaustive approach, although it does not guarantee an optimal result.

The computational complexity of the exhaustive approach is quadratic with respect to the lengths of the alignment targets (query sequence and subject sequence). In order to reduce the complexity, filtration has been introduced as a heuristic at the cost of a generally lower quality of the results[144]. Filtration assumes that good alignments

usually contain short exact matches. Such matches can be quickly identified using data structures such as lookup tables. Identified matches are then used as seeds for further detailed analysis. Several filtration tools for sequence database searching have been introduced, e.g. [69, 145, 146]. Among them, BLAST (the *B*asic *L*ocal *A*lignment *S*earch *T*ool[45, 69]) is the most popular software and is used to run millions of queries each day. Previous work on parallelizing BLASTP has focused on distributed memory architectures such as clusters[147] and reconfigurable hardware[148, 149].

In this chapter, we present new approaches to parallelize the scanning of protein databases using the BLASTP heuristic on the Cell/BE processor. This implementation is to our knowledge the first ever reported parallelization of BLASTP on the Cell/BE.

6.2. BLAST-P ALGORITHM

The basic idea for fast sequence database search is *filtration*. Filtration assumes that good alignments usually contain short exact matches. Such matches can be quickly computed by using data structures such as lookup tables. Identified matches are then used as seeds for further detailed analysis. The analysis pipeline of the BLASTP algorithm is shown in Figure 33. It consists of four stages. Each stage progressively reduces the search space in the database for significant alignment. We briefly describe each step in the following. More details can be found in[45, 69].

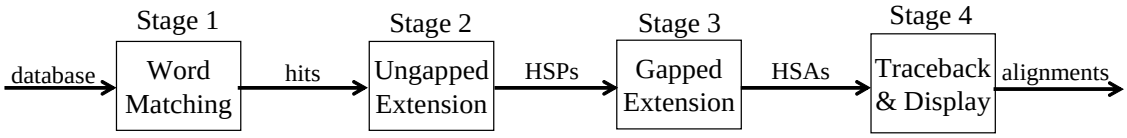

Figure 33. The BLASTP processing pipeline

Stage 1: This stage identifies *hits*. Each hit is defined as an offset pair (i,j) for which $\sum_{k=0}^{w-1} sbt(Q[i+k],D[j+k]) \geq T$, where *sbt* is an amino acid substitution matrix (e.g. BLOSUM65), w is the user-defined word length, T is a user-defined threshold, Q is the query sequence and D is the database. BLASTP implements this stage by preprocessing Q as follows. For each position i of Q the neighborhood $N(Q[i\ldots i+w-1],T)$ is computed consisting of all w-mers p for which $\sum_{k=0}^{w-1} sbt(Q[i+k],p[k]) \geq T$. The complete neighborhood of a query is typically stored in an efficient data structure such as a lookup table or a finite-state automaton. The default parameter values are w=3 and T=11.

Stage 2: Stage 2 outputs *HSPs* (high-scoring segment pairs) between Q and D. HSPs are identified by performing an ungapped extensions on a diagonal d which contains a non-overlapping hit pair (i_1,j_1), (i_2,j_2) within a window A; i.e. $d = i_1 - j_1 = i_2 - j_2$ and $w \leq i_2 - i_1 \leq A$. If the resulting ungapped alignment scores above a certain threshold it is passed to Stage 3.

Stage 3: This stage outputs *HSAs* (high scoring alignments) between Q and D. HSAs are identified by performing a seeded banded gapped dynamic programming based alignment algorithm using the previously identified HSPs as seeds. Alignments that score above a certain threshold are then passed to the final stage.

Stage 4: The final alignments of the highest scoring sequences are calculated and displayed to the user. This requires the computation of the traceback path using the Smith-Waterman algorithm.

An execution profiling of the BLASTP algorithm for scanning the GenBank non-redundant protein database shows the following breakdown of execution time, as shown in Table 9.

Table 9. Breakdown of execution time of BLASTP

Stage	Percentage of Execution Time
1	37%
2	31%
3	30%
4	2%

Hence, in order to efficiently map BLASTP on the Cell/BE all stages except Stage 4 need to be parallelized.

6.3. IMPLEMENTATION

Details of our BLASTP implementation on the Cell/BE are elaborated in this section. We discuss the parallelization approach in detail as well as the mapping of the algorithm to the Cell/BE.

6.3.1. PARALLELIZATION APPROACH

In order to achieve an efficient parallelization of protein sequence database scanning on the Cell/BE. processor, we need to address the following challenges.

1. Limited local storage of the SPE.

   A major limitation when designing SPE kernels is that their local memory is only 256 KByte for both instructions and data. Using default parameter for *w* and *T* the size of the lookup table used for Stage 1 by NCBI BLASTP is already around 400KByte for 100 randomly selected query sequences. Therefore, we need to use an alternative data structure which requires significantly less memory.

2. Data transfer and coordination between PPE and SPEs.

   The different stages of the BLASTP algorithm constitute a processing pipeline where the throughput of each stage in the pipeline depends on the filtration efficiency of the previous stage. Therefore, an efficient and flexible mechanism to transfer sequences from the database to the SPEs needs to be implemented. The PPE needs to coordinates this data transfer.

6.3.2. MAPPING TO THE CELL BROADBAND ENGINE

Figure 34 shows our mapping of the different stages of the BLASTP algorithm onto the Cell/BE. Stage 4 includes a ranking procedure on all database sequences that have passed Stages 1-3: The top 500 or less matching sequences whose scores exceed a certain threshold are displayed in descending order. Thus, this stage is performed by PPE. SPE kernels filter the database as follows. Information about all subject sequences from the database that have passed Stages 1-3 on an SPE are sent to the PPE. Upon receiving this information, the PPE completes Stages 1-4 for these subject sequences. The reason why not only Stage 4 is performed on the PPE is that this stage requires additional information from the previous stages and storing this on the SPEs would be too memory-intensive. However, since this redundant computation is merely performed for very few subject sequences the additional runtime is negligible.

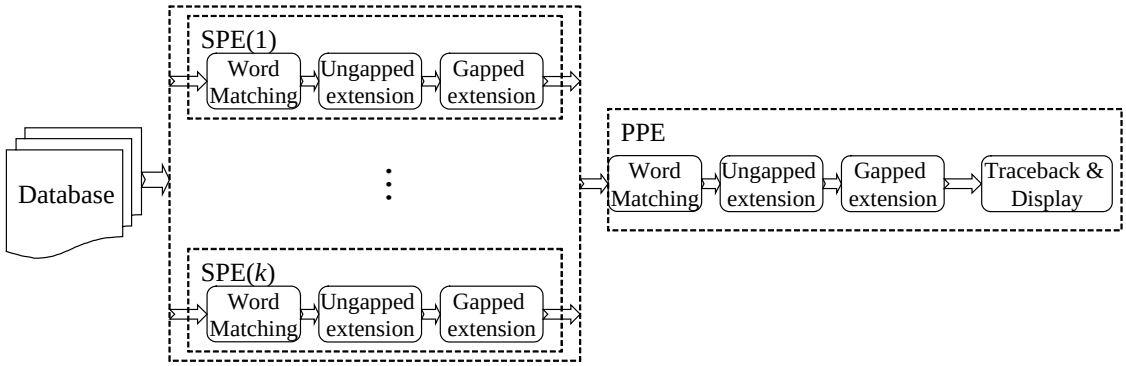

Figure 34. Mapping of the different stages of the BLASTP algorithm onto the Cell/BE

As mentioned above, the size of the codeword lookup data structure used by NCBI BLAST is too large for the local store of the SPEs. Therefore, we are using a more memory-efficient data structure for Stage 1. The utilized data structure is a compressed *deterministic finite-state automaton* (DFA), which is similar to the approach used by FSA-BLAST [72, 73]. The compressed DFA for w=3 is illustrated in Figure 35.

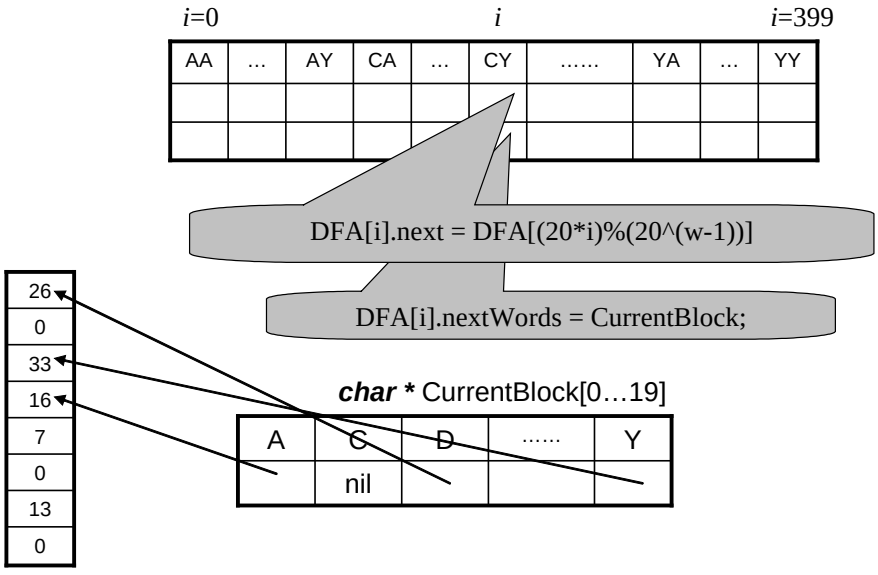

Figure 35. Illustration of the compressed FSA data structure for w=3

Each possible prefix of lengths $w-1$ is represented by a state; i.e. for $w=3$ there are 400 states representing the prefixes *AA* to *YY*, which are stored in the array DFA[i] in Figure 35. Each state has two transitions: one to the next state (DFA[i].next) and one to a list of 20 words (DFA[i].nextWords). Each entry in this list (currentBlock[0..19]) contains a pointer to an array of query positions. These query positions represent the neighborhood $N(w,T)$ of the associated w-mer. This data structure allows the compression of frequently used query positions that are in neighborhoods of similar w-mers. For example in Figure 35, N('CYC',T) = {33, 16, 7} and N={'CYA',T} = {16, 7}. By storing these positions in subsequent order terminated by "0" it is possible to re-use memory for both neighborhoods. Our experiments have shown that the size the compressed DFA is only 43.8 KByte on average. Hence, it is possible to store the complete data structure on each SPE for most queries.

The DFA is transferred into each SPE. The PPE then reads sequences from the database and transfers them to the SPEs by Direct Memory Access (DMA). In order to hide latencies and achieve good load balancing, we have implemented four buffers on the PPE per SPE and two buffers on each SPE, as shown in Figure 36. Our double buffering scheme allows SPEs to receive a new subject sequence through DMA while processing another previously received sequence. The PPE continuously prepares sequence data for free buffers. Once a buffer is filled, the PPE sends a mailbox notification to the corresponding SPE. The number of buffer in the PPE for each SPE is therefore restricted by the size of the SPE's Read Inbound Mailbox (which is four). Furthermore, the PPE dynamically assigns protein sequences to buffers depending on their lengths and the available memory. The maximum number of sequences inside a buffer is 32.

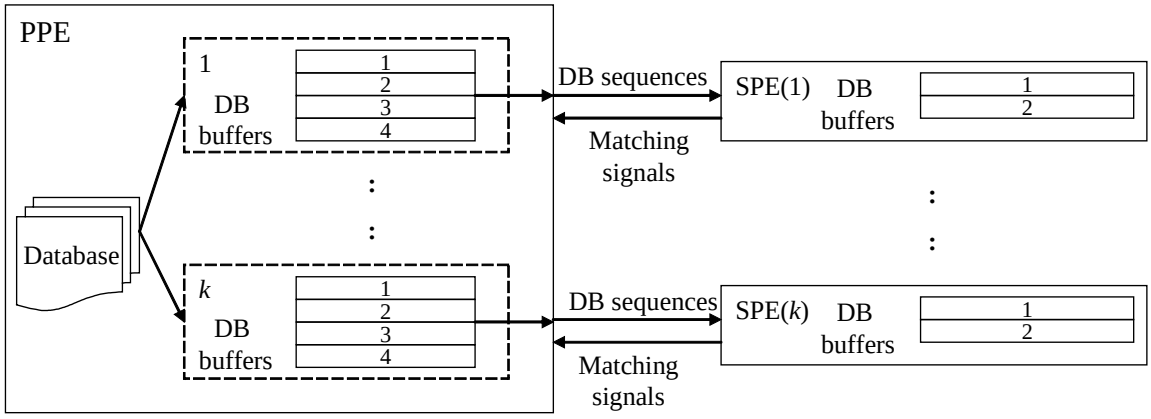

Figure 36. Buffering scheme

All sequences inside a buffer are filtered by Stages 1-3 on one of the SPEs. If a sequence passes all these stages, the corresponding bit in the matching signal (32 bits) is set. After all sequences are processed, this matching signal is sent back to the PPE via a mailbox. The PPE then identifies all sequences that have passed Stages 1-3 on SPE and perform Stages 1-4 on them. Pseudocodes of the programs running on the PPE and each SPE are shown in Figure 37 and 38, respectively.

```
1. Initialization
2. Create DFA
3. Start SPEs and send parameters and DFA lookup table to SPEs
4. Check whether there is mail from SPEs
    If there is a mail
        Collect information of sequences that passed stages 1-3 and
keep in a queue
        Mark the corresponding buffer as free
5. Check whether there is a free buffer
    If a free buffer is found
         Prepare data into it and mark it as occupied
    Else
         Do BLASTP searching stages 1-2 for sequences in the queue
```

```
6. Repeat steps 4-5 until there is no sequence in database
7. Send commands to SPEs to complete last buffered sequences
8. Wait until all buffers are marked as free
9. Do BLASTP stages 3-4
```

Figure 37. Pseudocode of the PPE code

```
1. Initialization
2. Receiving parameters and DFA from PPE
3. Receiving mail with command from PPE
4. If command is new-data-available
       DMA the new data
       If this is the 1st data
           Go to 3
       Else
            Wait for last data to be completely DMA transferred
            Do Stages 1-3 for sequences in the last data
            Return matching signal to PPE through SPU Write Outbound
            Mailbox
            Go to 3
5. If command is finish-last-sequence
       Do Stages 1-3 for sequences in the last data
       Return matching signal to PPE through SPE Write Outbound Mailbox
       Exit
```

Figure 38. Pseudocode of the SPE code

Because of the limited storage of each SPE (256 KBytes) it is important to analyze the associated memory consumption. The size of SPE program is 100KByte. Thus, we have at most 156KByte for storing the DFA data structure, the two buffers as well as other parameters and intermediate results. Hence, we have assigned 10KByte to each buffer and up to 80KByte to the DFA. 80KByte is sufficient for DFAs for query sequences of up to 2000 base-pairs (bps). In our experiment, the average DFA size is 43.8KByte. If the

length of a subject sequence is over 10Kbps, it will be put directly into the sequence queue of the PPE without sending it to an SPE.

Furthermore, some database sequences exceed a certain memory threshold during they are processed on the SPE. Such sequences will be marked and passed to the PPE for further processing. Although, this creates additional work, the number of such sequences is usually negligible. It is also another reason why the PPE performs all stages of the BLASTP algorithm instead of only Stage 4. Furthermore, note that we do not return results of matching sequences from SPEs because we do not want to increase SPE code size by increasing program complexity to return the search results.

6.4. PERFORMANCE EVALUATION

We have implemented the described Cell/BE BLASTP program using Cell/BE SDK 3.0 and evaluated it on a PlayStation®3 (PS3), which contains a Cell/BE as its main processor. In order to evaluate the performance on a PS3, we have installed LINUX version 2.6.23-rc3 (gcc version 4.1.1 20061011 (Red Hat 4.1.1-30)). Please note that on the PS3 two of eight SPEs are used by the operating system running. Therefore, our experiments can only use up to six SPEs.

We have compared the performance of our Cell/BE BLASTP program to FSA-BLASTP (available form www.fsa-blast.org) and NCBI-BLASTP (www.ncbi.nlm.nih.gov/BLAST/developer.shtml). FSA-BLAST uses an optimized sequential algorithm and is around 15% faster than NCBI-BLASTP with no loss in accuracy[72, 73]. FSA-BLASTP and NCBI-BLASTP are tested on a HP workstation xw4200 with Dual-core Pentium®4 (P4) CPU 3.0GHz, 2GB of RAM. Two-hit model [2]

is used for all BLASTP programs. Default values of *W*=3 and *T*=11 are adopted. The produced matching results by FSA-BLASTP and Cell/BE BLASTP are exactly the same. The protein sequence database we used in our experiments is the GenBank Non-Redundant Protein Database (which is downloaded from ftp://ftp.ncbi.nih.gov/blast/db/FASTA/nr.gz), containing 6,375,605 protein sequences. We have chosen 100 random sequences from the database as queries. The lengths of the query sequences are distributed uniformly between 1 and 2000bps.

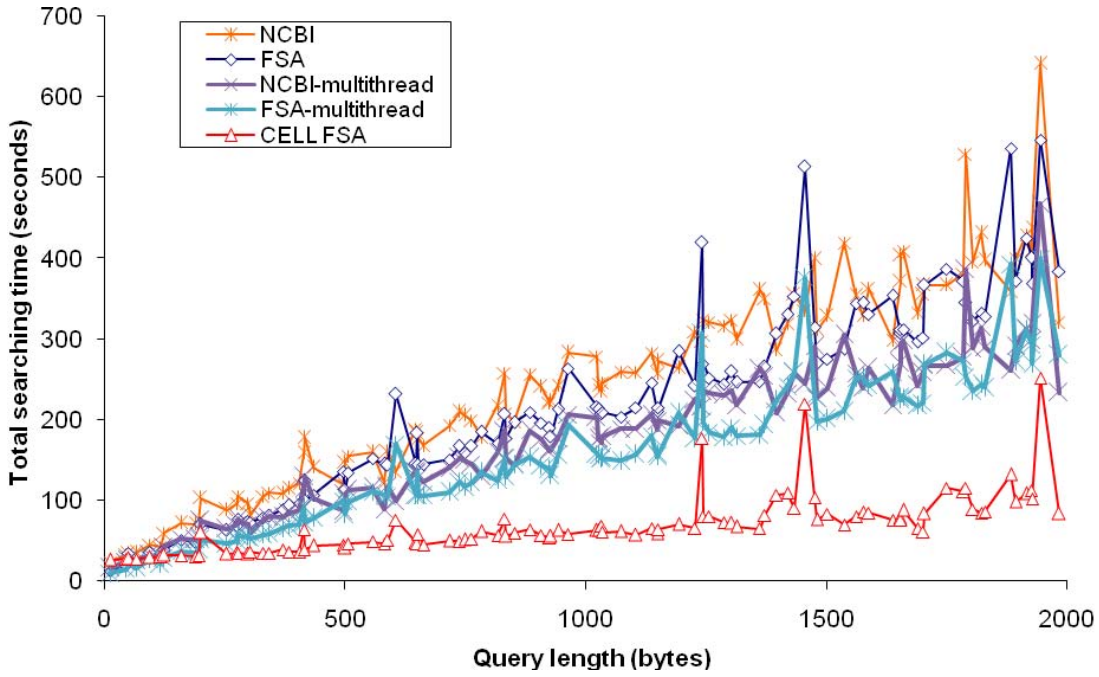

Figure 39. Performance comparison between our Cell/BE BLASTP implementation with the FSA-BLASTP and the NCBI-BLASTP

A performance comparison of the presented parallel Cell/BE BLASTP program to the sequential FSA-BLASTP and NCBI-BLASTP programs are shown in Figure 39. It can be seen that Cell/BE BLASTP is faster than FSA-BLAST in most cases. The average

searching times are 217.5s for FSA-BLASTP, 244.75s for NCBI-BLASTP, and 67.97s for Cell/BE BLASTP. This corresponds to an average speedup of 3.2 and 3.6 respectively. Activating the multithread option improves the average searching times to 159.1s and 178.3s for FSA-BLASTP and NCBI-BLASTP, respectively. This corresponds to an average speedup of 2.3 and 2.6, respectively.

More detailed statistics of the performance comparison are shown in Table 10. The performance is measured in terms of seconds and the speedup of the Cell/BE BLASTP implementation over the FSA-BLASTP implementation. From the table, we can see that Cell/BE BLASTP spends more time on Stage 4. This is because the PPE is a less powerful processor than a P4. The speedup of Cell/BE BLASTP mostly comes from stage 1-3 which are running on six PPEs of the PS3 in parallel.

Table 10. Performance comparison between Cell/BE BLASTP and FSA-BLASTP

Query length range	FSA-BLASTP				Cell/BE BLASTP				Speedup
	Stages 1-2	Stage3	Stage4	Total	Stages 1-2	Stage 3	Stage 4	Total	
1-300	40.1	5.66	0.30	46.5	28.9	1.77	0.74	32.9	1.41
301-500	74.0	23.09	0.32	97.8	35.4	3.10	0.81	40.9	2.39
501-800	110.3	46.57	0.50	157.8	44.5	4.30	1.10	51.5	3.06
801-1100	151.0	50.98	0.92	203.4	52.8	4.74	1.83	61.1	3.33
1101-1400	183.0	76.32	1.80	261.6	61.8	10.25	4.18	79.0	3.31
1401-1700	216.9	109.01	3.22	329.6	67.2	15.19	7.98	92.4	3.57
1701-2000	241.8	141.53	2.02	385.9	83.9	18.77	4.57	109.0	3.54

The average number of sequences that are processed in each stage by FSA-BLASTP and in the PPE by Cell/BE BLASTP are shown in Table 11. In FSA-BLASTP, every database sequence is processed by Stages 1-2. The PPE in Cell/BE BLASTP only processes a very small faction of database sequences since most sequences have been filtered by SPEs in

parallel. This reduced number of sequences contributes to the less total runtime of Cell/BE BLASTP. However, the ideal speedup of around six is not reached since the parallel SPE filters add some data transfer and coordination overhead and the PPU is less powerful than a P4. It should also be noted that the speedup for shorter query sequences is generally lower since the runtime is too short to effectively compensate for the associated overheads. Furthermore, the number of database sequences for the Cell/BE BLASTP implementation is larger than the number of found matching sequences. This can be explained as follows. Firstly, if a sequence is too long to be sent to the SPE, it will be processed by the PPE directly. In the experiment, 72 sequences are longer than the maximum buffer length (10KByte). Secondly, some sequences in Stages 1-3 in the SPE exceed the maximum available memory space. These sequences are returned as matches and need further processing on the PPE.

Table 11. Average number of sequences processed by each stage of FSA-BLASTP on a P4 and by the PPE in Cell/BE BLASTP

Query length	FSA-BLASTP			Cell/BE BLASTP (only on PPE)			Matching output
	Stages1-2	Stage3		Stages1-2	Stage3		
		Semi	Gapped		Semi	Gapped	
1-300	6,375,605	96954	9443	2113	2062	1731	328
301-500		334494	13749	2591	2570	1462	324
501-800		617225	19602	5480	5471	3713	443
801-1100		586139	24163	5408	5402	3569	471
1101-1400		761097	34028	7193	7189	5178	443
1401-1700		1096186	43616.1	15404	15402	12901	438
1701-2000		1206705	38761	6734	6733	4126	428

In addition, some query sequences require more processing time by both FSA-BLASTP and Cell/BE BLASTP than queries of similar lengths. The runtime statistics of the three

such exceptional sequences is shown in Table 12. It can be seen that for these three queries, a bigger number of database sequences need to be processed than average. This increases both CPU and PPE workload.

Table 12. Runtime statistics of three exceptional sequences.

Query length	Method	Time				
		Stages 1-2	Stage3		Stage4	Total
			Semi	Gapped		
605	FSA-BLAST	63.55	160.89	6.40	0.80	232.13
	Cell/BE	58.65	11.63	1.85	1.88	75.61
1455	FSA-BLAST	138.97	348.58	0.38	24.96	513.43
	Cell/BE	84.48	65.49	1.28	66.17	219.43
1945	FSA-BLAST	225.87	316.33	1.02	2.63	546.35
	Cell/BE	132.11	109.53	1.36	6.98	251.52

Query length	Method	Number of sequences			Matching output
		Stages 1-2	Stage3		
			Semi	Gapped	
605	FSA-BLAST	6,375,605	1,890,358	33,061	500
	Cell/BE	5,536	5,536	2,288	500
1455	FSA-BLAST	6,375,605	2,981,242	23,895	500
	Cell/BE	8,344	8,344	4,115	500
1945	FSA-BLAST	6,375,605	1,555,474	170,541	500
	Cell/BE	27,681	27,677	25,473	500

6.5. SUMMARY

In this chapter, we have presented parallelization strategies for scanning protein sequence databases on the Cell/BE. using the BLASTP heuristic. In order to derive efficient mappings onto this type of heterogeneous multi-core architecture, we have utilized SIMD vectorization, parallel data partitioning and communication schemes, and a compressed

deterministic finite state automaton for hit detection in order to reduce memory consumption. Our BLASTP implementation on a PS®3 achieves an average speedup of 3.2 compared to the optimized FSA-BLASTP and 3.6 compared to NCBI-BLASTP. The very rapid growth of biological sequence databases demands even more powerful high-performance solutions in the near future. Hence, our results are especially encouraging since high performance computer architectures are developing towards heterogeneous multi-core systems. Therefore, the techniques presented in this chapter are of particular importance since they compare and analyze the efficiency of parallelization approaches on different parallel architectures.

.

7. PAIRWISE DISTANCE MATRIX COMPUTATION

This chapter elaborates our parallel implementation that accelerates the distance matrix computation used in multiple sequence alignments on the x86 and Cell Broadband Engine architecture, a homogeneous and heterogeneous multi-core system, respectively. Furthermore, we compare the performance of our implementation on the Playstation®3 with other accelerator technologies, i.e. FPGA and GPU.

7.1. INTRODUCTION

Multiple sequence alignment (MSA) of many nucleotides or amino acids is an important tool in bioinformatics. It identifies patterns or motifs to characterize protein families, and is therefore utilized to detect homology between sequences as well as to perform phylogenetic analysis. Previous work on MSA heuristics to reduce the exponential complexity of computing optimal MSAs include MSA[46], ClustalW[58], T-Coffee[47], MAFFT[150], DIALIGN P[151] and PRALINE[63].

ClustalW is considered to be one of the most popular MSA tools. It is based on the progressive alignment method. Software approaches to improve the performance of ClustalW have been introduced, including caching [152, 153] and parallel processing [18, 154, 155]. Recent usage of easily accessible accelerator technologies to improve the ClustalW algorithm include FPGA[156] and GPU[93].

Our profiling of ClustalW reveals that the distance matrix computation is the most time consuming phase and takes typically more than 90% of the overall runtime. Therefore, accelerating this phase would greatly improve the performance as a whole.

In this chapter, we introduce our implementation that accelerates the distance matrix computation on the Cell/BE and the commonly used Intel x86 architecture.

7.2. MULTIPLE SEQUENCE ALIGNMENT ALGORITHM

ClustalW[58] has over 26,000 citations in the ISI Web of Science and is considered to be one of the most popular MSA tools. It implements a progressive alignment method[56], i.e. it adds sequences one by one to the existing alignment to build a new alignment. The order of the sequences to be added to the new alignment is indicated by a pre-computed phylogenetic tree, which is called a guide tree. The guide tree is constructed using the similarity of all possible pairs of sequences stored in the distance matrix.

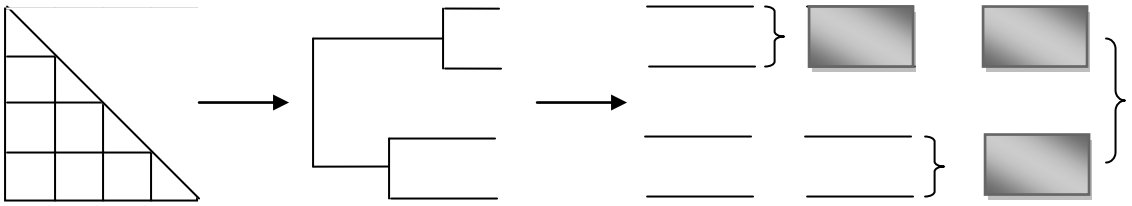

Figure 40. The three stages of the ClustalW algorithm. (a) Distance matrix computation. (b) Guide tree construction. (c) Progressive alignment.

The ClustalW algorithm consists of 3 phases, as shown in Figure 40:

1. Distance matrix computation:

   Each pairs of sequences are aligned separately to calculate their respective distance values. These values are stored in a so-called distance matrix.

2. Guide tree construction:

The guide tree is calculated from the distance matrix using a neighbor joining algorithm[157]. The guide tree defines the order which the sequences are aligned in the next stage.

3. Progressive alignment:

   The sequences are progressively aligned in accordance to the guide tree.

Given *n* number of sequences of length m, the distance matrix computation has a quadratic complexity of $O(n^2m^2)$. Profiling the three the above mentioned phases of ClustalW using *gprof* also shows that the distance matrix computation is the most computationally intensive phase and takes up more than 90% of the overall runtime. Hence, it can be concluded that accelerating the distance matrix computation would provide a good speed up for the ClustalW.

Given a set of *n* sequences $\boldsymbol{S} = \{S_1, S_2, \ldots, S_n\}$, for two sequences $S_i, S_j \in \boldsymbol{S}$, the distance value $d(S_i,S_j)$ can be defined as Equation 17 below:

$$d(S_i, S_j) = 1 - \frac{nid(S_i, S_j)}{\min\{l_i, l_j\}}$$

Equation 17. Distance value equation

where $nid(S_i,S_j)$ denotes the number of exact matches in the optimal local alignment of S_i and S_j with respect to the given scoring system and l_i and l_j denotes the length of S_i and S_j, respectively.

Liu et.al.[93] states that given two sequences S_1 and S_2 with affine gap penalties α and β and the substitution table *sbt*, a matrix $N_A(i,j)$ ($1 \le i \le l_1$, $1 \le j \le l_2$) can be recursively defined as shown in Equation 18.

$$N_A(i,j) = \begin{cases} 0 & ,if\ \ H_A(i,j) = 0 \\ N_A(i-1,j-1) + m(i,j) & ,if\ \ H_A(i,j) = H_A(i-1,j-1) + sbt(S_1[i], S_2[j]) \\ N_E(i,j-1) & ,if\ \ H_A(i,j) = E(i,j) \\ N_F(i-1,j) & ,if\ \ H_A(i,j) = F(i,j) \end{cases}$$

where

$$H_A(i,j) = \max\begin{cases} 0, \\ E(i,j), \\ F(i,j), \\ H_A(i-1,j-1) + sbt(S_1[i], S_2[j]) \end{cases}$$

$$E(i,j) = \max\{H_A(i,j-1) - \alpha, E(i,j-1) - \beta\}$$

$$F(i,j) = \max\{H_A(i-1,j) - \alpha, F(i-1,j) - \beta\}$$

$$m(i,j) = \begin{cases} 1, & if\ \ S_1[i] = S_2[j] \\ 0, & otherwise \end{cases}$$

$$N_E(i,j) = \max\begin{cases} 0 & ,if\ \ j = 1 \\ N_A(i,j-1) & ,if\ \ E(i,j) = H_A(i,j-1) - \alpha \\ N_E(i,j-1) & ,if\ \ E(i,j) = E(i,j-1) - \beta \end{cases}$$

$$N_F(i,j) = \max\begin{cases} 0 & ,if\ \ i = 1 \\ N_A(i-1,j) & ,if\ \ F(i,j) = H_A(i-1,j) - \alpha \\ N_E(i-1,j) & ,if\ \ F(i,j) = E(i-1,j) - \beta \end{cases}$$

Equation 18. Recurrence relation equation by Liu et. al.

For local alignment of sequences S_1 and S_2, given affine gap penalties α and β and the substitution table *sbt*, the *nid* (S_1,S_2) equation can be modified as shown in Equation 19.

$$nid(S_1, S_2) = N_A(i_{\max}, j_{\max})$$

Equation 19. Modified nid score equation

where (i_{max},j_{max}) denote the coordinates of the maximum value in the corresponding matrix H_A.

Thus, the distance value $d(S_i,S_j)$ can then be redefined as shown in Equation 20.

$$d(S_i, S_j) = 1 - \frac{N_A(i_{\max}, j_{\max})}{\min\{l_i, l_j\}}$$

Equation 20. Modified pairwise distance value equation

A more detailed explanation and proof of these formulas is described in[93].

7.3. MAPPING TO THE CELL/BE

This section explains the mapping of our Cell/BE implementation in details in terms of three subgroups, i.e. query profile, SIMD vectorization and multi-threading.

7.3.1. QUERY PROFILE

To speed up the computation, a query profile is pre-computed. A query profile is computed only once for the entire search and will save one memory lookup in the inner loop of the algorithm. Instead of indexing the original substitution matrix by the query sequence symbol and the database sequence symbol, the query profile is indexed by the query sequence position and the database sequence symbol. It contains the substitution score for matching each of the possible amino acid symbols with each symbol in the query sequence. The scores for matching a residue with each residue in the query sequence is followed by the scores for matching the next residue with each residue in the query sequence, and so on. Using this method, therefore, random accesses to the substitution matrix due to table lookup is replaced with sequential ones to the query. Figure 41 shows an example of a query profile for *Lysine-specific histone demethylase 1* protein (Swiss-Prot accession numbers O60341) with BLOSUM50 scoring matrix.

	M	L	S	…	P	S	M
A	-1	-2	+1	…	-1	+1	-1
B	-3	-4	0	…	-2	0	-3
C	-2	-2	-1	…	-4	-1	-2
⋮	⋮	⋮	⋮		⋮	⋮	⋮
X	-1	-1	-1	…	-1	-1	-1
Y	0	-1	-2	…	-3	-2	0
Z	-1	-3	0	…	-1	0	-1

Figure 41. Example of a query profile for Lysine-specific histone demethylase 1 protein (Swiss-Prot accession numbers O60341) with BLOSUM50 scoring matrix

For the Cell/BE implementation, the query profile computation is done in the PPE and is distributed to the respective SPEs using DMA transfer. For the SSE2 implementation, each thread contains its respective query profile information need to complete the computation.

7.3.2. SIMD-SPECIFIC IMPLEMENTATIONS

Our Cell/BE implementation takes advantage of the 128-bit Single Instruction Multiple Data (SIMD) vector registers of each SPEs. The Cell/BE mapping uses half word values (16 bits) for the computation, which is the smallest element supported by the Cell/BE instruction set. This allows eight cells to be processed per vector register. SPU intrinsics[141] are used improve the efficiency of the program. The SPE pseudocode is shown in Figure 42.

```
Initialization;
Fetch the context data from the mailbox;
Fetch the set of sequences using DMA transfer;
While there are sequences to be processed
      Calculate nid score;
Compile the nid scores into a list nidlist;
Send nidlist to PPE using DMA transfer;
```

Figure 42. Pseudocode of the SPE code

Based on Equation 20, $nid(S_i,S_j)$ is computed without computation of the actual traceback. Since all elements in the same minor diagonal of the dynamic programming matrix can be computed independent of each other in parallel, the computation is done in *minor diagonal order*, as illustrated in Figure 43.

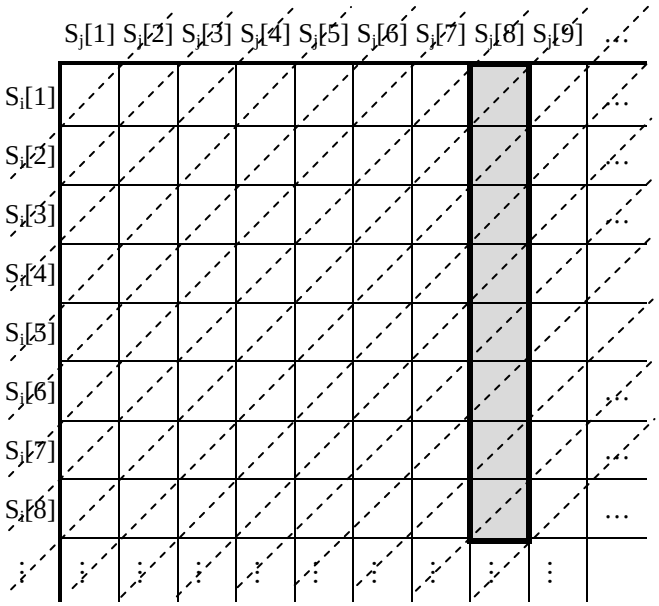

Figure 43. Block diagram of our pairwise distance matrix implementation

The minor diagonal approach is shown as dotted lines. The query profile is stored in a column based manner. For each computation of a minor diagonal, query profile values

for the respective cells needed for the computation are fetched and stored inside a score profile vector register.

```
Initialization;
Load gOpen to vector vGapOpen;
Load gExtend to vector vGapExtend;
For a = 1 to l1/k
      Initialize vector registers for 1 round (k rows);
      For b = 1 to l2+k-1
            Load the necessary vector registers for anti diagonal
                  computations;
            Fetch respective query profile scores;
            Calculate vector register of E vE;
            Calculate vector register of NE vNE;
            Calculate vector register of F vF;
            Calculate vector register of NF vNF;
            Calculate vector register of HA vH;
            Calculate vector register of NA vNA;
      End For
End For
Extract nid as NA (imax,jmax);
Return nid;
```

Figure 44. Pseudocode of the *nid* score calculation

Given are sequences S_i and S_j of lengths l_1 and l_2, respectively and vector registers *vH*, *vE*, *vF*, vN_A, vN_E and vN_F containing the values H_A, *E*, *F*, N_A, N_E and N_F, respectively. For each iteration c ($1 \leq c \leq (l_1+l_2-1)$), the values of $H_A(i,j)$, $E(i,j)$, $F(i,j)$, $N_A(i,j)$, $N_E(i,j)$ and $N_F(i,j)$ are computed for all $1 \leq i \leq l_1$ and $1 \leq j \leq l_2$. Calculations of the *vH*, *vE*, and *vF* vectors are done by utilizing the `spu_cmpgt` intrinsic, which compares each element of a vector with the corresponding element of another vector, to create vector masks. The masks are then used as patterns to generate the resulting vector using the `spu_sel` *intrinsic*, which selects the corresponding bit from either vector in accordance to a

provided pattern vector. The masks used in the *vE*, *vF* and *vH* computations are used to determine the value of the corresponding vN_E, vN_F and vN_A vectors, respectively. The nid score is extracted as $N_A(i_{max},j_{max})$, where (i_{max},j_{max}) denotes the coordinates of the maximum value in the corresponding matrix H_A.

Cell/BE does not support saturation arithmetic which are needed in the calculations to anticipate overflow problems. Hence, we utilized several spu intrinsics, i.e. `spu_sel`, `spu_splats`, `spu_rlmaska`, `spu_nor` and `spu_and` in conjunction with the existing `spu_add` and `spu_sub` to handle saturated additions and saturated subtractions, respectively.

7.3.3. MULTITHREADING-SPECIFIC IMPLEMENTATIONS

Our Cell/BE implementation utilizes the additional instructions of the PPE relating to control of the SPEs to implement the multi-threading. The PPE, which is capable of running a conventional operating system, has control over the SPEs and can start, stop, interrupt, and schedule processes running on the SPEs. Unlike SPEs, the PPE can read and write the main memory and the local memories of SPEs through the standard load/store instructions.

Given *k* SPEs, Figure 45 illustrates the mapping of our multi-thread implementation onto the Cell/BE.

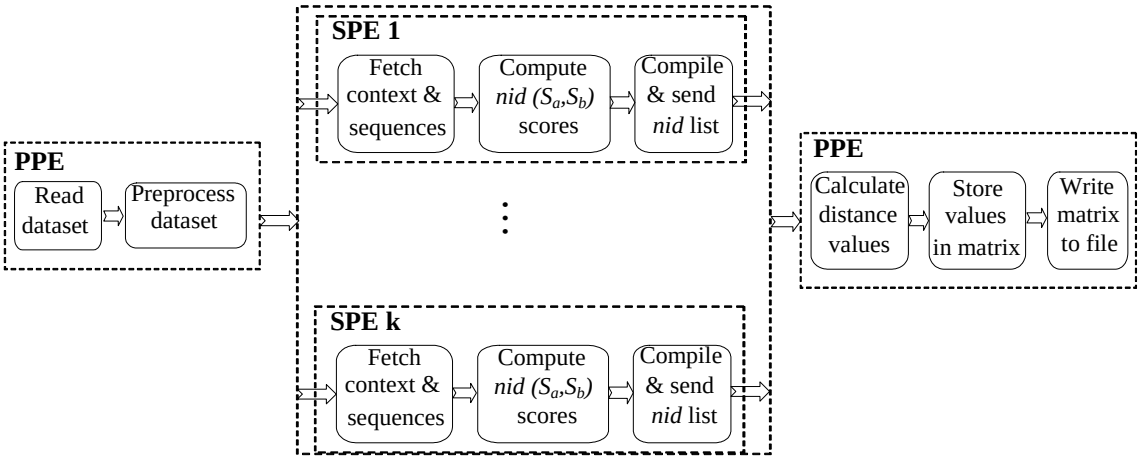

Figure 45. Mapping of pairwise distance matrix computation algorithm onto the Cell/BE

The PPE reads the input dataset, preprocesses it and divides the dataset into equal size blocks for each SPE to process. Since the blocks are independent of each other, no thread synchronization is necessary during the calculations. The mailbox functions `spe_in_mbox_write` and `spu_read_in_mbox` are used to ensure that all the SPEs obtain their respective contexts in their local memory. Using the context data, each SPE then transfers any required information and necessary sequences.

To improve transfer efficiency, the database sequences in main memory and in the local storage are aligned within the cache line and data structures are initialized during the transfer of the sequence. Once it has finished calculating all its respective *nid*(S_i,S_j) scores, each SPE sends the scores to the PPE in form of a list. The PPE compiles the lists and calculate the distance values and stores them in the distance matrix. The matrix is then outputted in a text file.

7.4. MAPPING TO THE X86/SSE2 ARCHITECTURE

Our SSE2 implementation uses *pthread[158]* to implement the multi-threading. The input dataset are preprocessed and sorted according to length. Each thread contains a copy of the database sequence, query sequence and its respective query profile. Since the dataset are sorted, the dataset into divided into roughly equal size workload for each thread to process. To avoid deadlock, *pthread_mutex_lock* and *pthread_mutex_unlock* operations are utilized.

SSE2 is a Single Instruction Multiple Data (SIMD) instruction set extension to the x86 architecture which allows the processors to operate on data in parallel. The SSE2 instructions support 8 bit elements in the vector registers. This allows 16 cells to be processed per vector register. Based on Equation 20, $nid(S_i,S_j)$ is computed without computation of the actual traceback. Since all elements in the same minor diagonal of the dynamic programming matrix can be computed independent of each other in parallel, the computation is done in minor diagonal order.

Given are sequences S_i and S_j of lengths l_1 and l_2, respectively and vector registers vH, vE, vF, vNA, vNE and vNF containing the values H_A, E, F, N_A, NE and NF, respectively. For each iteration c ($1 \leq c \leq (l_1+l_2-1)$), the values of $H_A(i,j)$, E(i,j), F(i,j), $N_A(i,j)$, NE(i,j) and NF(i,j), are computed for all $1 \leq i \leq l_1$ and $1 \leq j \leq l_2$.

Unlike Cell/BE, Intel's SSE2 instructions support saturation arithmetic. Hence, saturated subtractions and additions functions, *_mm_subs_epu8* and *_mm_adds_epu8*, respectively, are utilized to ensure that the values of the vector are within valid range

7.5. PERFORMANCE EVALUATION

In this section, we evaluate and compare our implementations. The first comparison is between our Cell/BE implementation and our x86/SSE2 implementation, a heterogeneous and homogeneous multi-core system, respectively. The second comparison is between our Cell/BE implementation and other accelerator technologies, i.e. FPGA and CUDA-enabled GPU.

7.5.1. PERFORMANCE ANALYSIS

A set of performance evaluation experiments has been conducted using different numbers of protein sequences i.e. 400 sequences of average length 408, 600 sequences of average length 462, 800 sequences of average length 454, and 1000 sequences of average length 446 as described in [93]. The experiments were carried out on a standalone PS®3 with Fedora Core 9.0 operating system and the Cell Software Development Kit (SDK) 3.1. The sequential ClustalW application, available online at http://www.bii.a-star.edu.sg/achievements/applications/clustalw/ , was benchmarked on an Intel Pentium 4 3.0 GHz processor with 1 GB RAM running on Windows XP.

Table 13 shows the performance analysis of our Cell/BE implementation using the above mentioned datasets. It compares the run times of our implementation and the baseline ClustalW on various processors. The performance analysis breaks down the speed up obtained by each phase of the improvement made by the implementation. The non-vectorized code is implemented according to the algorithm described in section 7.2, without the use of SIMD vectorization. The vectorized code is implemented according to

section 7.3.2. The term *n*(*m*) describes a dataset containing *n* sequences with an average length of *m*.

Table 13. Performance analysis of the parallel algorithm. The term *T* and *S* describes the runtime and speed up, respectively

#sequences (average length)	Processor	400 (408)		600 (462)		800 (454)		1000 (446)	
		T	S	T	S	T	S	T	S
Baseline ClustalW	Pentium 4 3.0 GHz	833.1	N.A	1697.0	N.A	2966.6	N.A	4409.6	N.A
Baseline ClustalW	PPE	667.86	1.24	1361.13	1.24	2379.0	1.24	3536.2	1.24
Non-vectorized code	PPE	357.89	1.87	717.83	1.89	1702.08	1.80	1871.08	1.89
Vectorized code	PPE+1SPE	57.15	6.26	113.41	6.33	168.54	7.83	237.12	7.89
Vectorized code	PPE+6SPEs	11.01	5.19	20.36	5.57	29.53	5.71	40.82	5.81

7.5.2. COMPARISON AGAINST X86/SSE2 ARCHITECTURE

A set of performance evaluation experiments has been conducted using six protein sequence datasets, which are divided into three representative datasets as shown in Table 14. Category A represents datasets of small number of long sequences, category B represents datasets of medium number of medium-length sequences and category C represents datasets of large number of short sequences. The datasets consist of sequences selected from the *Human Immunodeficiency Virus* (HIV) dataset downloaded from NCBI [159].

Table 14. Categories of input protein dataset

Dataset	Number of Sequences	Average Length	Category
1	400	856	A
2	1000	858	A
3	2000	266	B
4	4000	247	B
5	4000	83	C
6	8000	73	C

Our SSE2 implementation is benchmarked on IBM System x3650 with dual Xeon Quad Core E5430 2.66GHz and 6 GB RAM running CentOS 5.0 operating system. The Cell/BE experiments were carried out on a standalone PS®3 featuring a Cell/BE with frequency 3.2GHz and 256MB XDR Main RAM with Fedora Core 9.0 operating system and the Cell Software Development Kit (SDK) 3.1. The sequential ClustalW application, available online at http://www.bii.a-star.edu.sg/achievements/applications/clustalw/, was benchmarked on an Intel Pentium 4 3.0 GHz processor with 1 GB RAM running on Windows XP.

Figure 46 shows the speed-ups obtained by our SSE2 implementation up to 32 threads against our single-threaded vectorized version. Over the six datasets, our SSE2 implementation with 32 threads achieved an average speed-up of 6.6x over our single-threaded vectorized version.

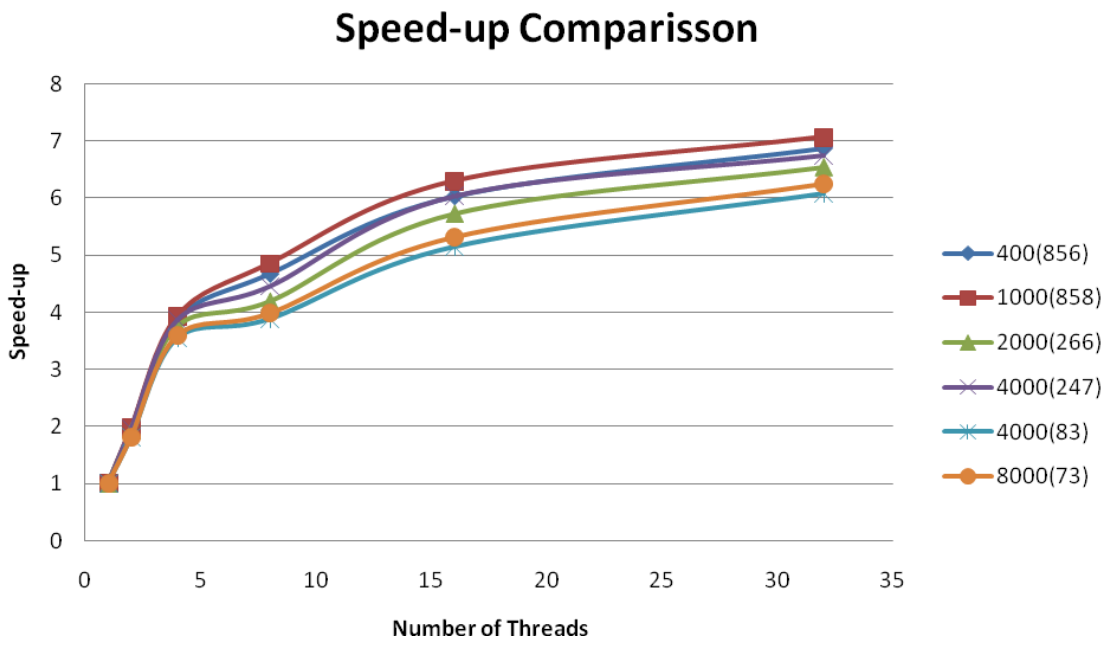

Figure 46. Speed-up of our x86/SSE2 implementation with up to 32 threads

Table 15. Performance evaluation results

Number of Sequences	Average Length	SSE2 implementation with 32 threads	Cell/BE implementation on the PS3®	ClustalW on P4 3.0 GHz
400	856	20.23	16.79	3114
1000	858	122.80	101.21	19670
2000	266	55.64	56.83	4386
4000	247	190.30	173.26	19424
4000	83	32.70	39.04	1595
8000	73	96.07	125.14	5165

Table 15 shows the performance evaluation of our implementations using the above mentioned datasets on different architectures, i.e. the SSE2 implementation with 32 threads, PS3® implementation with 6 SPEs and a baseline ClustalW on a P4. The term

n(*m*) describes that dataset contains *n* sequences with an average length of *m*. The speed-up of our SSE2 and Cell/BE implementations are benchmarked against the baseline ClustalW.

Throughout the benchmark, SSE2 implementation shows a comparable performance with the Cell/BE implementation. The Cell/BE implementation shows a better performance for datasets with fewer but longer sequences (category A), while the SSE2 implementation shows a better performance for datasets with more but shorter sequences (category C). This is due to the communication overhead for the PS3®, which involved DMA transfers of required data and sequences between the PPE and the SPEs. Over the six datasets, the SSE2 and Cell/BE implementations achieve an average of 99.6x and 108.5x speed-up over the phase one of the baseline ClustalW, respectively.

7.5.3. COMPARISON AGAINST OTHER ACCELERATOR TECHNOLOGIES

Our Cell/BE implementation was then compared to the FPGA and GPU implementations described in [87], in terms of speedups, programming productivity (in terms of implementation effort), cost efficiency, compute capability efficiency and power cost efficiency. Figure 47 shows the parallelization approaches utilized in each accelerator architecture.

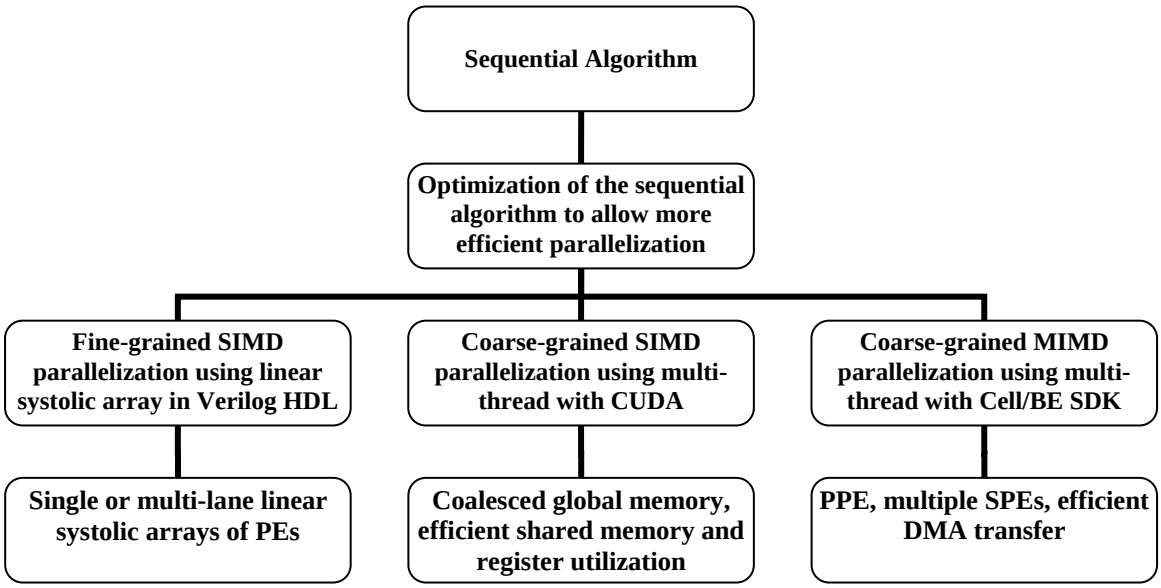

Figure 47. Utilized parallelization and optimization approach for each accelerator architecture

Our linear systolic array implementation is programmed using the Verilog HDL and targeted to the Xilinx XC5VLX330 FPGA device. The optimal performance can be obtained by fitting 16 linear systolic arrays of PEs with each array comprising 26 PEs and running at the maximum allowable frequency of 65MHz. The PE implementation has a 16-bit datapath and 12-bit *nid* path. A substitution table of size 32 × 32 with a resolution 16-bit is locally stored in each PE and the precision of the gap penalties is set to be 8-bit. To ease the access to sequences stored in the external RAM, each sequence is preprocessed and stored in one or more memory pages (1024 bytes for one page), depending on its length. When a new alignment starts, the alignment control logic reads in the memory pages occupied by the corresponding sequences from the external RAM. In this case, it takes a number of clock cycles to load in the sequences, but simplifies the system implementation.

Our CUDA implementation is benchmarked on a GeForce GTX 280 graphics card, with 30 streaming multiprocessors (SMs) comprising 240 scalar processors (SPs) and 1GB GDDR3 RAM, installed in a PC with an AMD Opteron 248 2.2 GHz processor running the Linux OS. The core frequency of the graphics card is 602MHz and the frequency of unified processors is 1296MHz. The performance evaluation of our Cell/BE implementation is carried out on a standalone PlayStation®3 featuring a Cell/BE with frequency 3.2GHz and 256MB XDR Main RAM running on the Linux OS.

The sequential runtime of pairwise distance computation in ClustalW (version 2.0.9) is profiled on a desktop computer with a P4 3.0GHz processor and 1GB RAM running the Linux OS. Because a FPGA development board equipped with the Xilinx XC5VLX330 FPGA device is not available as our resource, the runtime of the FPGA implementation is estimated through simulation. However, even though the runtime of the FPGA implementation is not so accurate, it still will not weaken the effect that gives readers a qualitative and intuitive performance comparison of these accelerators.

To remove the dependency on the input sequences used for the different tests, *cell updates per second* (CUPS) is a commonly used performance measure in bioinformatics. A CUPS represents the time for a complete computation of one cell in matrix H and N_H, including all memory operations and the corresponding computation of the values in the E, N_E, F and N_F matrices. Given a sequence dataset $S=\{S_1, S_2, \ldots, S_i, \ldots, S_n\}$, which consists of n sequences, the MCUPS (million cell updates per second) value of the pairwise distance computation for S is calculated by the following equation:

$$\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} l_i \times l_j}{t \times 10^6}$$

Equation 21. MCUPS calculation equation for pairwise distance matrix

where l_i and l_j denote the lengths of the sequences S_i and S_j respectively and t is the runtime in second.

Table 16. Runtime speedups of the three accelerators compared with the sequential implementation

Sequence Number	Average Length	P4	FPGA		GPU		Cell/BE	
		Time(s)	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
400	856	3114	2.59	1202.32	9.64	323.19	16.79	185.44
1000	858	19670	15.49	1269.85	58.54	336.01	101.21	194.35
2000	266	4386	8.68	505.30	31.39	139.71	56.83	77.17
4000	247	19424	31.46	617.42	99.33	195.55	173.26	112.10
4000	83	1595	11.29	141.28	26.25	60.75	39.04	40.85
8000	73	5165	39.42	131.02	68.61	75.28	125.14	41.27

Table 16 demonstrates the runtime speedups of the three accelerators compared with the sequential implementation. The FPGA implementation outperforms that of GPU and Cell/BE to a great extent for all the datasets, with a highest speedup of 1269.85, a lowest speedup of 131.02 and the average speedups of 1236.08, 561.36 and 136.15 for datasets of Category A, B and C respectively. The Cell/BE implementation demonstrates the lowest performance with a highest speedup of 106.49, a lowest speedup of 37.05 and the average speedups of 102.11, 70.12 and 38.51 for datasets of Category A, B and C respectively. The GPU implementation performs better than the Cell/BE implementation and worse than the FPGA implementation, with a highest speedup of 336.01, a lowest

speedup of 60.75 and the average speedups of 329.60, 167.63 and 68.02 datasets of Category A, B and C respectively. For each accelerator, its speedup degrades as the datasets change from Category A to Category C, which can be explained by the larger amount of computation when the average length of a dataset is longer.

Different physical characteristics of these accelerators determine the different programming models and languages. FPGA applications are mostly programmed using HDLs. In this chapter, Verilog HDL is used for the FPGA implementation. As a standard HDL, Verilog HDL is very similar in syntax to the C programming language and is easy to learn for designers with C programming experience and is easy to use for digital system implementation. It allows different levels of abstraction to be mixed in the same model: behavioral (or algorithmic) level, dataflow level, get level and switch level. Very efficient hardware implementation can be developed in Verilog HDL, but it requires a great deal of programming and implementation effort. CUDA is an extension of C/C++ which enables users to write scalable multi-threaded programs for CUDA-enabled GPUs[160]. CUDA programs can be executed on GPUs with NVIDIA's Tesla unified computing architecture[161]. Cell/BE accommodates different instruction set architectures (ISAs) for the PPE and SPEs. The PPE ISA is an extension of the PowerPC ISA and the extensions consist of the vector/SIMD multimedia extensions and C/C++ intrinsics for the vector/SIMD multimedia extensions. The SPE ISA is a new SIMD ISA, called the Synergistic Processor Unit Instruction Set Architecture, with accompanying C/C++ intrinsics. Most coding for the Cell/BE might be done by using a high-level languages such as C/C++, but to produce efficient, optimized code, an extra effort is required for software implementationers to understand and exploit the PPE and SPE

machine instructions. Generally, for complex applications or algorithms, software implementation using high-level languages on GPUs or Cell/BE is much easier than hardware implementation using HDLs on FPGAs. Even though high-level languages for FPGA implementation are being rapidly developed, the effort to implement and verify the FPGA implementation is still large. Hence, the implementation effort must be taken into account when making decisions about what kind of accelerator technologies is selected as a solution.

However, it is difficult to measure the accurate amount of implementation effort quantitatively for programming applications or algorithms. In this chapter, source lines of code (SLOC) is exploited as our implementation effort metric, which predicts the amount of effort required to develop a program by counting the number of effective lines in the program's source code in software engineering. Table 17 shows the SLOC and performance per line of code (LOC) of the different implementations written in Verilog HDL on FPGA, written in CUDA on GPUs and written on Cell/BE using the PowerPC instruction set for PPE and the synergistic processing units instruction set for SPEs. For the FPGA implementation, the SLOC of those modules generated by Xilinx CORE generator are not counted in.

As shown in Table 17, the FPGA implementation requires a lot more LOC than GPU and Cell/BE, and the LOC of the GPU implementation are approximately equivalent to those of the Cell/BE implementation. This suggests that GPU and Cell/BE require approximately equivalent implementation effort for a specific algorithm or application suitable for GPUs and Cell/BE, and that it require much more control logic and effort to directly implementation hardware in HDL languages on FPGAs than to program

equivalently functional software on GPUs or Cell/BE. After comparing the performance per LOC, we find that the GPU implementation shows the best performance per LOC for all the datasets. FPGA outperforms Cell/BE for datasets of Category A and B, but for dataset of Category C, Cell/BE does better than FPGA.

Table 17. SLOC and performance per LOC of the three accelerators

Accelerators	SLOC	MCUPS per LOC					
		400(856)	1000(858)	2000(266)	4000(247)	4000(83)	8000(73)
FPGA	4024	5.61	5.90	4.05	3.88	1.23	1.09
GPU	819	7.41	7.67	5.51	6.03	2.60	3.08
Cell/BE	987	3.53	3.68	2.52	2.85	1.43	1.38

These accelerators are all commonly available commercial hardware with different unit costs. When selecting accelerators, the unit cost coming along with the high performance must be taken into consideration. For the Xilinx XC5VLX330-1FFG1760C FPGA device, the unit price is US$8,382 from the Digi-Key Corporation (http://www.digikey.com/) and a PNY GeForce GTX 280 graphics card is available for about US$500 and a PlayStation®3 80GB system for about US$400 at Amazon (http://www.amazon.com/).

Table 18 gives the performance per dollar comparison of these accelerators. Even though the FPGA implementation gains the best performance but brings in very high unit cost, which results in its lowest performance per US$ compared to GPU and Cell/BE. With a medium unit cost and relatively higher performance, the GPU implementation shows the highest performance per US$, which is on average 4.47, 4.98 and 8.43 times better than the FPGA implementation and 2.58, 1.90 and 1.41x better than the Cell/BE implementation for datasets of Category A, B and C respectively.

Table 18. Performance per dollar of the three accelerators

Accelerators	Unit Price	MCUPS per US$					
		400(856)	1000(858)	2000(266)	4000(247)	4000(83)	8000(73)
FPGA	8,382	2.70	2.83	1.95	1.86	0.59	0.52
GPU	500	12.14	12.56	9.02	9.88	4.26	5.04
Cell/BE	400	8.84	9.21	6.30	7.13	3.57	3.45

To compare the compute capability utilizations of those accelerators, the theoretical peak performance of each accelerator is estimated for sequence alignments. For the implementation on FPGA, there are 16 linear systolic PE arrays with each array consisting of 26 PEs, running at the maximum allowable frequency of 65MHz. Because one cell can be computed in one clock cycle, if excluding the overhead incurred by sequence partition and sequence loading operations, the maximum compute capability can be estimated as 16 × 26 × 65 MCUPS = 27040 MCUPS. For the implementation on GPU and Cell/BE, due to the conditional branching instructions, the computing time of one cell is estimated by averaging the computing time of multiple cells. For the implementation on GPU, the average computing time of one cell is 35 clock cycles, measured using the clock () function in the kernel. Because there are 240 SPs on the graphics card and the frequency of unified processors is 1296 MHz, without considering the concurrency of all threads in a warp, the estimated theoretical compute capability should be at least 240 × 1296 / 35 MCUPS = 8886 MCUPS. For the implementation on the Cell/BE, the average computing time of one cell is measured using the SPU 32-bit decrementer functions, namely the `spu_read_decrementer` and `spu_write_decrementer`, yielding a result of 30 clock cycles. The PS3 consists of 6 SPEs, each with a 3.2 GHz clock, and the vector computation uses 16 bit values, which

means 8 cells are processed per vector register. Thus, the estimated theoretical compute capability is computed to be at least 6 × 8 × 3200 / 30 MCUPS = 5120 MCUPS. Table 19 demonstrates the compute capability utilization of the three accelerators.

Table 19. Compute capability utilizations of the three accelerators

Accelerators	Max. MCUPS	Compute Capability Utilization (%)					
		400(856)	1000(858)	2000(266)	4000(247)	4000(83)	8000(73)
FPGA	27040	83.54	87.83	60.32	57.69	18.28	16.24
GPU	≥ 8886	≤ 68.31	≤ 70.67	≤ 50.57	≤ 55.59	≤ 23.97	≤ 28.36
Cell/BE	≥ 5120	≤ 68.18	≤ 71.03	≤ 48.63	≤ 55.01	≤ 27.57	≤ 26.61

7.6. SUMMARY

We have presented a parallel algorithm on a homogeneous and a heterogeneous multi-core system for computation of distance matrix used in multiple sequence alignment algorithms. A performance analysis is done to break down the speed up obtained by each phase of the improvement. Three kinds of protein sequence datasets are used to evaluate the performance of our implementation. Our x86/SSE2 and Cell/BE implementations achieve an average of 99.6x and 108.5x speed-up over the phase one of the baseline ClustalW, respectively.

We also compare the performance of our Cell/BE implementation with other emerging accelerator technologies, i.e. FPGA and CUDA-enabled GPU. The comparison gives a comprehensive understanding of the advantages and disadvantages of these accelerators and also provides a reference for mapping algorithms or applications onto them. In addition to speedup, these accelerators are compared from a wide range of factors, including programming model and language, implementation effort, performance per dollar and compute capability utilization.

The experimental results show that the FPGA obtains the best performance in terms of runtime with speedups of up to four orders of magnitude compared to only three-orders of magnitude on GPU and Cell/BE. However, raw speedup is not the only aspect that determines the best accelerator choice for a developer. Our results show that the FPGA approach has a poor or medium programming productivity due to its large design effort requirement and poor cost efficiency due to its high unit cost. Furthermore, GPUs have medium performance, usually provide good programming productivity and good cost efficiency; while the Cell/BE has a slightly lower performance than a GPU, but provides medium programming productivity and medium cost efficiency. In addition, compute capability efficiency indicates the extent of which these accelerators exploit the available compute resources for a specific design or implementation. Our observations show that FPGAs are usually able to furthest exploit the compute capability of the device. GPUs and Cell/BE are also able to exploit the device compute capability to a great extent and show almost the same efficiency in our experiments. Finally, in consideration of energy and environmental factors, power cost efficiency indicates the tradeoff between performance and power dissipation. As shown in our results, FPGAs usually give the best power cost efficiency due to its highest performance and lowest power dissipation, compared with GPUs and Cell/BE; Cell/BE gives better power cost efficiency than GPUs, due to its much lower power dissipation compared with GPUs; and GPUs give the worst power cost efficiency due to its very high power dissipation.

8. CONCLUSION AND FUTURE WORK

This chapter concludes the report by discussing the conclusion and future works of the study.

8.1. CONCLUSION

Biological data available in genomic sequence databases are growing exponentially. This growth rate will continue since more sequencing projects will be finished in the near future. As data increases, so does the workload for managing, processing and analysing this data. Hence, due to this continuing improvements in high-throughput genomic sequencing and the ever-expanding sequence databases, bioinformatics to be rapidly moving towards a data-intensive, computational science. As a result, advances in computational power and methods for bioinformatics applications, such as genomic sequence analysis, are needed as well. Traditional approaches to sequence analysis techniques are expensive in terms of time and memory. High performance computing is a widely used method to improve performance. The emergence of accelerator technologies such as multi-core architecture has made it possible to achieve an excellent improvement in execution time for many bioinformatics applications, compared to current general-purpose platforms. Therefore, using multi-cores to solve sequence analysis problems is a promising and challenging research field, since large-scale computational bioinformatics problems can benefit much from this kind of processing power.

Multi-core architectures may take on a number of forms. One form is the heterogeneous multi-core architecture, which can address a variety of applications. Due to the characteristics of heterogeneous multi-core architectures, the application or algorithm

development process must be significantly changed in order to fully explore its potential. Hence, it brings a shift of paradigm in applications development since in order to implement efficient and scalable code for this type of architecture, novel programming techniques are required. New sequence analysis algorithms have to be presented in order to execute efficiently on multi-core architectures and new parallel communication patterns and partitioning scheme in parallel models are required. In this thesis, we have investigated various algorithms and techniques how to efficiently map bioinformatics applications onto heterogeneous multi-core systems.

Aligning long DNA sequences is a common and often repeated task in molecular biology. In this thesis, we have developed a novel, efficient and scalable parallel algorithm for very long DNA sequence alignment on a heterogeneous multi-core system, the Cell Broadband Engine. Our implementation utilizes two types of parallelization techniques: (i) SIMD vectorization within a processor and (ii) wavefront parallelization between processors. We also introduced a partitioning scheme to overcome the local storage limitation of the Synergistic Processor Elements (SPEs) as well as a direct SPE to SPE DMA transfer communication technique. Performance evaluation shows that our implementation shows almost linear speedup and leads to significant computational time savings.

Next, we have demonstrated how the PlayStation® 3, powered by the Cell Broadband Engine, can be used as a computational platform to accelerate the Smith-Waterman algorithm, an optimal pairwise sequence alignment. For large protein datasets, our implementation on the PlayStation® 3 provides a significant improvement in running

time compared to other implementations such as SSEARCH, Striped Smith-Waterman and CUDA-SW.

Furthermore, we have developed an novel implementation to accelerate a heuristic protein sequence database scanning algorithm, the BLASTP heuristic, on to a heterogeneous multi-core system, the Cell Broadband Engine. To our knowledge, this is the first ever reported parallelization of BLASTP on a heterogeneous multi-core system. We also introduced a new parallel communication pattern, in which the Power Processor Element (PPE) coordinates the data transfer. Furthermore, we utilized a data structure similar to compressed *deterministic finite-state automaton* (DFA) to fit the codeword lookup data in the SPEs. The BLASTP implementation on a Playstation®3 leads to significant runtime savings compared to corresponding sequential implementations.

Finally, we have developed an efficient parallel implementation that accelerates the distance matrix computation used in multiple sequence alignments on the x86 and Cell Broadband Engine architecture, a homogeneous and heterogeneous multi-core system, respectively. By taking advantage of multiple processors as well as SIMD vectorization, we are able to achieve speedups of two orders of magnitude compared to the publicly available implementation utilized in multiple sequence alignment algorithms. We have also compared the performance of our implementation on the Playstation®3 with other accelerator technologies, i.e. FPGA and GPU. In general, Cell/BE offers a lower performance compared to other accelerator architectures. However, it is able to exploit the device compute capability to a great extent and even show better efficiency for Category C dataset. Furthermore, it requires less implementation effort in terms of LOC and provides acceptable performance-to-cost ratio due to its low cost.

The speed-up of various bioinformatics implementations on the Cell/BE show that the Cell Broadband Engine Architecture is an attractive avenue for bioinformatics applications. It supports single and double precision floating point computation. Considering that the total power consumption of Cell/BE is less than half of a contemporary superscalar processor[106], Cell/BE can be considered as a promising power-efficient platform for future bioinformatics computing. Due to its low cost compared to other accelerator technologies, Cell/BE also provides a good performance-to-cost ratio.

The disadvantage of the Cell/BE Architecture is that it is a challenging environment for software development, i.e. it favors peak computational throughput over simplicity of program code. Another drawback is the 256 KB local store limitation of the SPE, requiring partitioning of input as well as data dependency checking mechanism in most applications. Data transfer through DMA also needs careful consideration as the data must be aligned equally in a 16byte grid and is restricted to 1,2,4,8, or n*16 bytes. Lastly, the capability of Cell/BE in performing double precision calculations as inferior compared to single precision. One way to handle this is to use iterative refinement, which means values are calculated in double precision only when necessary.

Overall, we conclude that the Cell/BE is an attractive and suitable platform for bioinformatics algorithms. However, in order the reach its optimal potential, the programmer needs to be able to work out a solution to overcome its limitations, e.g. local store and data transfer, which may result in the increase of complexity of the program code.

8.2. FUTURE WORK

Our future work includes three parts. The first part is to identify more bioinformatics applications that benefit much from the heterogeneous multi-core architecture. This will include Protein-Protein Interaction (PPI) prediction, which is a very important part in the field of bioinformatics and structural biology. The second one is to integrate our pairwise distance matrix computation algorithm with multiple sequence alignment tools. The third one is to apply new communication and dynamic load balancing techniques to our algorithms.

8.2.1. PROTEIN-PROTEIN INTERACTION PREDICTION USING PARALLEL GA WITH ISLAND MODEL ON THE CELL/BE ARCHITECTURE

In recent years, analysis of protein-protein interactions (PPIs) is an emerging issue to elucidate the mechanism of many biological processes, such as enzyme-substrate binding and immune response. Understanding protein-protein interactions is important in investigating intracellular signaling pathways and therefore is a very important aspect in the field of bioinformatics and structural biology.

Proteins that interact are more likely to co-evolve[162-165], therefore it is possible to make inferences about interactions between pairs of proteins based on their phylogenetic distances. It has also been observed in some cases that pairs of interacting proteins have fused orthologues in other organisms. In addition, a number of bound protein complexes have been structurally solved and can be used to identify the residues that mediate the interaction so that similar motifs can be located in other organisms.

Wang et. al.[166] proposes a novel hybrid Genetic Algorithm (GA)/Support Vector Machine (SVM) method that can predict the interactions between proteins intermediated by the protein-domain relations. A protein is represented by the domains contained inside, which can consider the effects of domain duplication. To simulate the combination of different domains, a transformation of the domain composition was taken subsequently. Lastly, a genetic algorithm was used to seek the optimized transformation, which had been adopted as the input vector of a predictor constructed using support vector machines method.

It would be interesting to see how such PPI prediction algorithm using a parallel Genetic Algorithm would be mapped on to the Playstation®3. The Playstation®3 with its 6 SPEs would benefit an island model approach of genetic algorithm, in which each SPEs is modeled as an island. Figure 48 illustrates this concept. The island model is scalable to continents, or even galaxies, with a multiple hierarchal clusters of the Playstation®3.

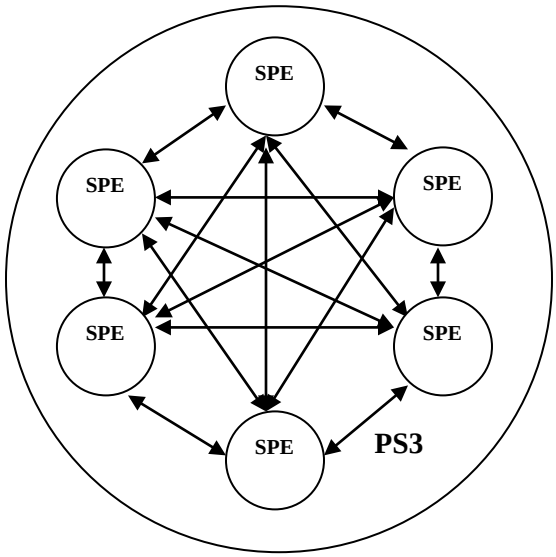

Figure 48. Parallel genetic algorithm with island model on the Playstation®3

The remaining work to develop such implementation is to present a communication and load balancing strategy between islands and between continents (if applies), as well as a parallel genetic algorithm for mapping onto the SPEs.

8.2.2. IMPLEMENTATION OF A SHORT READ ASSEMBLY ALGORITHM FOR DE NOVO GENOMIC SEQUENCING ON THE CELL/BE ARCHITECTURE

Determining the complete genome sequence of a species is an important application of bioinformatics. New sequencing technologies have emerged recently[167], for example, pyrosequencing (454 Sequencing) [168] and sequencing by synthesis (Solexa) [169]. Compared to the traditional Sanger[170] method, these technologies are capable of generating sequence data at a fraction of the cost and much quicker produce shorter reads, currently ~200 bp for pyrosequencing and 35 bp for Solexa[171].

A critical stage in genome sequencing is the assembly of shotgun reads, or piecing together fragments randomly extracted from the sample, to form a set of contiguous sequences (contigs) representing the DNA in the sample. Traditional methods for whole-genome shotgun fragment assembly rely on the overlap-layout-consensus approach [172], representing each read as a node and each detected overlap as an arc between the appropriate nodes. In sequencing projects that use Sanger technology, genomes are typically covered 6- to 10-fold. To assemble such data sets, the algorithms described above put great emphasis on the optimal exploitation of all reads. Issues like the correction of sequencing errors and the assembly of reads containing mismatches

increase the complexity of these algorithms. Due to their complexity, existing assemblers are incapable of assembling very large numbers of reads.

Therefore, very short reads are not well suited to the traditional approach. Because of their length, they must be produced in large quantities and at greater coverage depths than traditional Sanger sequencing projects.

Zerbino and Birney[173] developed a novel set of algorithms called *Velvet* to manipulate de Bruijn graphs for genomic sequence assembly. A de Bruijn graph is a compact representation based on short words (*k*-mers) that is ideal for high coverage, very short read (25–50 bp) data sets. *Velvet* represents a new approach to assembly that can leverage very short reads in combination with read pairs to produce useful assemblies.

Heterogeneous multi-core systems have been shown to be able to improve the performance of multiple application due to its characteristics. Therefore, it would be interesting to see how a short read assembly algorithm can be mapped on the a heterogeneous multi-core system to improve its performance. To our knowledge, an implementation of a short read assembly algorithm for de novo genomic sequencing on a heterogeneous multi-core system such as the Cell Broadband Engine would be the first ever implementation and therefore be a novel contribution for the scientific community.

8.2.3. OPEN PROGRAMMING LANGUAGE (OPENCL) ON THE CELL/BE

OpenCL is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors. It seeks to provide a framework for parallel programming on heterogeneous systems by using task-based and data-based parallelism.

The OpenCL 1.0 specification is made up of three main parts: the language specification, platform layer API and runtime API[174]. The language specification describes the syntax and programming interface for writing compute kernels that run on supported accelerators. The language used is based off of a subset of ISO C99. C was chosen as the basis for the first OpenCL compute kernel language due to its prevalence and familiarity in the developer community. To foster consistent results across different platforms, a well-defined IEEE 754 numerical accuracy is defined for all floating point operations along with a rich set of built-in functions. The developer has the option of pre-compiling their OpenCL compute kernel or letting the OpenCL runtime compile their kernels on demand.

The platform layer API gives the developer access to routines that query for the number and types of devices in the system. The developer can then select and initialize the necessary compute devices to properly run their work load. It is at this layer that compute contexts and work-queues for job submission and data transfer requests are created. Finally, the runtime API allows the developer to queue up compute kernels for execution and is responsible for managing the compute and memory resources in the OpenCL system.

Programming the Cell/BE is complicated both by the need to explicitly manage DMA data transfers for SPE computation, as well as the multiple layers of parallelism provided in the architecture, including heterogeneous cores, multiple SPE cores, multithreading, SIMD units, and multiple instruction issue. There is a significant amount of ongoing research in programming models and tools that attempts to make it easy to exploit the computation power of the Cell/BE architecture[175].

Currently, an OpenCL implementation for Cell/BE is not yet available. IBM has said it is "in the works", but there is no hint as to when it will be available. Therefore, an OpenCL framework on the Cell/BE, which allow any program written with the framework runnable on the Cell/BE with little to no modification of the source code, would be very helpful for programmers to be able to exploit the computation power of the Cell/BE architecture to the fullest.

8.2.4. THE FUTURE OF THE CELL BROADBAND ENGINE ARCHITECTURE

In 2008, IBM announced a revised variant of the Cell/BE called the PowerXCell 8i[176], which is available in QS22 Blade Servers from IBM. The PowerXCell 8i is manufactured on a 65 nm process, and adds support for up to 32 GB of slotted DDR2 memory. It is similar to the Cell/BE, in which it consists of eight Synergistic Processor Elements (SPEs) and one PowerPC® Processor Element (PPE). The PowerXCell 8i also improves one of the drawbacks of the Cell/BE by dramatically improving the double-precision floating-point performance on the SPEs from a peak of about 12.8 GFLOPS to 102.4 GFLOPS total for eight SPEs.

The IBM Roadrunner supercomputer[177, 178], currently the world's second fastest[79], consists of 12,240 PowerXCell 8i processors, along with 6,562 dual-core AMD Opteron processors. Beside the QS22 and RoadRunner computers, the PowerXCell processor is also available as an accelerator on a PCI Express card and is used as the core processor in the QPACE project[179].

The PowerXCell 32iv chip, which was marked to be the next extension of the PowerXCell 8i, however, had its development halted by IBM. According to IBM, the

design of the processor did not provide expected performance and Cell/BE would reappear in another form. The PowerXCell 32i was projected to feature four PowerPC processor elements (PPE) as well as 32 synergistic processing elements (SPEs).

Another unfortunate development from the PS3 front is also disconcerting. The latest PS3 Firmware system software update 3.21 released on April 2010[180] disables the *Install Other OS* feature that was available on the PS3 systems due to security concerns. This feature allowed PS3 users to install other operating systems such as Linux on the PS3 and use it as an entry-level personal computer which can be used as a complete development environment for the Cell/BE. Most of the experiments of our implementations are based on such setting. PS3 users currently using the *Other OS* feature can choose not to upgrade their systems. However, doing so would banned them from accessing the PlayStation Network and other gaming and entertainment contents. While this will have little impact on the PS3 as a supercomputer, it may be the end of the PS3 as a low-cost development environment for the Cell/BE.

Although the future of the Cell/BE development looks bleak at the moment, the design, concept and algorithm elaborated in this thesis on various bioinformatics applications would still be relevant for other next-generation accelerator technologies, such as Larrabee [181].

Larrabee is a General Purpose GPU (GPGPU) chip that Intel is developing from its current line of integrated graphics accelerators. It is planned to be released in 2010 and is expected to be a platform for research and development in computer graphics and HPC. Larrabee's design of using many small, simple cores is similar to the multi-core concepts behind the Cell/BE. Further similarities in the use of a high-bandwidth ring bus to

communicate between cores also indicate that communication design and methods introduced by our Cell/BE implementations are applicable to next generation hardware such as Larrabee.

Below are some significant differences in implementation which should make programming Larrabee simpler.

- The Cell/BE is a heterogeneous multi core processor, which consists of one PPE and several SPE processors. Additionally, the PPE can run an OS. In contrast, Larrabee's cores are homogeneous, and it is not expected to run an OS.
- Each SPE has a local store, for which explicit DMA operations are used for all accesses to DRAM. Ordinary reads/writes to DRAM are not allowed. In Larrabee, all on-chip and off-chip memories are under automatically-managed coherent cache hierarchy, so that its cores virtually share a uniform memory space through standard copy (MOV) instructions. Larrabee cores each have 256K of local L2 cache, and an access which hits another L2 segment takes longer to access
- Because of the cache coherency noted above, each program running in Larrabee has virtually a large linear memory just as in traditional general-purpose CPU; whereas an application for Cell/BE should be programmed taking into consideration limited memory footprint of the local store associated with each SPE but with theoretically higher bandwidth. However, since local L2 is faster to access, an advantage can still be gained from using Cell/BE-style programming methods.
- Cell/BE uses DMA for data transfer to/from on-chip local memories, which enables explicit maintenance of overlays stored in local memory to bring memory

closer to the core and reduce access latencies, but requiring additional effort to maintain coherency with main memory; whereas Larrabee uses a coherent cache with special instructions for cache manipulation, which mitigate miss and eviction penalties and reduce cache pollution (e.g. for rendering pipelines and other stream-like computation) at the cost of additional traffic and overhead to maintain cache coherency.

- Each SPE in the Cell/BE runs only one thread at a time, in-order. A core in Larrabee runs up to four threads, but only one at a time. Larrabee's hyperthreading helps hide the latencies inherent to in-order execution.

REFERENCES

1. Benson, D.A., et al., *GenBank.* Nucl. Acids Res., 2008. **36**(suppl_1): p. D25-30.
2. Kulikova, T., et al., *EMBL Nucleotide Sequence Database in 2006.* Nucl. Acids Res., 2007. **35**(suppl_1): p. D16-20.
3. Sugawara, H., et al., *DDBJ working on evaluation and classification of bacterial genes in INSDC.* Nucl. Acids Res., 2007. **35**(suppl_1): p. D13-15.
4. International Nucleotide Sequence Database Collaboration. *http://www.insdc.org/*. 2009 [cited; Available from: http://www.insdc.org/.
5. National Center for Biotechnology Information. *http://www.ncbi.nlm.nih.gov/sites/entrez?db=taxonomy*. 2009 [cited; Available from: http://www.ncbi.nlm.nih.gov/sites/entrez?db=taxonomy.
6. National Center for Biotechnology Information. *http://www.ncbi.nlm.nih.gov/collab/FT/index.html*. 2009 [cited; Available from: http://www.ncbi.nlm.nih.gov/collab/FT/index.html.
7. Apweiler, R., et al., *The Universal Protein resource (UniProt) 2009.* Nucleic Acids Research, 2009. **37**(SUPPL. 1): p. D169-D174.
8. Apweiler, R., et al., *UniProt: The universal protein knowledgebase.* Nucleic Acids Research, 2004. **32**(DATABASE ISS.): p. D115-D119.
9. Margulies, M., et al., *Genome sequencing in microfabricated high-density picolitre reactors.* Nature, 2005. **437**(7057): p. 376-380.
10. Porreca, G.J., et al., *Multiplex amplification of large sets of human exons.* Nat Meth, 2007. **4**(11): p. 931-936.
11. Bentley, D.R., *Whole-genome re-sequencing.* Current Opinion in Genetics & Development, 2006. **16**(6): p. 545-552.
12. Crescenzi, P., et al., *On the complexity of protein folding.* Journal of Computational Biology, 1998. **5**(3): p. 423-465.
13. Watson, J. and F. Crick, *A structure of deoxyribonucleic acid.* Nature, 1953. **171**: p. 737-738.
14. Setubal, J. and J. Meidanis, *Introduction to computational molecular biology*. 1997: PWS Publishing Company.

15. Bader, D.A., *Computational biology and high-performance computing*. Communications of the ACM, 2004. **47**(11): p. 34-40.
16. Darling, A., L. Carey, and W. Feng. *The Design, Implementation, and Evaluation of mpiBLAST*. in *4th International Conference on Linux Clusters: The HPC Revolution 2003 in conjunction with ClusterWorld Conference & Expo*. 2003.
17. Kofune, Y., T. Koita, and A. Fukuda. *Performance evaluation of MPI-HMMER on the OBIGrid*. in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'04*. 2004.
18. Li, K.-B., *ClustalW-MPI: ClustalW analysis using distributed and parallel computing*. Bioinformatics, 2003. **19**(12): p. 1585-1586.
19. Kim, H.C., et al. *Massive multiple sequence alignment of 16S bacterial ribosomal RNAs using ClustalW-Message Passing Interface (MPI) based on beowulf linux system*. in *2005 IEEE Computational Systems Bioinformatics Conference, Workshops and Poster Abstracts*. 2005.
20. Stamatakis, A.P., T. Ludwig, and H. Meier. *A fast program for maximum likelihood-based inference of large phylogenetic trees*. in *Proceedings of the ACM Symposium on Applied Computing*. 2004.
21. Pande, V. *Folding@Home: Using Worldwide distributed computing to break fundamental barriers in molecular simulation*. in *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*. 2006.
22. Mullikin, J.C. and Z. Ning, *The phusion assembler*. Genome research, 2003. **13**(1): p. 81-90.
23. Liu, W., et al. *GPU-ClustalW: using graphics hardware to accelerate multiple sequence alignment*. in *High Performance Computing - HiPC 2006. 13th International Conference. Proceedings, 18-21 Dec. 2006*. 2006. Bangalore, India: Springer-Verlag.
24. Oliver, T., et al., *Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW*. Bioinformatics, 2005. **21**(16): p. 3431-3432.
25. Gelsinger, P.P., et al., *Microprocessors circa 2000*. IEEE Spectrum, 1989. **26**(10): p. 43-7.

26. Intel, *A new era of architectural innovation arrives with Intel dual-core processors*, in *Technology@Intel Magazine*. 2005. p. 1-11.

27. Advanced Micro Devices, *Multi-core processors-the next evolution in computing*, in *Website: http://multicore.amd.com/WhitePapers/Multi-Core Processors WhitePaper.pdf*. 2005, White Paper.

28. Sun Microsystems, *Introduction to throughput computing*. 2003, White Paper.

29. Kalla, R., B. Sinharoy, and J.M. Tendler, *IBM Power5 chip: a dual-core multithreaded processor.* IEEE Micro, 2004. **24**(2): p. 40-7.

30. Mirman, I. *Dual- and quad-core systems dominate today*. 2009 [cited; Available from: http://www.cilk.com/multicore-blog/bid/8097/Don-t-get-caught-with-your-multicore-pants-down.

31. Lin, G., et al., *Computational assignment of protein backbone NMR peaks by efficient bounding and filtering.* Journal of bioinformatics and computational biology, 2003. **1**(2): p. 387-409.

32. Konc, J. and D. Janezic, *A branch and bound algorithm for matching protein structures*, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2007. p. 399-406.

33. Smith, T. and M. Waterman, *Identification of common molecular subsequences.* J Mol Biol, 1981. **147**(1): p. 195-197.

34. Needleman, S.B. and C.D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal of Molecular Biology, 1970. **48**(3): p. 443-453.

35. Nussinov, R. and I. Tinoco Jr, *Sequential folding of a messenger RNA molecule.* Journal of Molecular Biology, 1981. **151**(3): p. 519-533.

36. Zuker, M. and P. Stiegler, *Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information.* Nucleic Acids Research, 1981. **9**(1): p. 133-148.

37. Wang, J., K.B. Li, and W.K. Sung, *G-PRIMER: Greedy algorithm for selecting minimal primer set.* Bioinformatics, 2004. **20**(15): p. 2473-2475.

38. Blekas, K., D.I. Fotiadis, and A. Likas, *Greedy mixture learning for multiple motif discovery in biological sequences.* Bioinformatics, 2003. **19**(5): p. 607-617.

39. Stoye, J., V. Moulton, and A.W.M. Dress, *DCA: An efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment.* Computer Applications in the Biosciences, 1997. **13**(6): p. 625-626.

40. Sammeth, M., B. Morgenstern, and J. Stoye, *Divide-and-conquer multiple alignment with segment-based constraints.* Bioinformatics, 2003. **19**(SUPPL. 2).

41. Baldi, P. and S. Brunak, *Bioinformatics : the machine learning approach.* Adaptive computation and machine learning. 2001, Cambridge, Mass.: MIT Press.

42. Pugalenthi, G., et al., *Identification of structurally conserved residues of proteins in absence of structural homologs using neural network ensemble.* Bioinformatics, 2009. **25**(2): p. 204-210.

43. Hwang, T., et al., *Robust and efficient identification of biomarkers by classifying features on graphs.* Bioinformatics, 2008. **24**(18): p. 2023-2029.

44. Pavlidis, P., I. Wapinski, and W.S. Noble, *Support vector machine classification on the web.* Bioinformatics, 2004. **20**(4): p. 586-587.

45. Altschul, S.F., et al., *Basic local alignment search tool.* Journal of Molecular Biology, 1990. **215**(3): p. 403-410.

46. Lipman, D.J., S.F. Altschul, and J.D. Kececioglu, *A tool for multiple sequence alignment.* Proceedings of the National Academy of Sciences of the United States of America, 1989. **86**(12): p. 4412-4415.

47. Notredame, C., D.G. Higgins, and J. Heringa, *T-coffee: A novel method for fast and accurate multiple sequence alignment.* Journal of Molecular Biology, 2000. **302**(1): p. 205-217.

48. Wallace, I.M., et al., *M-Coffee: Combining multiple sequence alignment methods with T-Coffee.* Nucleic Acids Research, 2006. **34**(6): p. 1692-1699.

49. Durbin, R., et. al, *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* . 1998: Cambridge University Press, Cambridge.

50. Hancock, J.M. and M.J. Zvelebil, *Dictionary of Bioinformatics and Computational Biology*. 2004: Wiley-Liss.

51. Waterman, M.S. and M. Eggert, *A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons.* Journal of Molecular Biology, 1987. **197**(4): p. 723-728.

52. Henikoff, S. and J.G. Henikoff, *Amino acid substitution matrices from protein blocks.* Proceedings of the National Academy of Sciences of the United States of America, 1992. **89**(22): p. 10915-10919.

53. Sander, C. and R. Schneider, *The HSSP database of protein structure-sequence alignments.* Nucleic Acids Research, 1994. **22**(17): p. 3597-3599.

54. Carrillo, H. and D.J. Lipman, *The multiple sequence alignment problem in biology.* SIAM J. Appl. Math., 1988. **48**(5): p. 1073-1082.

55. Hogeweg, P. and B. Hesper, *The alignment of sets of sequences and the construction of phyletic trees: An integrated method.* Journal of Molecular Evolution, 1984. **20**(2): p. 175-186.

56. Feng, D.F. and R.F. Doolittle, *Progressive sequence alignment as a prerequisitetto correct phylogenetic trees.* Journal of Molecular Evolution, 1987. **25**(4): p. 351-360.

57. Altschul, S.F., R.J. Carroll, and D.J. Lipman, *Weights for data related by a tree.* Journal of Molecular Biology, 1989. **207**(4): p. 647-653.

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

59. Heringa, J., *Two strategies for sequence comparison: Profile-preprocessed and secondary structure-induced multiple alignment.* Computers and Chemistry, 1999. **23**(3-4): p. 341-364.

60. Larkin, M.A., et al., *Clustal W and Clustal X version 2.0.* Bioinformatics, 2007. **23**(21): p. 2947-2948.

61. Thompson, J.D., et al., *The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools.* Nucl. Acids Res., 1997. **25**(24): p. 4876-4882.

62. Edgar, R.C., *MUSCLE: Multiple sequence alignment with high accuracy and high throughput.* Nucleic Acids Research, 2004. **32**(5): p. 1792-1797.

63. Simossis, V.A. and J. Heringa, *PRALINE: A multiple sequence alignment toolbox that integrates homology-extended and secondary structure information.* Nucleic Acids Research, 2005. **33**(SUPPL. 2): p. W289-W294.

64. Dayhoff, M.O., R.M. Schwartz, and B.C. Orcutt, *A model of evolutionary change in proteins.* Atlas of protein sequence and structure, Nat. Biomed. Res, 1978. **5**(suppl 3): p. 345-351.

65. Gotoh, O., *An improved algorithm for matching biological sequences.* J Mol Biol, 1982. **162**(3): p. 705-708.

66. Korf, I., M. Yandell, and J. Bedell, *BLAST*. 2003: O'Reilly \& Associates, Inc.

67. Gish, W. *http://blast.wustl.edu*. 1996-2003 [cited; Available from: http://blast.wustl.edu.

68. Lopez, R., et al., *WU-Blast2 server at the European Bioinformatics Institute.* Nucleic Acids Research, 2003. **31**(13): p. 3795-3798.

69. Altschul, S.F., et al., *Gapped BLAST and PSI-BLAST: A new generation of protein database search programs.* Nucleic Acids Research, 1997. **25**(17): p. 3389-3402.

70. Darling, A., L. Carey, and W. Feng, *The Design, Implementation, and Evaluation of mpiBLAST*. ClusterWorld, 2003.

71. Yang, C.T., T.F. Han, and H.C. Kan. *G-BLAST: A grid-based solution for mpiBLAST on computational grids*. in *IEEE Region 10 Annual International Conference, Proceedings/TENCON*. 2007.

72. Cameron, M., H.E. Williams, and A. Cannane, *Improved gapped alignment in BLAST*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2004. **1**(3): p. 116-129.

73. Cameron, M., H.E. Williams, and A. Cannane, *A deterministic finite automaton for faster protein hit detection in BLAST.* Journal of Computational Biology, 2006. **13**(4): p. 965-978.

74. Grama, A., et al., *Introduction to Parallel Computing, 2nd Edition*. 2003: Addison Wesley.

75. Aspray, W., *The stored program concept.* IEEE Spectrum, 1990. **27**(9): p. 51.

76. Quinn, M.J., *Parallel Programming in C with MPI and OpenMP*. 2004: McGraw-Hill.

77. Flynn, M.J., *Some Computer Organizations and Their Effectiveness.* IEEE Transactions on Computers, 1972. **C-21**(9): p. 948-960.

78. Duncan, R., *Survey of parallel computer architectures.* Computer, 1990. **23**(2): p. 5-16.

79. Top500. *Top 500 supercomputer sites*. 2010 [cited; Available from: www.top500.org.

80. White, C.T., et al. *BioSCAN. A VLSI-based system for biosequence analysis*. in *IEEE International Conference on Computer Design - VLSI in Computers and Processors*. 1991.

81. Chow, E.T., et al., *A systolic array processor for biological information signal processing*, in *Proceedings of the 5th international conference on Supercomputing*. 1991, ACM: Cologne, West Germany.

82. Guerdoux-Jamet, P. and D. Lavenier, *SAMBA: Hardware accelerator for biological sequence comparison.* Computer Applications in the Biosciences, 1997. **13**(6): p. 609-615.

83. Oliver, T., L.Y. Yeow, and B. Schmidt, *Integrating FPGA acceleration into HMMer.* Parallel Computing, 2008. **34**(11): p. 681-691.

84. Derrien, S. and P. Quinton, *Hardware Acceleration of HMMER on FPGAs.* Journal of Signal Processing Systems, 2008.

85. Aung, Y., et al., *C-Based Design Methodology for FPGA Implementation of ClustalW MSA*, in *Pattern Recognition in Bioinformatics*. 2007. p. 11-18.

86. Oliver, T.F., B. Schmidt, and D.L. Maskell, *Reconfigurable architectures for bio-sequence database scanning on FPGAs.* IEEE Transactions on Circuits and Systems II: Express Briefs, 2005. **52**(12): p. 851-855.

87. Liu, Y., et al., *Comparison of Accelerator Architectures for*

Large-Scale Biological Sequence Alignment. IEEE Trans. Parallel Distrib. Syst., 2009. **Under review**.

88. Boukerche, A., et al., *An FPGA-Based Accelerator for Multiple Biological Sequence Alignment with DIALIGN*, in *High Performance Computing – HiPC 2007*. 2007. p. 71-82.
89. Owens, J.D., et al., *A Survey of General-Purpose Computation on Graphics Hardware.* Computer Graphics Forum, 2007. **26**(1): p. 80-113.
90. Buck, I., et al. *Brook for GPUs: Stream computing on graphics hardware*. in *ACM Transactions on Graphics*. 2004.
91. McCool, M.D., Z. Qin, and T.S. Popa. *Shader metaprogramming*. in *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*. 2002.
92. Manavski, S.A. and G. Valle, *CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment.* BMC Bioinformatics, 2008. **9**.
93. Liu, W., et al., *Streaming Algorithms for Biological Sequence Alignment on GPUs.* IEEE Transactions on Parallel and Distributed Systems., 2007.
94. Chen, C., et al., *GPU-MEME: Using graphics hardware to accelerate motif finding in DNA sequences*, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2008. p. 448-459.
95. Schatz, M.C., et al., *High-throughput sequence alignment using Graphics Processing Units.* BMC Bioinformatics, 2007. **8**.
96. Zheng, H., et al. *Cone beam reconstruction speedup using trigonometric relevancy and GPU technology*. in *2nd International Conference on Bioinformatics and Biomedical Engineering, iCBBE 2008*. 2008.
97. Liu, Y., D. Maskell, and B. Schmidt, *CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units.* BMC Research Notes, 2009. **2**(1): p. 73.
98. Nickolls, J., et al., *Scalable Parallel Programming with CUDA.* ACM Queue, 2008. **6**(2): p. 40 - 53.

99. Justin, H., *AMD CTM overview*, in *ACM SIGGRAPH 2007 courses*. 2007, ACM: San Diego, California.

100. Lindholm, E., et al., *NVIDIA Tesla: A Unified Graphics and Computing Architecture.* IEEE Micro, 2008. **28**(2): p. 39 - 55.

101. Wirawan, A., et al., *CBESW: Sequence alignment on the playstation 3.* BMC Bioinformatics, 2008. **9**: p. 377.

102. Wirawan, A., C.K. Kwoh, and B. Schmidt, *Parallel DNA Sequence Alignment on the Cell Broadband Engine.* Lecture Notes on Computer Science, 2008. **4967**: p. 1249-1256.

103. Wirawan, A., C.K. Kwoh, and B. Schmidt, *Pairwise Distance Matrix Computation for Multiple Sequence Alignment on the Cell Broadband Engine.* Lecture Notes on Computer Science, 2009. **5544**: p. 954-963.

104. Wirawan, A., et al., *High Performance Protein Sequence Database Scanning on the Cell B.E. Processor.* Scientific Programming, 2009. **17**(1-2): p. 97-111.

105. Nim, T.H., et al. *Applications of heterogeneous structure of cell broadband engine architecture for biological database similarity search*. in *2nd International Conference on Bioinformatics and Biomedical Engineering, iCBBE 2008*. 2008.

106. Sachdeva, V., et al. *Exploring the viability of the Cell Broadband Engine for bioinformatics applications*. in *IEEE International Parallel and Distributed Processing Symposium*. 2007. Long Beach, CA, USA: IEEE.

107. Stamatakis, A., et al., *Exploring new search algorithms and hardware for phylogenetics: RAxML meets the IBM cell.* Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, 2007. **48**(3): p. 271-286.

108. Intel. *Intel® Core™ i7 Processor Extreme Edition and Intel® Core™ i7 Processor Datasheet*. 2009 [cited; Available from: http://download.intel.com/design/processor/datashts/320834.pdf.

109. Kahle, J.A., et al., *Introduction to the Cell multiprocessor.* IBM Journal of Research and Development, 2005. **49**(4-5): p. 589-604.

110. Kumar, R., et al., *Heterogeneous chip multiprocessors.* Computer, 2005. **38**(11): p. 32-38.

111. Buttari, A., J. Dongarra, and J. Kurzak, *Limitations of the PlayStation 3 for High Performance Cluster Computing*. 2007, Innovative Computing Laboratory, University of Tennessee Knoxville.

112. Pham, D., et al. *The design methodology and implementation of a first-generation CELL processor: a multi-core SoC*. in *Proceedings of the IEEE 2005 Custom Integrated Circuits Conference*. 2005. San Jose, CA, USA: IEEE.

113. Bader, D.A., et al., *High performance combinatorial algorithm design on the Cell Broadband Engine processor.* Parallel Computing, 2007. **33**(10-11): p. 720-740.

114. Hofstee, H.P. *Power efficient processor architecture and the cell processor*. in *Proceedings. 11th International Symposium on High-Performance Computer Architecture, 12-16 Feb. 2005*. 2005. San Francisco, CA, USA: IEEE (Comput. Soc.).

115. Balakrishnan, S., et al. *The impact of performance asymmetry in emerging multicore architectures*. in *Proceedings - International Symposium on Computer Architecture*. 2005.

116. Kumar, R., et al. *Single-ISA heterogeneous multi-core architectures for multithreaded workload performance*. in *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*. 2004.

117. Kumar, R., D.M. Tullsen, and N.P. Jouppi. *Core architecture optimization for heterogeneous chip multiprocessors*. in *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*. 2006.

118. Mueller, S.M., et al. *The vector floating-point unit in a synergistic processor element of a CELL processor*. in *Proceedings. 17th IEEE Symposium on Computer Arithmetic, 27-29 June 2005*. 2005. Cape Cod, MA, USA: IEEE Computer Society.

119. Chen, T., et al., *Cell broadband engine architecture and its first implementation.*, in *IBM developerWorks.* 2005.

120. Kistler, M., M. Perrone, and F. Petrini, *Cell multiprocessor communication network: Built for speed.* IEEE Micro, 2006. **26**(3): p. 10-23.

121. Hofstee, P. and M. Day. *Hardware and software architectures for the CELL processor*. in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2005. Jersey City, USA.

122. International Business Machines, *Cell Broadband Engine Programming Tutorial v1.1.*, in *IBM developerWorks.* 2006.

123. Buttari, A., et al., *A Rough Guide to Scientific Computing On the PlayStation 3.* 2007, Innovative Computing Laboratory, University of Tennessee Knoxville.

124. Blagojevic, F., et al. *RAxML-cell: parallel phylogenetic tree inference on the cell broadband engine*. in *2007 IEEE International Parallel and Distributed Processing Symposium, 26-30 March 2007*. 2007. Long Beach, CA, USA: IEEE.

125. Asahara, A., et al. *Cell-broadband-engine-based realtime wavelet decomposition for HDTV video images and beyond*. in *2006 IEEE International Conference on Multimedia and Expo, 9-12 July 2006*. 2006. Toronto, Ont., Canada: IEEE.

126. Benthin, C., et al. *Ray tracing on the cell processor*. in *IEEE Symposium on Interactive Ray Tracing 2006, 18-20 Sept. 2006*. 2006. Salt Lake City, UT, USA: IEEE.

127. Hjelte, N., *Smoothed Particle Hydrodynamics on the Cell Broadband Engine.* 2006, Umeå University, Department of Computer Science.

128. Kim, J. and J. JaJa, *Streaming model based volume ray casting implementation for Cell Broadband Engine.* Scientific Programming, 2009. **17**(1-2): p. 173-184.

129. Felsenstein, J., *Evolutionary Trees from DNA Sequences: A Maximum-Likelihood Approach.* 1981: United States. p. 27p.

130. Wald, I., S. Boulos, and P. Shirley, *Ray tracing deformable scenes using dynamic bounding volume hierarchies.* ACM Transactions on Graphics, 2007. **26**(1): p. 18 pp.

131. Monaghan, J.J., *Smoothed particle hydrodynamics.* Reports on Progress in Physics, 2005. **68**(8): p. 1703-59.

132. Hadap, S. and N. Magnenat-Thalmann. *Modeling dynamic hair as a continuum*. in *European Association for Computer Graphics. 22nd Annual Conference. EUROGRAPHICS 2001, 4-7 Sept. 2001 Computer Graphics Forum*. 2001. Manchester, UK: Blackwell Publishers for Eurographics Assoc.

133. Di Blas, A., et al., *The UCSC Kestrel parallel processor.* IEEE Transactions on Parallel and Distributed Systems, 2005. **16**(1): p. 80-92.

134. Wozniak, A., *Using video-oriented instructions to speed up sequence comparison.* Comput. Appl. Biosci., 1997. **13**(2): p. 145-150.

135. International Business Machines, *Cell Broadband Engine Programming Tutorial v1.1.*, in *IBM developerWorks.* 2005.

136. Benson, D.A., et al., *GenBank.* Nucleic Acids Research, 2000. **28**(1): p. 15-18.

137. Farrar, M., *Striped Smith-Waterman speeds database searches six times over other SIMD implementations.* Bioinformatics, 2007. **23**(2): p. 156-161.

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

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

140. Liu, Y., B. Schmidt, and D. Maskell, *CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions.* BMC Research Notes, 2010. **3**(1): p. 93.

141. IBM, *C/C++ Language Extensions for Cell Broadband Engine Architecture v.2.5*. 2008: IBM developerWorks.

142. Pearson, W.R., *Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms.* Genomics, 1991. **11**(3): p. 635-650.

143. Pearson, W.R., *Rapid and sensitive sequence comparison with FASTP and FASTA.* Methods in Enzymology, 1990. **183**: p. 63-98.

144. Brenner, S.E., C. Chothia, and T.J.P. Hubbard, *Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships.* Proceedings of the National Academy of Sciences of the United States of America, 1998. **95**(11): p. 6073-6078.

145. Kent, W.J., *BLAT - The BLAST-like alignment tool.* Genome research, 2002. **12**(4): p. 656-664.

146. Li, M., et al., *PatternHunter II: Highly sensitive and fast homology search.* Journal of bioinformatics and computational biology, 2004. **2**(3): p. 417-439.

147. Oehmen, C. and J. Nieplocha, *ScalaBLAST: A scalable implementation of BLAST for high-performance data-intensive bioinformatics analysis.* IEEE Transactions on Parallel and Distributed Systems, 2006. **17**(8): p. 740-749.

148. Jacob, A., et al. *FPGA-accelerated seed generation in Mercury BLASTP*. in *Proceedings 2007 IEEE Symposium on Field-Programme Custom Computing Machines, FCCM 2007*. 2007.

149. Jacob, A., et al., *Mercury BLASTP: Accelerating Protein Sequence Alignment.* ACM Trans. Reconfigurable Technol. Syst., 2008. **1**(2): p. 1-44.

150. Katoh, K., et al., *MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform.* Nucleic Acids Research, 2002. **30**(14): p. 3059-3066.

151. Schmollinger, M., et al., *DIALIGN P: Fast pair-wise and multiple sequence alignment using parallel processors.* BMC Bioinformatics, 2004. **5**.

152. Catalyurek, U., et al. *Improving Performance of Multiple Sequence Alignment Analysis in Multi-client Environments*. in *Proceedings of the First International Workshop on High Performance Computational Biology (HiCOMB 2002, IPDPS 2002)*. 2002.

153. Catalyurek, U., et al. *A component-based implementation of multiple sequence alignment*. in *Proceedings of the ACM Symposium on Applied Computing*. 2003.

154. Chaichoompu, K., S. Kittitornkun, and S. Tongsima. *MT-ClustalW: Multithreading multiple sequence alignment*. in *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*. 2006.

155. Luo, J., et al. *Parallel multiple sequence alignment with dynamic scheduling*. in *International Conference on Information Technology: Coding and Computing, ITCC*. 2005.

156. Oliver, T., et al. *Multiple sequence alignment on an FPGA*. in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*. 2005.

157. Saitou, N. and M. Nei, *The neighbor-joining method: a new method for reconstructing phylogenetic trees.* Molecular biology and evolution, 1987. **4**(4): p. 406-425.

158. Lewis, B. and D.J. Berg, Multithreaded Programming with Pthreads, 1998.

159. NCBI. *NCBI Homepage*. [cited; Available from: http://www.ncbi.nlm.nih.gov/.

160. Lindholm, E., et al., *NVIDIA Tesla: A unified graphics and computing architecture.* IEEE Micro, 2008. **28**(2): p. 39-55.

161. Ucb/Eecs, et al., *The landscape of parallel computing research: a view from Berkeley*. 2006.

162. Dandekar, T., et al., *Conservation of gene order: A fingerprint of proteins that physically interact.* Trends in Biochemical Sciences, 1998. **23**(9): p. 324-328.

163. Enright, A.J., et al., *Protein interaction maps for complete genomes based on gene fusion events.* Nature, 1999. **402**(6757): p. 86-90.

164. Marcotte, E.M., et al., *Detecting protein function and protein-protein interactions from genome sequences.* Science, 1999. **285**(5428): p. 751-753.

165. Pazos, F. and A. Valencia, *Similarity of phylogenetic trees as indicator of protein-protein interaction.* Protein Engineering, 2001. **14**(9): p. 609-614.

166. Wang, B., et al. *Prediction of protein interactions by combining genetic algorithm with SVM method*. in *2007 IEEE Congress on Evolutionary Computation, CEC 2007*. 2008.

167. Metzker, M.L., *Emerging technologies in DNA sequencing.* Genome research, 2005. **15**(12): p. 1767-1776.

168. Margulies, M., et al., *Genome sequencing in microfabricated high-density picolitre reactors.* Nature, 2005. **437**(7057): p. 376-380.

169. Bentley, D.R., *Whole-genome re-sequencing.* Current Opinion in Genetics and Development, 2006. **16**(6): p. 545-552.

170. Sanger, F., S. Nicklen, and A.R. Coulson, *DNA sequencing with chain-terminating inhibitors.* Proceedings of the National Academy of Sciences of the United States of America, 1977. **74**(12): p. 5463-5467.

171. Dohm, J.C., et al., *SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing.* Genome research, 2007. **17**(11): p. 1697-1706.

172. Batzoglou, S., *Algorithmic challenges in mammalian genome sequence assembly.* Encyclopedia of genomics, proteomics and bioinformatics, 2005(PART 4).

173. Zerbino, D.R. and E. Birney, *Velvet: Algorithms for de novo short read assembly using de Bruijn graphs.* Genome research, 2008. **18**(5): p. 821-829.

174. Khronos Group. *OpenCL - The open standard for parallel programming of heterogeneous systems*. 2009 [cited; Available from: http://www.khronos.org/opencl/.

175. O'Brien, K., et al., *Supporting OpenMP on cell*, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2008. p. 65-76.

176. IBM. *PowerXCell 8i*. 2008 [cited; Available from: https://www-01.ibm.com/chips/techlib/techlib.nsf/products/PowerXCell_8i.

177. IBM, *Fact Sheet & Background: Roadrunner Smashes the Petaflop Barrier.* 2008.

178. IBM. *IBM Roadrunner*. 2009 [cited; Available from: http://en.wikipedia.org/wiki/IBM_Roadrunner.

179. Wikipedia. *IBM Roadrunner*. 2009 [cited; Available from: http://en.wikipedia.org/wiki/IBM_Roadrunner.

180. Sony. *Support: System Software Updates*. 2010 [cited; Available from: http://us.playstation.com/support/systemupdates/ps3/index.htm.

181. Larry, S., et al., *Larrabee: a many-core x86 architecture for visual computing.* ACM Trans. Graph., 2008. **27**(3): p. 1-15.