Sebastián Isaza Ramírez

Multicore Architectures for Bioinformatics Applications

Multicore Architectures for Bioinformatics Applications

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op maandag 3 oktober 2011 om 10:00 uur

door

Sebastián ISAZA RAMÍREZ

Master of Science in Embedded Systems Design
University of Lugano, Switzerland
geboren te Medellín, Colombia

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. H.J. Sips

Copromotor:
Dr. ir. G.N. Gaydadjiev

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter	Technische Universiteit Delft
Prof. dr. ir. H.J. Sips, promotor	Technische Universiteit Delft
Dr. ir. G.N. Gaydadjiev, copromotor	Technische Universiteit Delft
Prof. dr. ir. M.J.T. Reinders	Technische Universiteit Delft
Prof. dr. P.J. French	Technische Universiteit Delft
Prof. dr. H. Blume	Leibniz Universitat Hannover, Duitsland
Prof. dr. N.J. Dimopoulos	University of Victoria, Canada
Dr. A. Ramirez	Technical University of Catalonia, Spanje
Prof. dr. K.G. Langendoen	Technische Universiteit Delft, reservelid

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Sebastián Isaza Ramírez

Multicore Architectures for Bioinformatics Applications
Delft: TU Delft, Faculty of Elektrotechniek, Wiskunde en Informatica - III
Thesis Technische Universiteit Delft. – With ref. –

Met samenvatting in het Nederlands.

ISBN 978-90-72298-24-9

Subject headings: multicores, bioinformatics, sequence alignment, performance analysis.

Cover design: Carolina Ochoa

Printed in The Netherlands

Multicore Architectures for Bioinformatics Applications

Sebastián Isaza Ramírez

Abstract

In this dissertation, we address the challenges of performance scaling for bioinformatics applications on multicore architectures. In particular, we focus on sequence alignment, one of the fundamental tasks in bioinformatics. Due to the exponential growth of biological databases and the computational complexity of the algorithms used, high performance computing systems are required. Recently, computer architecture has shifted towards the multicore paradigm in an attempt to sustain the performance scalability that single-core processors did provide in the past. Although multicore architectures have the potential of exploiting the task-level and data-level parallelism found in bioinformatics workloads, efficiently harnessing systems with hundreds of cores requires deep understanding of the applications and the architecture specifics. This thesis presents a study of two sequence alignment applications that are modeled, characterized, mapped and optimized targeting two multicore architectures. More precisely, we use the Cell BE and the SARC architectures, the latter developed within the project this thesis was part of. The targeted applications, i.e., HMMER and ClustalW, are used for pairwise alignment and multiple sequence alignment. We first propose an analytical model to predict the performance of applications parallelized under the master-worker scheme. We use HMMER and the Cell BE processor for our experimental case study and for validation of our model. Results show the high accuracy of the model and the scaling behavior of the application phases. Next we investigate the optimal mapping of ClustalW on the Cell BE, identify a number of limitations in the architecture and propose few instruction-set extensions to accelerate the main ClustalW kernel. Last, we study ClustalW and HMMER scalability on the SARC architecture with up to one thousand cores. We also investigate the impact of different input types on ClustalW performance.

Acknowledgements

I remember well the few moments I spent with Stamatis. They were fun and surprising. I met him in April 2006 when I was applying for a PhD position in his lab and talked to him just a few times in October, the same year, before he got ill. I thank him for inviting me to join his Computer Engineering Laboratory and for leaving us SARC, a large and interesting European project to work on.

I want to thank Georgi, my PhD supervisor. I deeply appreciate his trust and constant support for all my plans, be them part of my PhD or not, since the beginning to this day. Despite of clearly being different, we always understood each other and most of the time agreed on the fundamental. I have to say that I particularly enjoy his black humor, especially during difficult times. Apart from the many technical and scientific things, determination and pragmatism at work are among the things I learned from him.

My PhD work was done in close collaboration with the UPC and BSC in Barcelona. I would like to thank our partners Friman Sanchez, Felipe Cabarcas, Alex Ramirez, Mateo Valero and the rest of the team in Barcelona. Special thanks to Friman, for his comments on the first draft of this thesis and for the many and extensive discussions about our work and not only.

I thank the members of my PhD committee for accepting our invitation. I want to thank Prof. Nikitas Dimopoulos for a thorough examination of my thesis and many useful comments. I thank my promotor Prof. Henk Sips for promptly helping us in the last phase of my PhD under a tight schedule. And I thank Dr. Zaid Al-Ars for supporting the last months of my PhD through the COMCAS project.

I would like to mention two persons I appreciate and have been my mentors in the past. Prof. Jose Aedo was the first to introduce me to computer engineering research during my time as an undergraduate student in Colombia. He has been helpful and supportive until today. Prof. Laura Pozzi was my master thesis advisor in Switzerland. She was helpful and I learned many things from her that were especially valuable when starting my PhD.

I thank Catalin and Bogdan, my office mates, for being always helpful, friendly and open to any questions I would ask. I thank my colleagues from the CE Lab for the technical discussions and for fun chats in the corridors. In particular Chunyang, Kamana, Marius, Arnaldo, Vlad, Laiq, Said, Demid, Lui and Ghazaleh. I thank the staff from our Lab: Lidwina, Bert, Monique, Erik and Eef have been very helpful in all kinds of administrative and computer issues.

I want to thank the group of great friends that were closest to me during my time in Delft: Daniele, Carlo, Christos, Dimitris, Lotfi, Yiannis and Niki. Our endless and circling discussions about recurrent topics are very fun, still today. Thanks for being so special to Manu and Aleja. I thank Carlo for proofreading my thesis in the last minute, Rob for translating the propositions and Ernst for translating the abstract. With Ernst I had an enjoyable time when we worked together during his thesis.

I want to thank a few special Colombian friends for the many dinners, parties and trips together. Jero and Jorge for the fun we had in Lugano and the friendship that remained. Milo, Ana, Patrol and Caro for being always around, making me happy. Very special thanks to Caro for putting so much time and patience on designing this thesis cover I'm very happy with. In Delft, Aleja, Manu and me also met and enjoyed the company of very special friends: Angela, Wiki, Sandra, Oswaldo, Cami and Israel.

My parents taught me freedom and love, nothing more I could wish for. I have missed them very much as well as my brothers and I look forward to seeing them much more often.

Aleja and Manu are the girls that make my everyday a happy day, a happy life. I owe them so much.

Sebastián Delft, The Netherlands, October 2011.

Table of Contents

Abstract . . . . . **i**

Acknowledgments . . . . . **iii**

List of Tables . . . . . **ix**

List of Figures . . . . . **xi**

List of Acronyms . . . . . **xv**

1 Introduction **1**
- 1.1 Thesis Contributions 4
- 1.2 Thesis Organization 5

2 Bioinformatics Theory and Applications **7**
- 2.1 Molecular Biology 7
- 2.2 Bioinformatics 8
- 2.3 Biological Databases 11
- 2.4 Sequence Alignment 12
  - 2.4.1 Needleman-Wunsch and Smith-Waterman 17
  - 2.4.2 FASTA 18
  - 2.4.3 BLAST 19
  - 2.4.4 HMMER 20
- 2.5 Multiple Sequence Alignment 22
  - 2.5.1 ClustalW 25
- 2.6 Summary 29

3 Architectures, Simulators and Related Work **31**
- 3.1 Multicore Architectures 31
  - 3.1.1 The Cell Broadband Engine 32
  - 3.1.2 The SARC Architecture 35

3.2 Simulators . . . 37
3.2.1 CellSim . . . 37
3.2.2 TaskSim . . . 38
3.3 Related Work . . . 39
3.3.1 ClustalW Implementations . . . 39
3.3.2 HMMER Implementations . . . 43
3.4 Summary . . . 46

4 Performance Estimation Model . . . 47
4.1 HMMER Analytical Model . . . 48
4.1.1 HMMERCELL Program Phases . . . 48
4.1.2 Model Derivation . . . 49
4.1.3 Model Parametrization . . . 51
4.1.4 Maximum Effective SPE Count . . . 52
4.1.5 Model Validation . . . 53
4.2 Experimental Methodology . . . 53
4.3 Cell BE Profiling Results . . . 55
4.3.1 The PPE Buffering Phase . . . 56
4.3.2 The SPE Viterbi Phase . . . 57
4.3.3 The PPE Traceback Phase . . . 58
4.4 Discussion . . . 60
4.5 Summary . . . 61

5 Cell BE Performance Optimizations . . . 63
5.1 ClustalW and its Implementation on the Cell BE . . . 64
5.1.1 ClustalW Analysis . . . 64
5.1.2 Cell BE Implementation . . . 65
5.2 Experimental Setup . . . 68
5.3 Results . . . 69
5.3.1 Pairwise Alignment . . . 69
5.3.2 Progressive Alignment . . . 71
5.3.3 Overall application's performance . . . 75
5.3.4 Limitations . . . 78
5.4 Enhancing the SPE Instruction-set . . . 79
5.4.1 Proposed Instructions . . . 79
5.4.2 Simulation Setup . . . 80
5.4.3 Acceleration Results . . . 81
5.5 Summary . . . 82

6 Performance Scalability on Manycores 85
6.1 Application's Description and Parallelism 86
6.1.1 ClustalW 86
6.1.2 HMMER 89
6.2 Experimental Methodology 91
6.2.1 Workers 92
6.3 The Influence of the Input Data Sets 93
6.3.1 ClustalW 93
6.3.2 HMMER 95
6.4 Simulation Results 95
6.4.1 ClustalW 96
6.4.2 HMMER 104
6.5 Summary 107

7 Conclusions and Future Work 111
7.1 Summary and Conclusions 111
7.2 Open Issues and Future Directions 115

Bibliography 117

List of Publications 127

Samenvatting 129

About the Author 131

List of Tables

2.1 ClustalW profiling for various input data sets. 28

3.1 Performance comparison of various SW and HW implementations of Smith-Waterman. 43

4.1 Maximum effectively usable workers. 53
4.2 Validation results. 54

5.1 PA phase names used. 72
5.2 Functionality of the proposed SPE instructions 80

6.1 Characterization of various ClustalW input sets. 87
6.2 HMMER program phases. 91
6.3 List of parameter names and value ranges. 92

List of Figures

2.1 The transfer of genetic information. 8
2.2 The Tree of Life. 10
2.3 Exponential growth of biological databases. 11
2.4 Two possible alignments of a pair of sequences. 12
2.5 The dotplot alignment method. 14
2.6 The Levenshtein distance between two sequences. 14
2.7 The BLOSUM62 substitution matrix for amino acids. 15
2.8 Global and local alignments. 16
2.9 HMMER Input/Output block diagram (*hmmsearch*). 22
2.10 HMMER (*hmmsearch*) processing pipeline in versions 2 and 3. 23
2.11 ClustalW Input/Output block diagram. 25
2.12 Multiple sequence alignment produced with ClustalX. 26
2.13 Pairwise alignment output similarity table. 27
2.14 Guide (phylogenetic) tree. 27

3.1 The Cell BE processor block diagram. 32
3.2 Synergistic Processing Element (SPE) block diagram. 33
3.3 The SARC architecture. 36
3.4 The TaskSim simulation scenario. 39
3.5 HMMERCELL program phases. 45

4.1 Relationship of dependence between functions. 51
4.2 Distribution of protein database element sizes. 55

4.3 HMMERCELL execution time overview. 56
4.4 Scaling characteristics of the PPE buffering phase (M_BUF) parametrized for the HMM size and the sequence size in the left and right figures respectively. 57
4.5 Scaling characteristics of the SPE Viterbi phase (W_VIT) parametrized for the HMM size and the sequence size in the left and right figures respectively. 57
4.6 Scaling characteristics of the PPE Viterbi Traceback phase (M_PP) parametrized for the HMM size and the sequence size in the left and right figures respectively. 58
4.7 M_PP analysis: total number of tracebacks, parametrized for the HMM size and the sequence size in the left and right figures respectively. 59
4.8 M_PP analysis: average time per individual traceback, parametrized for the HMM size and the sequence size in the left and right figures respectively. 59

5.1 PA workflow. a.) Baseline sequential version. b.) Parallel version showing the PPE and one of the SPEs. 67
5.2 Generation of the *prfscore matrix* by dot-products of all rows in the two input *profiles*. 68
5.3 ClustalW performance for different platforms and optimizations. 69
5.4 ClustalW speedup using multiple SPEs. 70
5.5 Execution time distribution for *prfscore* phases in the SPEs. . . 73
5.6 Scalability of *prfscore* processing on the SPEs. 73
5.7 Execution time distribution of PA phases in the SPEs. 74
5.8 Time growth of PA phases in the SPEs. 75
5.9 Execution time distribution of PA phases in the PPE. 76
5.10 Time growth of PA phases in the PPE. 76
5.11 Execution time of ClustalW phases with varying number of SPEs. 77
5.12 Execution time of ClustalW phases with varying number of input sequences. 77

5.13 Speedup and dynamic instruction count reduction of the proposed instructions compared to the original SPE 81

6.1 Parallel HMMER diagram. 90
6.2 Sequence length histogram for different input test sets. 94
6.3 Entry length histogram of protein databases. 95
6.4 Sensitivity to memory latency. Other parameters are: 32 MICs, No cache, INF Rings. 96
6.5 Sensitivity to memory bandwidth. Other parameters are: No cache, INF Rings. 97
6.6 Sensitivity to cache size. Other parameters are: 4 cache banks, 1 MIC, 1 DRAM, INF Rings. 98
6.7 Sensitivity to cache banks. Other parameters are: 1MB cache, 1 MIC, 1 DRAM, INF Rings. 99
6.8 Sensitivity to GDB bandwidth. Other parameters are: No cache, 32 MICs. 100
6.9 Sensitivity to SQ latency. Other parameters are: No cache, 32 MICs, INF rings. 101
6.10 Load balancing with 200 sequences as input, with 1024 general-purpose workers. The vertical axis represents the cores and the horizontal axis represents time. Dark (blue) segments are computations and light ones (gray and green) show idleness. 102
6.11 Performance scalability with and without task sorting using general-purpose cores. 200 sequences from BioPerf are used as input set. 102
6.12 Performance comparison for up to 1024 workers of two types: general-purpose cores ($1\times$) and application-specific accelerators ($100\times$). The vertical axis is in Log scale and the baseline is a single general-purpose core. 103
6.13 HMMERCELL execution time overview. 104
6.14 Speedup relative to the BASE version. 105
6.15 HMMER performance scalability with a single worker core as baseline. 105
6.16 Time share of HMMER program phases for a short HMM. . . 106

List of Acronyms

ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
BLAST	Basic Local Alignment Search Tool
CellBE	Cell Broadband Engine
CGRA	Coarse Grain Reconfigurable Array
CMOS	Complementary Metal-oxide-semiconductor
CMP	Chip Multiprocessor
CUPS	Cell Updates Per Second
DB	Database
DDR	Double Data Rate
DLP	Data-level Parallelism
DMA	Direct Memory Access
DNA	Deoxyribonucleic acid
DRAM	Dynamic Random Access Memory
EIB	Element Interconnect Bus
EMBL	European Molecular Biology Laboratory
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GDB	Global Data Bus
GPU	Graphics Processing Unit
GNU	GNU's Not Unix
GT	Guide Tree
HMM	Hidden Markov Model
ILP	Instruction-level Parallelism
ISA	Instruction-set Architecture
LDB	Local Data Bus
LS	Local Store
MFC	Memory Flow Controller
MIC	Memory Interface Controller
MPI	Message Passing Interface
MSA	Multiple Sequence Alignment
PA	Progressive Alignment
PPE	PowerPC Processing Element
PPU	PowerPC Processing Unit

PW	Pairwise Alignment
RAM	Random Access Memory
RNA	Ribonucleic acid
SARC	Scalable Computer Architecture
SDK	Software Development Kit
SIMD	Single-instruction Multiple-data
SMT	Simultaneous Multi-Threading
SPE	Synergistic Processing Element
SPEC	Standard Performance Evaluation Corporation
SPM	Scratchpad Memory
SPU	Synergistic Processing Unit
SQ	Synchronization Queue
TLP	Thread-level Parallelism

1 Introduction

The discovery of the DNA boosted the developments in Molecular Biology, which, in turn, gave birth to a completely new scientific discipline in the seventies of the last century. Today, this discipline is known as Bioinformatics and it is concerned with a special class of computational algorithms that solves various molecular biology problems. With the time, bioinformatics became an essential driver of the progress in medical sciences. Nowadays, not only new medications designers use the bioinformatics tools. Very frequently, discoveries in the other medical fields, e.g., cancer research, are achieved thanks to the available computational methods.

Following the advancements in lab equipment used to extract, for instance, the nucleotide pattern of a DNA sample, the last three decades have seen a proliferation of bioinformatics databases types and their exponential growth in sizes. Some of the most widely used databases today, such as the GenBank, UniProtKB, Pfam, and SCOP, contain up to hundreds of millions of nucleotide sequences, amino acid sequences, protein family models, protein structures, etc. The fast growth of these databases, combined with the algorithms computational complexity have enforced the use of high performance computing equipment for Molecular Biology research. The type and scale of data produced by modern medical laboratory research can be efficiently used only if the computer systems used by scientists allow them to interpret the data in reasonable time. Many algorithms and software tools have been proposed for analyzing the large and continuously growing amounts of biological data. One of the most fundamental bioinformatics algorithms class is Sequence Alignment. Sequence Alignment algorithms are used to find regions of similarity between DNA or protein sequences, which is a pivotal problem in the bioinformatics field. Finding an optimal alignment is normally a very computationally expensive task because typical sequences have thousands of elements and create an enormous number of possible solutions. In some scenarios, such as in multiple

sequence alignment, finding an optimal solution has been shown as intractable and only suboptimal heuristic approaches are used in practice. In other less demanding applications such as database search with short queries, exact algorithms, such as the well-known Smith-Waterman are applied. Therefore, the main goal of this thesis is to propose solutions capable of improving the computing technology to better suit the computational requirements of sequence alignment workloads.

A recent trend in computer architecture has been the introduction of a variety of multicore processors[1]. Up to the beginning of the last decade, the performance improvement in microprocessors was achieved mostly through the steadily increase in clock frequency, due to the CMOS technology miniaturization. Every new uniprocessor generation was directly faster than its predecessor because of the increasing clock frequencies. However, the challenges in distributing the clock signal in designs with increasing complexity, and the physical limitations on power dissipation forced a paradigm change. Today, most general-purpose and many embedded processors are implementing multiple cores on the same silicon die. Having several independently operating processors on a single chip has brought back many challenges and opportunities known from the parallel computing research efforts in the eighties. Although hardware designers can now easily place several cores of various types on a single chip, issues such as the best parallel memory hierarchy and the most convenient programming model can not be avoided any longer.

One question this thesis addresses is: *how much faster can a sequence alignment application become on a thousand core processor?* Although the exact answer depends on the application and data set characteristics and could only be truly measured after fabricating such a chip, today we can look into current multicores performance and use analytical and simulation techniques to formulate design considerations for future scalable computer architectures able to deliver the performance required by the developments in bioinformatics. It should be noted that although the peak performance of multicores grows exponentially as the biological databases, now that performance improvements are achieved by adding more cores instead of increasing the clock frequency, the actual performance growth is slowing down. Furthermore, in absolute terms, current processors are often too weak for many bioinformatics applications.

In the beginning, computers were mostly used for scientific applications. Then, the personal computer was introduced in the eighties and soon Internet was born. Since then, the main driving forces behind processor design have been

[1] A single-chip system consisting of two or more independent processors (cores).

commercial applications, such as multimedia, communications and networking. Luckily, today the computing research community has also recognized the importance and societal impact of improving the efficiency of bioinformatics applications. In fact, we could fairly state that the computational demands of bioinformatics are higher compared to many other domains. The potential contributions to medicine alone can be considered as a strong reason for doing bioinformatics related research.

As a sign of a growing interest in the field, several companies (e.g., Progenic, Convey Computer and Impulse Accelerated Technologies) have appeared in recent years offering high performance computing solutions for bioinformatics, mostly using specialized hardware implemented on FPGAs. Although the advertised performance benefits advertised are impressive and their products are good solutions for large companies or research centers, smaller institutions cannot afford them. In this respect, future generations of manycore processors incorporating domain-specific cores, could provide significant computational power for the bioinformatics field at affordable costs. In such general-purpose architectures, a portion of the cores can be tailored to specific bioinformatics kernels in order to increase their performance and efficiency. Moreover, the economy of scale of general-purpose processors can keep prices affordable for more users.

The work presented in this thesis has been performed in the context of the Scalable Computer Architecture (SARC) project [10] and in close collaboration with the Technical University of Catalonia and the Barcelona Supercomputing Center. SARC was an integrated project sponsored by the European Commission through the Sixth Framework Programme and was successfully completed in 2010. The SARC project main goal was fundamental research in computer architecture aimed at finding solutions for the scalable design of future multicore processors. Although SARC covered a wide range of topics from programming models and runtime systems to on- and off-chip interconnects, this thesis is concerned with domain-specific accelerators and their contribution to performance scalability when integrated in the rest of the system. Three application domains were defined to demonstrate the SARC advantages: bioinformatics, multimedia and scientific computing. In this thesis we only deal with bioinformatics acceleration.

The objectives of this thesis are the following:

- to gain deep insight in the behavior and the computational requirements of bioinformatics applications, with sequence alignment as the main target and capture it in a precise analytical model;

- to investigate and understand the bottlenecks when sequence alignment applications are parallelized on modern multicore architectures;
- to propose architectural improvements, estimate their performance and predict upcoming bottlenecks for future manycore systems.

1.1 Thesis Contributions

The main contributions of this thesis can be summarized as follows:

- An analytical model of the sequence analysis program HMMER, capturing its performance behavior for multicore architectures and the characterization of the main program phases. Based on theoretical expectations, code inspection and profiling, we propose a highly accurate mathematical model that is capable of predicting HMMER's execution time. The proposed model assumes a master-worker parallelization scheme and can be used for runtime scheduling. Moreover, the model also contributes to understanding the available parallelism and the application bottlenecks.
- The parallelization and code optimization of the multiple sequence alignment program ClustalW, running on the Cell BE processor. We port ClustalW to the Cell BE architecture and explore a number of optimizations, both at the fine-grain level inside the kernels and at the thread-level. We improve the performance, the scalability and analyze the limitations found with respect to the Cell BE. We also propose instruction-set extension aimed at increasing performance.
- The characterization of various input data set types and their impact on performance when exploiting parallelism. We show that different sets of sequences have a significant different impact on ClustalW parallelism, and, therefore, on the maximal achievable performance.
- The performance scalability comparison of a manycore architecture based on general-purpose processors against one based on application-specific accelerators. In addition, we propose load-balancing optimizations for ClustalW on such architecture.
- The sensitivity study of various architecture parameters to ClustalW performance in a simulated manycore system. We separately investigate parameters in the cache system, the interconnect, the memory controllers

and the synchronization queue and measure their impact on performance scalability.

1.2 Thesis Organization

This thesis is organized as follows:

Chapter 2 compiles a number of background concepts in bioinformatics. Some of them are necessary to understand the experiments and analysis in the following chapters. Other concepts are complementary knowledge that contribute to a broader view of the problems addressed. In this chapter we also describe the two applications used in the experiments, namely ClustalW and HMMER.

Chapter 3 presents the two multicore architectures studied in this thesis: Cell BE and SARC. The second part of the chapter describes the simulators used in our instruction-set and manycore experiments: CellSim and TaskSim. Finally, we discuss what other authors have reported with respect to the parallelization and performance analysis of relevant bioinformatics applications.

Chapter 4 presents an analytical model for the HMMER bioinformatics application. The model is able to predict the execution time of a HMMER master-worker parallelization. As an example, the Cell BE architecture is used to validate the model. Furthermore, the chapter presents profiling results of each of the program phases in order to reveal their scaling behavior. The proposed model helps us understand the parallelism and the various bottlenecks of the applications.

Chapter 5 presents the mapping of ClustalW onto the Cell BE processor. While applying different code optimizations, we focus on the limitations of the architecture and the programming model. We run experiments on real hardware to report performance results and scalability with the number of cores. As a result of the experience acquired with the applications and the Cell BE, we propose few new instructions aimed at improving the performance. We implement them in the CellSim simulator and report the achieved speedups.

Chapter 6 presents simulation results of the SARC manycore architecture running ClustalW and HMMER. We present a scalability study, where various parameters of the architecture are analyzed independently. Furthermore, we discuss the performance and scalability gap between a manycore system based on general-purpose cores and one based on accelerators. Finally, the chapter also analyzes the impact that different input sequences have on the parallelism of ClustalW, which, ultimately, determines the achievable performance.

Chapter 7 summarizes the work presented in this thesis and brings together the conclusions from the various chapters. Finally, some ideas for future work are proposed.

2

Bioinformatics Theory and Applications

Since this thesis' main focus is on computer architecture, this chapter provides an overview of the bioinformatics background needed to understand and put in context the work presented in the remainder of the thesis. We start by introducing the disciplines of molecular biology and bioinformatics, and continue in more detail with the sequence alignment problem and the description of the applications used in the thesis. Subsequent chapters in this thesis present experiments that need the concepts, algorithms and applications described here. Although, each of those chapters include brief reminders, this chapter is intended to be used as a reference.

This chapter is divided in sections as follows. Sections 2.1 and 2.2 briefly introduce the fields of molecular biology and bioinformatics. Section 2.3 presents the biological databases. In Section 2.4 we describe the algorithms and applications for sequence alignment, while Section 2.5 explains the multiple sequence alignment case.

2.1 Molecular Biology

Molecular biology is the field of biology that studies the processes that take place in living cells at the molecular level. It is tightly linked to biochemistry and genetics but it specifically aims at explaining the processes of replication, transcription and translation, that is, the transfer of genetic information from DNA to RNA and proteins [32].

In the first half of the twentieth century, early geneticist realized that in order to better understand the processes that create new living organisms, significant knowledge from physics and chemistry was needed [33]. That was how in the following decades, scientist from the three disciplines worked together and a

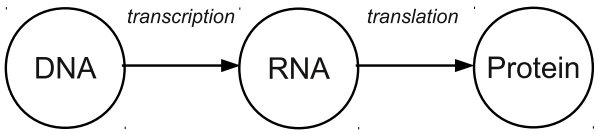


Figure 2.1: The transfer of genetic information.

new field emerged: molecular biology, still considered part of biology. In 1953 Watson and Crick [99] made use of X-ray imaging to propose and discover what today is accepted as an accurate molecular model of the DNA structure, marking the most notable achievement in the early days of molecular biology.

At the macromolecular level, the biological processes that take place in living cells involve a flow of information that goes from the DNA to proteins, as in Figure 2.1. The DNA is often abstracted as a sequence of elements that encode the genetic information used to "produce" a living organism. The process involves the generation of RNA from the DNA and from there, the production of proteins, the actual components that implement the cell's functions. DNA is composed of a long sequence of base pairs that are made of four types of nucleotides (Adenine, Cytosine, Guanine and Thymine). When RNA is produced from DNA, only one strand is left (of the double DNA helix) and the Thymine is replaced by Uracil. Groups of three consecutive nucleotides in RNA are called codons and a code is defined where each RNA codon corresponds to an amino acid, the components that make up proteins. Depending on what the purpose of a given study is, scientists maybe interested in analyzing a DNA, RNA or protein sequences. From an abstract computational point of view, the three of them are just sequences of symbols with a two-letters alphabet in the case of DNA and RNA, and, in the case of proteins, an alphabet that has 20 letters or more.

An accurate explanation of the biological processes mentioned in this section is out of the scope of this thesis and the interested reader is encouraged to read the bibliography [32, 33, 56, 73, 99].

2.2 Bioinformatics

After the discovery of the DNA molecule structure in 1953, the scientific community's interest in molecular biology grew rapidly and technological developments helped boosting more achievements in this discipline. In particular, the design of the first successful sequencing methods (i.e. lab procedures through

which a DNA sequence is read) in the seventies, made genetic sequences available to scientists for analysis. In the following years, sequencing methods were improved which eventually led to an explosion of genetic data. This marked the birth of the bioinformatics discipline, when the huge amount of biological data produced made analysis by hand impossible.

Bioinformatics is a discipline that develops algorithms and uses computer technology to solve problems in molecular biology. Its primary goal is to obtain biological knowledge from raw data stored in databases. Surely, the most notable achievement in bioinformatics so far is the sequencing of the human genome. In 1990 the Human Genome Project [11] was set up with the purpose of obtaining the full sequence of nucleotides in the human genome. Since the number of base pairs in the humane genome is in the order of $3 \cdot 10^9$, the project was tremendously ambitious also in computational terms. Many universities and research institutes around the world contributed with sequencing fractions of the genome and in 2003 the project was completed.

The development of better bioinformatics tools is not only important because of the exponential growth of databases but also because of its potential impact in important fields of science such us evolutionary biology and genetics. Moreover, by means of bioinformatics analyses, genetic-related diseases such us cancer and diabetes are being better understood so that cures can be developed hopefully soon.

Bioinformatics programs have also served to construct the *Tree of Life*. Figure 2.2 is a Leonard Eisenberg's [40] illustration of the evolutionary relationships between living and extinct species. Through mutations and adaptation to the environment, as in Darwin's theory [34], species appear as branches of a very large evolutionary tree of life. In more recent times, biologist have discovered and recognized that genetic information is also transferred in a horizontal way, that is, from one specie to another [46]. There is already evidence of horizontal gene transfer at least in prokaryotes such us bacteria and in unicellular eukaryotes.

There are numerous areas within bioinformatics, from sequence alignment, gene finding, phylogenetics and protein structure prediction to biological systems modeling and biomedical image processing. This thesis focuses on *sequence alignment*, a fundamental part of bioinformatics that is also used in some of the other areas.

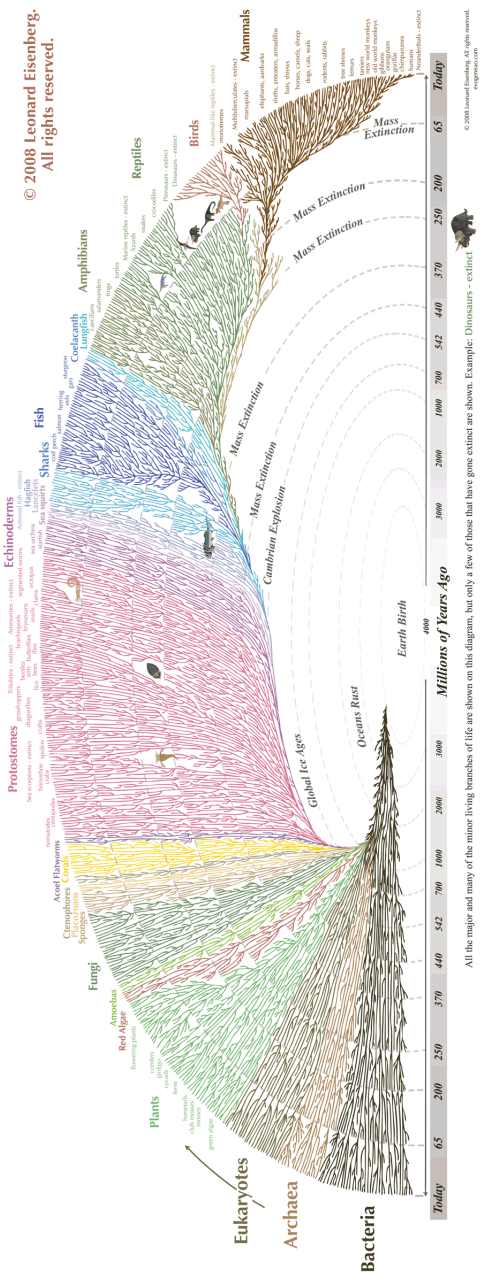


Figure 2.2: The Tree of Life.

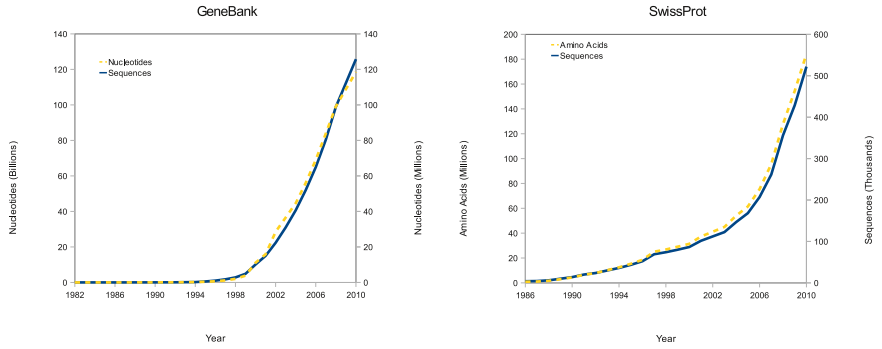


Figure 2.3: Exponential growth of biological databases.

2.3 Biological Databases

Databases containing biological information acquired in different ways are essential to bioinformatics and biology. The design and maintenance of such repositories are considered a discipline in itself, part of the bioinformatics. Nowadays, there exist many types of biological databases containing information about DNA, genomes, protein sequences, protein structure, phylogenetic trees, microarrays, etc. The exponential growth of most of the biological databases is the reason for the high interest in developing better and faster bioinformatics applications and computer systems. Figure 2.3 shows the exponential growth of the GenBank and SwissProt databases through the years.

Since this thesis focuses on sequence alignment applications, the relevant databases are the ones that contain DNA or proteins. In particular we have used the GenBank, SwissProt and Pfam databases for the experiments. The *GeneBank* [25] is a database containing all available nucleotide sequences and their protein translations. It receives submission from laboratories and sequencing centers and is synchronized on a daily basis with its partner databases, the *EMBL Nucleotide Sequence Database* [16] and the *DNA Data Bank of Japan* [9]. The 183.0 release of the GenBank in April 2011 reported 126551501141 bases from 135440924 sequences.

The *SwissProt* [15] is a database containing manually annotated and reviewed proteins. It is part of the UniprotKB [18] which also covers the UniprotKB/TrEMBL database, containing unreviewed and automatically annotated proteins. The 2011_06 SwissProt release from May 2011 reported 187423367 amino acids from 529056 sequence entries.

Alignment one:

```
g c t g a - a - - c g
- - c t - a t a a t c
```

Alignment two:

```
g c t g - a a - c g
- c t a t a a t c -
```

Figure 2.4: Two possible alignments of a pair of sequences.

Pfam is a database of protein families that are built from multiple sequence alignments (see Section 2.5) and represented by HMMs (see Section 2.4.4). Pfam is an essential part of the HMMER programs. The Pfam release 25.0 from March 2011 reported 12273 protein families.

2.4 Sequence Alignment

In bioinformatics, DNA, RNA and protein macromolecules are abstracted as just sequences of letters, called residues (either nucleotides or amino acids). When stored as raw data in bioinformatics databases, the long sequences of residues need to be analyzed by computer programs in order to obtain biological knowledge about the species they are part of. DNA sequences for example, contain genes (portions of DNA) that are responsible for the organism's phenotype. Evolutionary biologists are interested in finding genes that are common to several species, which may indicate a common ancestor in the evolutionary history. Sequence similarity may also be consequence of structural or functional relationships. A computer program can therefore intensively search for regions of similarity between sequences in order to detect such relationships.

Figure 2.4 shows two possible alignments of a pair of sequences. Residue mismatches are called substitutions (mutations in genetic terminology) and dashes (gaps) are called insertions/deletions depending on the point of view.

Sequence alignment refers to the search of such similarity regions within biological sequences, being them DNA, RNA or proteins. Besides the biological significance and interpretation of the results, the main problem of sequence alignment from the computer science point of view is the very large number of residue-to-residue comparisons that are needed when searching for similarities. Not only the logical definition of the problem makes it time consuming but also the fact that in practice, a single genome may contain in the order of

billions (10^9) of residues.

The problem of aligning two sequences is discussed in this section and is called *pairwise alignment*. Section 2.5 presents the *multiple sequence alignment* problem, that consist in searching for similarities that are common to a group of sequences rather than just two.

The dotplot

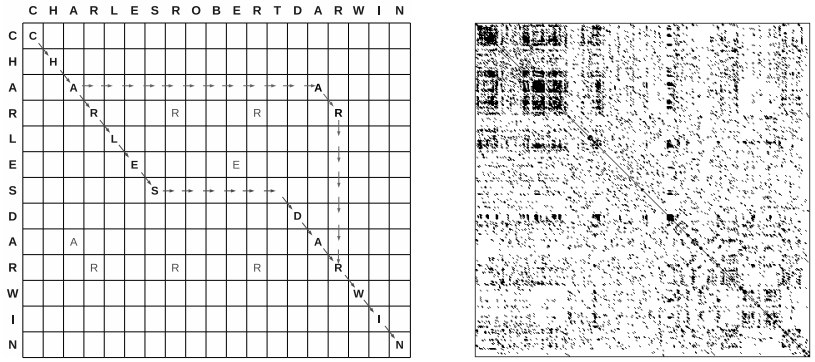
A simple visual method to align two sequences is the *dotplot*. In a two-dimensional matrix, one sequence is placed horizontally on top and the other one vertically on the left side of the matrix, as illustrated in Figure 2.5(a). In its simplest version, a dot is put only on matrix cells that correspond to a match between the two sequences' residues. Figure 2.5(a) shows and example that uses the short and the full name of a person as the two sequences to be aligned. Instead of simply dots, the corresponding letter is written on matching cells. Through a quick visual inspection, diagonal subsequences are highlighted to reveal the (obvious in this case) similarities between the two name strings. Figure 2.5(b) shows the DNA dotplot matrix of the human zinc finger transcription factor (GenBank ID NM_002383) sequence against itself. The main diagonal in black represents the perfect alignment of the sequence with itself. Other black areas show regions of repetitive patterns within the sequence.

Despite giving a quick visual impression of the relationships between two sequences, the dotplot method is very limited if more precise and quantitative knowledge is required from the sequence comparison. Furthermore, if more than two sequences need to be analyzed together, a dotplot cannot be used. Alignment algorithms are then required in order to measure the similarities and decide for the best alignment given some objective function.

Figure 2.5(a) shows the paths of two possible alignments where horizontal/vertical arrows indicate gaps on the vertical/horizontal sequence, and, the diagonal arrows indicate a match. Sequence alignment algorithms are used to decide what path is more significant.

Measures of Sequence Similarity

The *edit distance* is a type of metric used to determine the similarity between two strings. One such metric is the Levenshtein distance, where the allowable edit operations are insertion, deletion, or substitution of a single character. These operations are useful to biological sequence alignment because they



(a) Alignment of two name strings.

(b) self-alignment of a nucleotide sequence.

Figure 2.5: The dotplot alignment method.

```
a g - t c c
c g c t c a
```

Levenshtein distance = 3

Figure 2.6: The Levenshtein distance between two sequences.

can model the mutations that occur in genomes. The Levenshtein distance is defined as the minimum number of edit operations required to transform one sequence into the other one. In Figure 2.6 the Levenshtein distance is 3. However, things are not as simple in genetics. Biologists have learned that some mutations are more likely than others (e.g., a DNA mutation from Adenine to Guanine is more common than Adenine to Thymine) and they need alignment algorithms to reflect this property. For this purpose, substitution matrices have been built using statistical data from known sequences and mutations. Figure 2.7 shows the BLOSUM62 [50, 51], a common substitution matrix for amino acids alignments. The elements on the main diagonal have the highest values to encourage matching of identical residues in alignment algorithms.

Besides having substitution matrices for mutations, it is also desirable to score insertions and deletions (gaps) differently. The score given to an insertion/deletion is commonly called a *gap penalty* and there are three schemes used: constant gap penalty, linear gap penalty and affine gap penalty. While in the first case the same penalty is used for any gap, the linear scheme assigns a score that is proportional to the length of the gap open so far. The affine gap penalty defines two penalties, one for opening a new gap and one for extending an existing one.

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figure 2.7: The BLOSUM62 substitution matrix for amino acids.

After having substitution matrices and gap penalty schemes, scoring functions to measure the similarity between two sequences can be constructed. Alignment algorithms are designed to maximize a scoring function so that the best alignment is obtained.

Dynamic Programming

Figure 2.5(a) shows two paths that represent two possible alignments between the sequences. In fact, any path from upper left to lower right can form a possible alignment and a large number of them exist.

Dynamic programming is a method in computer science for solving complex problems by breaking them down into simpler subproblems that satisfy certain properties. In sequence alignment, dynamic programming techniques are used to find optimal alignments. The main idea applied is that once an optimal alignment (path) of a small region of the matrix is found, it can be reused when dealing with a larger region or the complete matrix. While the computational complexity of an exhaustive search for the optimal path is $O(n2^{2n})$, using dynamic programming reduces it to $O(n^2)$, with n being the length of the sequences. The Needleman-Wunsch and Smith-Waterman algorithms use the dynamic programming method and are widely used in bioinformatics (see Section 2.4.1).

Global vs. Local Alignments

Alignment algorithms can be global or local. Global alignment algorithms like the Needleman-Wunsch, which attempt to align every residue in every

```
Global FTFTALILLAVAV
       F--TAL-LLA-AV

Local  FTFTALILL-AVAV
       --FTAL-LLAAV--
```

Figure 2.8: Global and local alignments.

sequence, are most useful when the sequences in the query set are similar and of roughly equal size. Local alignments instead, like in Smith-Waterman, are more often used for dissimilar sequences that are suspected to contain regions of similarity within their larger sequence context. Figure 2.8 shows a global and a local alignment of the same two sequences. It shows that if sequences are not sufficiently similar, a global alignment tends to spread gaps that hide possible regions of similarity.

Optimal vs. Heuristic Algorithms

Despite of the computational complexity advantage of dynamic programming algorithms against an exhaustive search, it is often impractical to use optimal alignments. Many real alignment scenarios involve sequences that are too large or comparisons against huge databases that take too much time. When finding the optimal solution is not affordable, heuristic algorithms are used instead. For instance, BLAST (discussed in Section 2.4.3) is a popular tool based on heuristics that can "replace" Smith-Waterman (Section 2.4.1). In other cases and especially with the help of some sort of hardware acceleration, the Smith-Waterman algorithm can be used for practical purposes too.

In multiple sequence alignment (Section 2.5), algorithms than find the optimal solution are not used because of their intractability [41, 98]. Comparing it with the dynamic programming matrix problem in pairwise alignment, the multiple sequence alignment problem can be intuitively thought of as finding the optimal path through a many-dimensional array. However, since multiple sequence alignments are useful to biologists even if they are not optimal, several heuristics have been proposed. ClustalW (Section 2.5.1) is a widely used software tool that uses heuristics to perform multiple sequence alignments.

2.4.1 Needleman-Wunsch and Smith-Waterman

The *Needleman-Wunsch* algorithm [65], proposed in 1970, was the first to apply dynamic programming to compute a global pairwise alignment. In 1981, the Smith-Waterman algorithm was proposed as a modification to the Needleman-Wunsch in order to compute local alignments instead.

The Needleman-Wunsch algorithm is based on the the following formula:

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \tag{2.1}$$

And it is computed in three steps as follows:

- Make $H_{i,0} = 0$ and $H_{0,j} = 0$ for $0 < i < m$ and $0 < j < n$. The input sequences A and B have length m and n respectively.
- Compute all other values of matrix $H_{i,j}$ following the Formula 2.1.
- Recover the alignment by tracing back the path that led to the maximum score in $H_{i,j}$.

Formula 2.1 is used to fill up the dynamic programming matrix H, from which a similarity score can be computed and the actual alignment can be obtained with a traback. $H_{i,j}$ is the matrix position that corresponds to the i^{th} element of sequence A and the j^{th} element of sequence B. Each matrix element $H_{i,j}$ is computed in a recursive fashion as the maximum between three items that represent three options for continuing the alignment at a given position: match/mismatch, deletion and insertion. For a match/mismatch, the upper left element is added to the score $S_{i,j}$ that corresponds to the particular match/mismatch of element i from A and element j from B. As explained before, the score S is obtained from substitution matrices like BLOSUM62. The two other terms in the formula are obtained from subtracting the gap penalty d from the upper or left element, thus representing deletion and insertion.

After matrix H is computed, the position of the maximum score should be found. This score provides a similarity measure of the two sequences and it is sometimes used to decide whether recovering the actual alignment is worth or not. The alignment is then constructed by starting at the position where the maximum score is. At every step, the path should continue at the left, upper

left or upper position depending on which of the three could have led to the current element, according to Formula 2.1.

The *Smith-Waterman* algorithm imposes a small modification to the Needleman-Wunsch in order to implement a local alignment. Formula 2.2 shows that zero has been added as operand to the max expression. Intuitively it means that if the similarity to a given point is so low that it falls into the negative, the zero will restart the alignment in order to find new portions that can be good subalignments.

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \\ 0 \end{cases} \tag{2.2}$$

The Smith-Waterman algorithm is probably the most known in bioinformatics and is widely used, not only as standalone but also as part of heuristics in more complex applications like ClustalW. The Ssearch tool within FASTA (see Section 2.4.2), contains an optimized Smith-Waterman implementation used in parts of this thesis as a baseline.

2.4.2 FASTA

FASTA [60,72] is a software package for DNA and protein sequence alignment. It is freely available and there are web servers [4, 5] to let users run database searches using the FASTA tools. FASTA takes a given nucleotide or amino-acid sequence and searches a corresponding sequence database by using local sequence alignment to find matches of similar database sequences.

FASTA uses a heuristic algorithm whose aim is to be faster than Smith-Waterman without loosing much sensitivity. The first part in the FASTA algorithm consists of listing all $k - letter$ words of the query sequence, with k taking different values for nucleotides and amino acids. Then, as each sequence is read from the database it is also divided into its constituent overlapping words. These two lists of words are compared to find the identical words in both sequences. An initial score is computed based on the number of identities concentrated within small regions of the dot plot. If this initial score is high enough, a second score is computed by evaluating which of the initial identities can be joined into a consistent alignment using only gaps of less than some maximum length. This maximum lengths for gaps is set by the window size parameter. Finally, if the secondary score is high enough then a Smith-

Waterman alignment is performed within the region of the dot plot defined by the concentrated identities and the window size. This third score is reported as the optimal score.

The FASTA package also includes the *Ssearch* tool among others (FASTX, TFASTX, PRSS, PRFX, LALIGN, etc). Ssearch is an implementation of Smith-Waterman algorithm whose source code features optimizations targeted at modern processors.

2.4.3 BLAST

The *Basic Local Alignment Search Tool*, or BLAST [19, 20], is an application for sequence alignment, typically used to search databases. BLAST is freely available for download and through web servers [8] to run scans against genetics databases. A query sequence is compared against millions of longer database sequences in order to find similarities, and therefore, using local alignment. When a new genome is sequenced for instance, BLAST can be used to quickly search the presence of its genes in the already known database sequences. If the database contains one or more sequences that are fairly similar to the query, BLAST's results are often satisfactory. On the other hand, an optimal algorithm such as Smith-Waterman would be able to detect more distantly related sequences and build more accurate alignments.

The main idea behind BLAST algorithm is that homologous sequences contain short high scoring similarity regions that after being found, can be extended in both directions in order to improve the quality of the alignment. A list of short subsequences candidates to be the high scoring regions is extracted from the query in order to search them in the database. First, BLAST optionally filters out low-complexity regions that are not useful for producing meaningful sequence alignments. Then, an initial list of all $k - \texttt{letter}$ words is extracted from the query sequence. This list is extended with all similar words that score above a certain threshold in an un-gapped matching and using a substitution matrix like BLOSUM62. In the next step BLAST should identify all the exact database occurrences of all the words in the list that are used to seed a possible un-gapped alignment. The previous two steps are performed efficiently since BLAST keeps precomputed look-up tables to quickly extend the words list and find the word matches in the database. In recent BLAST versions, high-scoring segment pairs are defined by two non-overlapping matching words within a certain distance from each other, within the same diagonal. These regions are searched for in the database and the statistically significant ones are kept. Smith-Waterman is used in the last step to produce alignments with

matched database sequences.

BLAST and its variants (BLASTP, BLASTN, PSI-BLAST, BLASTX, TBLASTN, TBLASTX, etc.) are probably the most used bioinformatics programs as they provide a fast way of doing fundamental tasks for molecular biologists. BLAST is faster and more accurate than FASTA when high similarities are present. For more distant similarities, FASTA is more accurate.

2.4.4 HMMER

HMMER [7, 36, 37] is a free and commonly used software package for sequence alignment, originally developed by S. Eddy [37] while at Washington University. HMMER uses probabilistic techniques based on Hidden Markov Models [57] (HMMs) to store information on the alignment of protein or DNA sequences. A typical use of HMMER is to search databases (with its *hmmsearch* tool) for homologues of a query sequence or set of sequences that are modeled with a profile-HMM [37]. A multiple sequence alignment (Section 2.5) can be used to build a profile-HMM with *hmmbuild*, part of HMMER.

Hidden Markov Models

A Markov model of a system is a statistical model defined by a set of states and a set of probabilities for transitioning from one state to another. At any given time, the system is in one of its states. According to a matrix of transition probabilities A, at each time step the model transitions from the current state to another one (possibly the same state). The model is characterized by its n states and its $n \times n$ transition probability matrix A. Over time, the states through which the model transitions can be listed in a sequence. The Markov property of a system says that transitions from the current state to the next state are dependent only on a fixed number of previous states.

Hidden Markov Models (HMMs)[1] are a special type of Markov models in which the sequence of states cannot be observed directly. However, the states emit symbols and a sequence of symbols is observed instead. By looking at the sequence of symbols emitted, assumptions can be made on the model states that produced the symbols. Thus, for a symbol alphabet of size m, HMMs are further characterized by a $n \times m$ emission probability matrix B. Given an HMM with known parameters, one question to answer is: what is the most likely se-

[1] In biosequense analysis, HMMs are also called profile-HMMs because they represent the profile of a protein sequence for instance.

quence of states leading to the observed sequence of emitted symbols? The *Viterbi algorithm* [94] is used in HMMER to compute this sequence. Furthermore, the *Plan7 Profile HMM Architecture* [37] is the specific model used by HMMER to describe DNA/protein sequences.

In contrast to the Smith-Waterman and Needleman-Wunsch approaches, in the HMM approach the model instead of the algorithm itself determines whether local or global alignment is used. Also, the model determines if multiple domain hits per sequence are allowed. The main advantage of using HMMs over the dynamic programming methods is that gaps are modeled in a systematic way. For example, in Formula 2.2 d is used to assign a score to gaps. Even if linear or affine penalties are used, a drawback all these techniques share is that they assume that positions can be scored independently: higher-order correlations between consecutive residue positions in the sequences cannot be modeled.

Program phases

In this thesis we focus on one tool within the HMMER package, i.e. *hmmsearch*, but we refer to both HMMER and hmmsearch indistinctly. Figure 2.9 shows the execution scenario of hmmsearch: an HMM and a protein database are the program inputs, and, the output is a list of potential homologous sequences or hits. A block diagram of the main *hmmsearch* phases is shown in Figure 2.10. A query HMME is loaded from disk once and then the program enters in a loop that scans the entire database. For every sequence loaded from the database to the memory, a formatting function is applied first. It converts the amino acid symbols encoded in ASCII characters into an index code that allows for faster processing when computing the alignment. Then, the Viterbi algorithm is used to compute the alignment between the HMM and the query sequence, producing a bit score. In order to evaluate the statistical significance of the result, the E-value is computed using the bit score, and, depending on a given threshold the program decides whether it was a hit or not.

The processing shown in Figure 2.10 is to be applied to every sequence in the database in an independent manner. This means that every Viterbi job has no dependencies with other and, therefore, can be computed concurrently.

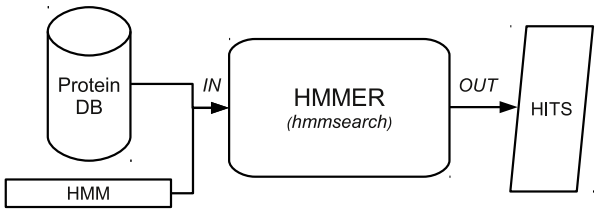


Figure 2.9: HMMER Input/Output block diagram (*hmmsearch*).

Profiling

As expected from theory, a real execution of HMMER on a processor results in the Viterbi function taking the vast majority of time. Profiling results of a random HMM (taken from the Pfam database) search against the SwissProt database show that the *P7Viterbi* function takes above 98% of the time.

HMMER2 and HMMER3

Although previously HMMER used to trade speed for sensitivity, the current version of the software (HMMER3) is considered to be as fast as BLAST while being more sensitive. The inclusion of HMMER2 in the SPEC2006 and the recent development of HMMER3 show its significance.

HMMER3 is a re-design of the HMMER2 program structure. With respect to hmmsearch, in HMMER2 the vast majority of the processing is done in the function that implements the Viterbi algorithm. Figure 2.10 shows how the HMMER3 processing is different, incorporating three main processing stages through which every database sequence is passed. Moving from the start to the end (or from left to right in Figure 2.10), sensitivity increases while speed decreases. The first stage uses the MSV filter, a fast algorithm to roughly estimate the similarity and filter out a lot of unrelated sequences. The second stage is equivalent to HMMER2 where the Viterbi algorithm is used to further refine the filtering. In the last stage, the Forward algorithm used to finally select the homologous sequences from the database.

2.5 Multiple Sequence Alignment

Section 2.4 dealt with the problem of aligning a pair of sequences, also known as pairwise alignment. In this section we present *multiple sequence align-*

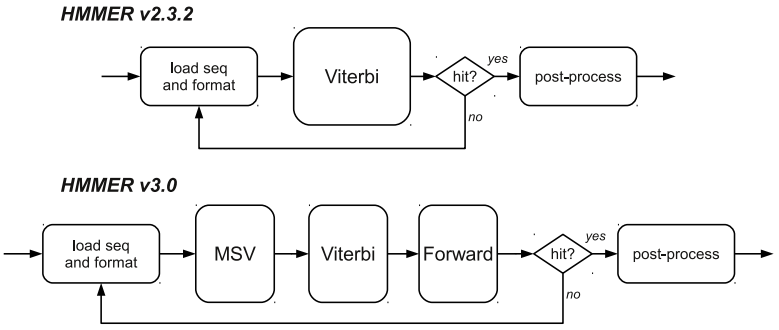


Figure 2.10: HMMER (*hmmsearch*) processing pipeline in versions 2 and 3.

ment, whose aim is to find regions of similarity that are common to most of the sequences in an input set. Unlike in searching a database with pairwise alignments, in multiple sequence alignment the input data of the problem is a limited set of sequences that are expected to have some degree of genetic relationship. If this premise is not satisfied, divergent sequences might introduce noise and make it more difficult for the algorithms to find the similarities among the rest of the sequences.

A multiple sequence alignment is evidently more computationally intensive than a pairwise alignment, even when having the same amount of input data. The matrix in the dotplot earlier discussed, grows to a many-dimensional structure in the multiple sequence alignment case, and consequently, the complexity of finding the optimal path explodes. The running time complexity can be expressed as $O(l^n)$, with l and n being the average sequence length and the total number of sequences to be aligned. Optimal algorithms for multiple sequence alignment are therefore not used due to their intractability [41, 98]. However, a number of heuristic algorithms have been proposed to find approximate alignments that are nevertheless very useful to biologists. These can be categorized as follows:

Dynamic Programming for MSA

The dynamic programming approach described before for pairwise alignments can be naturally extended to a multi-dimensional matrix to construct a multiple sequence alignment but with dramatic consequences in the required running time. However, in previous works [49, 59], methods have been proposed to reduce the search space. In those approaches pairwise dynamic programming alignments are performed on each pair of sequences in the query set, and only

the space near the n-dimensional intersection of these alignments is searched for the n-way alignment. The MSA program optimizes the sum of all of the pairs of characters at each position in the alignment (the so-called sum of pair score) and has been implemented in a software program for constructing multiple sequence alignments [45].

Progressive Alignment

The progressive alignment methods are among the most popular for multiple sequence alignment. The process starts by aligning two sequences. A third sequence is aligned to the previously obtained alignment and all sequences are processed the same way, one at the time, to progressively construct the multiple alignment. Different algorithms vary in the methods they use to orderly picking the sequences to be aligned at each step. The main drawback of progressive alignment algorithms is that any errors happening at a certain alignment step will propagate and possibly affect the final multiple alignment. Although progressive multiple alignments do not guarantee a global optimal solution, they are considered useful by biologists. Up to thousands of sequences are possible to align these days within a reasonable amount of time (hours to a few days). The most popular software package for multiple alignments is ClustalW, a progressive alignment heuristic that is also the focus of this thesis.

Iterative Methods

Iterative methods have been proposed in order to reduce the inherent errors in progressive alignment techniques. These type of algorithms iterate the multiple alignment process in order to go back and re-align previous sequences in light of new sequences. Examples of software that belong to this category are MUSCLE [38, 39] and CHAOS/DIALIGN [28, 67].

Hidden Markov Models

Hidden Markov Models (HMMs) can also be used for multiple alignments. The Viterbi algorithm is used to align sequences to the partial multiple alignment, one at the time. A fundamental difference in this case is that every time a new sequence is aligned, the alignment of new sequences gets updated. However, like in the progressive methods, the final result depends on the order in which the sequences were picked for alignment. Examples of software that uses HMMs for multiple alignment are HMMER [7] and SAM [66].

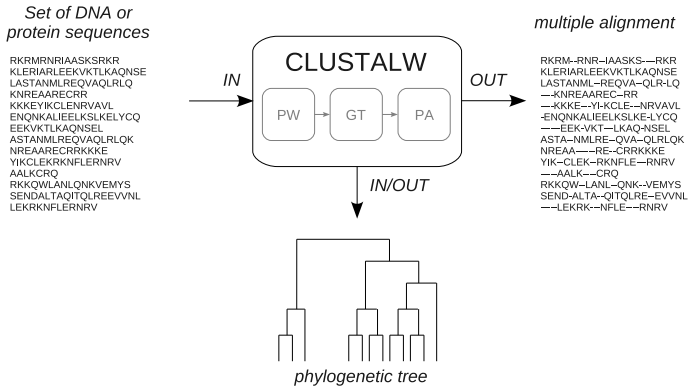


Figure 2.11: ClustalW Input/Output block diagram.

2.5.1 ClustalW

ClustalW is a program to perform multiple sequence alignment. It was developed by J.D. Thomson et al. [90] and is available online through the European Bioinformatics Institute website [2]. Besides the command-line interface of ClustalW, there also exists a graphical version called ClustalX. ClustalW takes DNA or protein sequences as input in several formats (e.g FASTA, Clustal). ClustalW uses a heuristic algorithm that falls under the progressive alignment techniques. The program is divided into three main phases: pairwise alignments (PW), guide tree (GT) and progressive alignment (PA). ClustalW can be used to produce only a phylogenetic tree from a set of sequences, to produce a multiple alignment from a phylogenetic tree or to produce a multiple alignment from a set of sequences. Figure 2.11 illustrates the input/output viewpoint of ClustalW.

A multiple alignment also contains gaps, substitutions and matches so that the global score of similar aligned regions is maximized. These regions are often indicated with colors to facilitate the analysis. Figure 2.12 shows a multiple alignment as colored by ClustalX. Specific colors are assigned to either the occurrence of some significant sub-sequence of residues or to segments whose percentage of appearance in a given column is above certain threshold.

The three main phases of ClustalW function in a sequential chain as shown in Figure 2.11, where only one stage is active at a time. In the following we describe the three program phases.

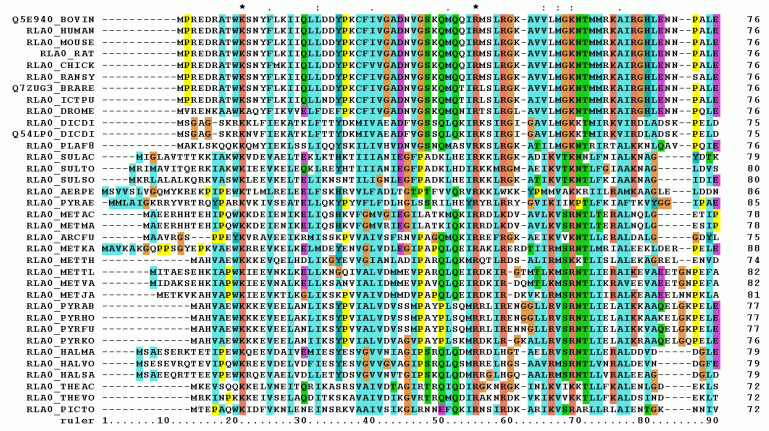


Figure 2.12: Multiple sequence alignment produced with ClustalX.

Pairwise Alignment

In the first stage of ClustalW, the objective is to compute a similarity score of all individual sequence pairs. To do so, ClustalW calculates independently, the alignment score of all possible pairs in the input set, which leads to $n(n-1)/2$ alignments for n sequences. Every alignment job takes the two sequences as input and produces a scalar similarity score that is stored in a table for later processing. Since there is no dependency among alignments jobs, these can be computed concurrently in an efficient way.

Although the pairwise alignment phase in ClustalW involves several functions that process the sequences, the majority of the running time is spent on *forward_pass*. This function uses the Smith-Waterman algorithm to compute a preliminary similarity score that is later refined with further processing. The output of the pairwise alignment phase is a table, filled in as a diagonal matrix, that provides the similarity score between any pair of sequences in the input set. Figure 2.13 shows an example of such a table for a set of four sequences.

Guide Tree

After the similarity table is completed in the first phase, the guide tree phase uses the table to construct a phylogenetic tree, either as program output or as input for the progressive alignment phase. The tree is constructed with the Neighbor-joining algorithm proposed in [81]. The goal of the guide tree phase is to produce a tree-like structure where sequences with a high degree of simi-

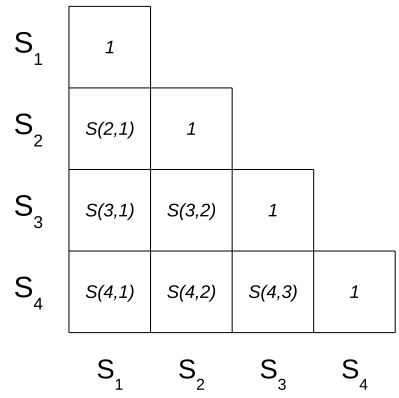


Figure 2.13: Pairwise alignment output similarity table.

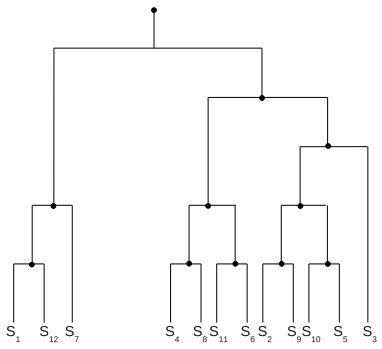


Figure 2.14: Guide (phylogenetic) tree.

larity are placed in near branches to form clusters. This organization allows for an efficient alignment order to build a multiple alignment. Figure 2.14 shows an example of a guide tree containing twelve sequences.

Progressive Alignment

In the third phase, the progressive alignment is performed by aligning the sequences, one by one, following the tree topology. Sequences that are more similar are aligned first so that the more divergent ones do not introduce noise. On each step, a sequence is aligned to the accumulated multiple alignment. Figure 2.12 shows and example of a progressive multiple alignment result. Vertical columns of colors represent similar segments found in most of the sequences.

Table 2.1: ClustalW profiling for various input data sets.

Function	Phase	InputA	Input B	Input C
forward_pass	(PW)	26.1%	59.4%	79.7%
prfscore	(PA)	8.7%	20.5%	8.8%
pdiff	(PA)	$\sim$ 0%	17.9%	7%
diff	(PW)	34.8%	0.3%	0.6%
pairalign	(PW)	21.7%	1%	1.8%
other functions	-	8.7%	0.9%	2.1%

One important implication of following the tree topology to construct the multiple alignment is that the number of alignment jobs that can be computed concurrently is reduced. At the beginning, all the nodes whose both branches are leaves of the tree can be processed concurrently. However, as the tree is "consumed", less nodes can be processed in parallel. In the example of Figure 2.14, 5 nodes can be processed concurrently in the first step, then 3, 1, 1 and 1. This parallelism will also heavily depend on the sequence data.

Profiling

Running ClustalW on a processor to extract profiling information gives us an insight into the importance of each of the program phases with respect to execution time. Table 2.1 shows that the *forward_pass* function from the pairwise alignment phase takes most of the time. Other functions belonging to the progressive alignment phase such as *pdiff* and *prfscore* take most of the remaining time. Table 2.1 presents three columns with different input data sets that are taken from BioPerf [22], a benchmark suite that often used in the literature. It should be noted that for *Input A*, *forward_pass* does not dominate the time anymore. However, this input contains few and very short sequences and is therefore less interesting when looking for ways of improving performance. Results in Table 2.1 were obtained with ClustalW v1.83 running on an Intel Core 2 Duo E8400, using a single core and with 4GB of RAM.

ClustalW2 and Clustal Omega

ClustalW2 [3, 91] is the current stable version of ClustalW. At the time when we started this work (2007) the latest ClustalW version available was 1.83. For the experiments in this thesis, we did not upgrade to ClustalW2 because the parts of the code we cared about have not been modified. ClustalW2 is basically a C++ re-write that incorporates the same algorithms in the PW and PA phases, therefore not affecting our analysis. It does not include code optimizations of these two kernels either. ClustalW2 includes new algorithms for constructing phylogenetic trees, concerning the GT phase that we do not study in this thesis.

In recent days, a new variant of the Clustal series of programs has been announced. The new version has been named Clustal Omega and it is claimed to be capable of aligning about 10 times more sequences than ClustalW, using the same time budget. Support for multithreading has been added and the algorithms have been modified although the details are not clear at this moment.

2.6 Summary

This chapter started with presenting the background on the bioinformatics knowledge relevant to this thesis. A general introduction to the field of molecular biology and bioinformatics has been followed by a brief description of the databases used in the experiments. An extensive description of the sequence alignment problem has been presented next. We have defined the problem, discussed the different types of algorithms that have been proposed and presented relevant applications for sequence alignment such as BLAST, FASTA and HMMER, which is later used in our study. Next, we have introduced the multiple sequence alignment problem and the different algorithmic techniques used to build these type of alignments. Last, we present and describe in detail ClustalW, one of the two applications studied in this thesis.

3

Architectures, Simulators and Related Work

In this chapter we start by introducing the multicore architectures used in our studies, in Section 3.1. First we describe the Cell BE and after the SARC architecture. In the second part of the chapter, in Section 3.2 we present the two simulators used: CellSim and TaskSim. Last, Section 3.3 presents the related work.

3.1 Multicore Architectures

Multicore is today considered a widespread paradigm in the field of microprocessor design. As discussed in Chapter 1, this has been the computer architecture community's answer to the appearance of several CMOS technological barriers. Most notably, the physical limits for power dissipation do not longer allow for obtaining a performance increase by using higher clock frequencies, as it used to be. With multicores, however, performance scaling is not straightforward and it largely depends on the available application's parallelism and the programming model.

Sequence alignment applications exhibit multiple levels of parallelism. Multicore processors with individual cores capable of extracting data-level parallelism are therefore good candidates to increase performance and efficiency. In this section we present the two architectures studied in our work. The first one (Cell BE) is a state-of-the-art commercial processor and the second (SARC) is a research architecture aimed at performance scalability for the manycore era.

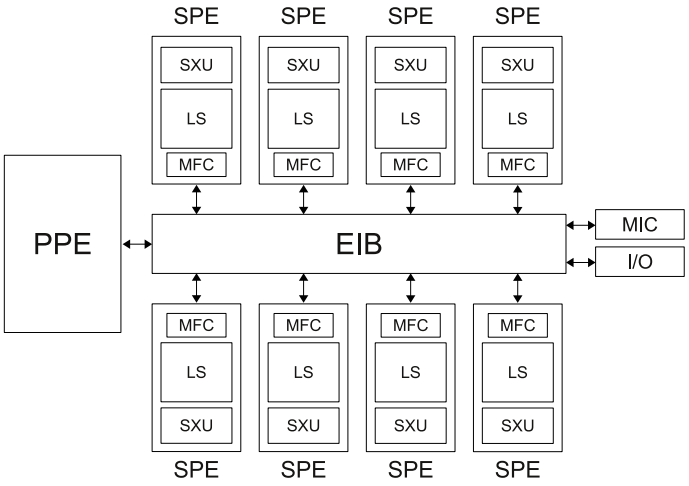


Figure 3.1: The Cell BE processor block diagram.

3.1.1 The Cell Broadband Engine

The Cell BE processor [48, 55] was designed in a joint effort from IBM, Sony and Toshiba, mainly to be used for Sony's game console PlayStation 3. Figure 3.1 shows the Cell BE architecture, composed of a PowerPC Processing Element (PPE), eight Synergistic Processing Elements (SPEs) and the Element Interconnect Bus (EIB). The PPE is a 2-way Simultaneous Multithreading (SMT) dual-issue in-order PowerPC processor. The SPEs are SIMD processors, each having a 256KB scratchpad memory referred to as Local Stores (LS). The EIB is a circular ring comprising four 16B-wide unidirectional channels that connect the SPEs, the PPE, a dual-channel memory controller and a double I/O controller. The operating system runs on the PPE and software can spawn threads to the SPEs. Data has to be explicitly copied to the SPEs LSs using Direct Memory Access (DMA) commands. The Memory Flow Controller (MFC) in each SPE takes care of these DMA transfers and it does so in parallel to the SPEs' execution, allowing it to hide the DMA transfers latency. Although these features enable programmers to write highly efficient code, they also significantly increase programming difficulty.

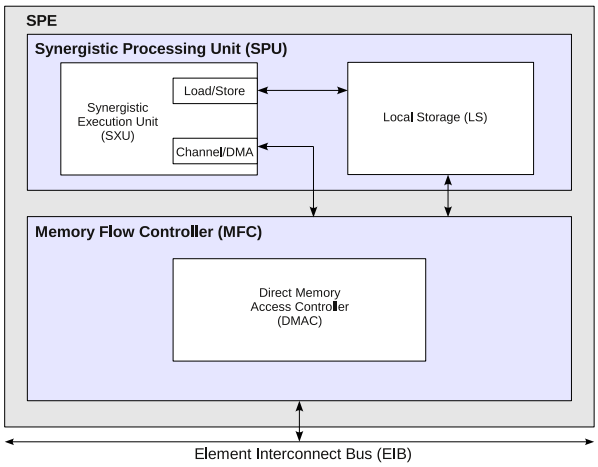


Figure 3.2: Synergistic Processing Element (SPE) block diagram.

The Synergistic Processing Element

Arguably, the most innovative component in the Cell BE is the SPE. It is composed of the Synergistic Processing Unit (SPU)[1] and the MFC as seen in Figure 3.1. The SPEs are dual-issue in-order SIMD processors with 256KB LS and 128 registers, 128-bit wide. The SPEs are designed to run portions of applications with little flow control but with a lot of repetitive processing on large amounts of data. Because only SIMD registers and functional units are present, scalar processing is achieved by using only one slot of the SIMD word. The compiler takes care of producing scalar code using SIMD resources, which may lead to significant overhead because of the additional instructions needed. The SPE designers have also opted for not including common hardware structures like dynamic branch predictors and out-of-order execution support, found in most superscalar processors. By doing so, the circuit complexity is reduced and a high power efficiency is achieved for appropriate applications.

The SPE relies on the compiler to perform branch prediction. Special instructions can be inserted at compile time to indicate the expected outcome of a branch. This mechanism works well for a simple branch behavior like the one in loops, but fails in control flow code with dynamic behavior.

The SPU has two execution pipelines and is therefore able to commit up to 2 instructions per cycle. One pipeline processes fixed-point and floating-point operations while the other one takes care of DMA and operations, branches

[1] In this thesis we use SPE and SPU indistinctly.

and loads/stores. Each SPE runs its own binary code that should be stored in its LS. Load/store instructions are also served by the LS without any hardware support for caching. To read/write the main memory, the SPU has to issue a DMA command that is handled by the MFC. Additionally, an SPE can also read/write another SPE's LS through DMA operations. Figure 3.2 shows a simplified block diagram of the SPE.

The Programming Model

The introduction of a new architecture style in the Cell BE also required a programming model with special capabilities. Since the release of the Cell BE, a Software Development Kit (SDK) was made available to programmers. The SDK includes libraries that are used to control the specific Cell BE features such as the mailboxes.

The PPE is the component that runs the operating system in the Cell BE and offers binary compatibility with the PowerPC ISA, including its SIMD instruction-set known as Altivec. Thus, any older program that was compiled for a PowerPC processor can execute straight away on Cell BE. On the other hand, such a program would only be able to use the PPE and none of the SPEs, with obvious consequences in terms of performance. In order to use the SPEs, a program has to be written (or re-written) using libraries that enable the programmer to interface with the architecture. Actually two different programs should be written: one for the PPE and one for the SPE. The SDK comes with the *libpthreads* library that extends the standard POSIX Threads and can be used to create and manage software threads running on the SPEs. Special functions are used to load the SPE binaries onto their local stores and a thread is associated with them. Synchronization among threads can be done via *mailboxes* or *signals*. These low level mechanisms have dedicated special hardware registers that can be used to send and receive controls between threads, using polling or interrupt driven techniques.

Due to the memory organization in the Cell BE, the data should be explicitly copied from the main memory to the local stores before the SPEs can operate on them. When setting up a thread, the PPE attaches a pointer to the main memory location where the input data is stored. The SPE issues a DMA command that takes care of bringing the data (to its local store) and copying back the results after finishing a task. Since the DMA operation is handled by an independent DMA controller, the programmer can use multi-buffering in order to hide the transfer latency while some computation takes place. One critical issue in this respect is the restriction on data alignment. In order to keep the

hardware simpler and faster, data transfers in the Cell BE must be 16-byte aligned, that is, the origin and destination addresses in any DMA operation have to be multiples of 16, so that the 4 least significant address bits can be ignored. This poses a programming hassle since every variable or data structure to be transferred should be carefully aligned by the programmer. Special attention should be paid to multi-dimensional arrays and data structures in the way they are allocated so as to satisfy the alignment condition.

The SPE program is compiled with an SPE compiler, different from the one used to compile the PPE code. Although a standard plain C program can be compiled and executed, due to the SPE SIMD nature and the lack of many hardware mechanisms such as dynamic branch prediction and out-of-order execution, the performance is likely to be low. In order to efficiently use the SPEs, the SDK provides a C language extensions library containing intrinsics for the programmer to directly use the SPE SIMD instruction-set in a convenient way with respect to data types and register handling. This library also contains functions to control the DMAs, mailboxes and signals.

Although the various Cell BE programming aspects are not complex nor difficult to understand for most programmers, all together they do contain a significant amount of details that will slowdown the programmer's job. It is for this reason that other programming models have been proposed in literature in order to improve Cell BE's ease of programming (e.g. [23]).

3.1.2 The SARC Architecture

The second architecture considered in this thesis is shown in Figure 3.3. It is a clustered (or tiled) manycore interconnected with a system of hierarchical buses. Clusters of 8 (P)rocessors (similar to a single Cell BE) are grouped together through a local data bus (LDB) and all clusters are connected by the global data bus (GDB). The memory interface controllers (MIC) to access off-chip DRAM memory and the L2 cache banks are also connected to the GDB and shared by all processors.

It is a heterogeneous architecture given that different processor types coexist. The (M)aster processor is a powerful out-of-order superscalar processor that can efficiently run the operating system and the applications control sections. It accesses memory through a private L1 instruction/data cache and is connected to the rest of the system through the GDB. (W)orker processors perform data processing, that is, the bulk work of the targeted applications. Each worker can access three different memory types: its own SPM, the global shared memory

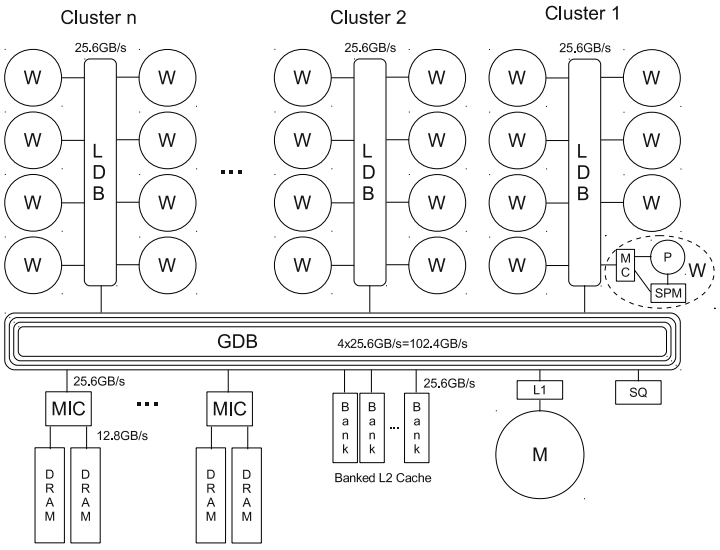


Figure 3.3: The SARC architecture.

and the other worker's SPM. For the last two cases, the memory controller (MC) DMA engine is used for the data transfers, decoupling them from the worker processor execution.

Each LDB is a bus with a ring topology, able to transmit 8 bytes per cycle at 3.2GHz (that is, 25.6 GB/s). Since the GDB is expected to handle more traffic, its bandwidth is increased by adding more rings. Every ring provides an independent communication channel between any pair of components in the system. Main memory is composed of several off-chip DRAMs, controlled by one or more MICs each providing up to 25.6GB/s. The DRAMs are DDR3-1600 with a peak bandwidth of 12.8GB/s, two of them are connected to every MIC. Fine-grain interleaving at the MIC level and then of DRAM channels is used, that is, consecutive memory lines change MICs and DRAMs so that large DMA requests of consecutive memory ranges can be processed in parallel accessing several DRAMs in parallel.

The L2 cache is organized in banks, each able to provide 25.6GB/s. As in the MICs case, fine-grain interleaving in accessing L2 banks provides high bandwidth for accesses to consecutive addresses. Since data transfers in this architecture mostly rely on DMAs, fine-grain interleaving enables the cache banks to serve multiple parts of a single DMA request in parallel. As a consequence, the effective bandwidth observed by the request can be higher than

that of a single bank. The L2 is meant to take advantage of data reuse among different workers. However, since the data set of one task easily fits on the worker's SPM, there is no need to have per worker L1 caches. Having them would imply a lot of extra area, complexity to support coherency and power.

The synchronization queue (SQ) has two hardware FIFOs implemented in an independent memory module connected to the GDB. One queue is for the master to submit tasks and the other for the workers to report task completion.

3.2 Simulators

In this thesis we have used two different simulators for two specific purposes. With the CellSim simulator we are able to explore instruction-set extensions for the Cell BE SPU in Chapter 5. Afterwards, TaskSim is used in Chapter 6 in order to study the scalability of the target applications when having hundreds of cores available. In the following we briefly describe the two simulators.

3.2.1 CellSim

For the performance evaluation of the proposed instruction-set extensions in Chapter 5, we have used the CellSim [1] simulator. Based on the UNISIM framework [12], CellSim is a modular simulator that allows one to test modifications to the Cell BE architecture. CellSim uses a set of SPU pipeline parameters that allows one to calibrate the accuracy of the simulations [63].

New instructions to the baseline ISA can be described in C language. Additionally, in order to make easy use of the new instructions, intrinsics can be defined for the CellSim GCC compiler to automatically recognize them. The code running on the SPE can be instrumented in order to generate statistics for the CellSim profiler. This gives us access to useful data per core and for the code section selected, such as: cycles, instructions executed, loads/stores count, load/stores stalls, channel stalls, branch prediction accuracy, etc.

In Chapter 5 we propose new instructions for the SPE ISA. Since those instructions are independent of the use of multiple cores and due to the long simulation times, we have only simulated the ClustalW most important function (*forward_pass*), running on one SPE and processing two 50-symbols long input sequences. The previous conditions do not affect the analysis since the number of loop iterations and computations only depends on the inputs size but not on the data content. That is, the speedup will remain the same for any

other input data.

3.2.2 TaskSim

The study presented in Chapter 6 of this thesis has used the *TaskSim* simulator [75, 77] that models the SARC architecture. The simulator uses a hybrid trace-driven high-level/cycle-accurate technique, that is, some components are simulated very accurately while others are abstracted out, depending on the aspects one wants to analyze. The simulator input are trace files that describe the application functioning by providing three types of information: data transfers, computational bursts and synchronization signals. To obtain these traces we have manually instrumented a ClustalW (v.1.83) port to the Cell BE [55]. Thereafter the instrumented code was run on an IBM QS21 Blade, running at 3.2GHz with 4GB of RAM. The code has been compiled with GCC4.1.1 and -O3 flag. Only the master and one worker processor (that is, a PPE and a single SPE in Cell BE) were used to generate the trace. This in order to create a single pool of tasks that can be dynamically scheduled to any of the simulated worker processors.

Since the processor's load and store operations are served by the SPM (without polluting any caches), the execution time of processing a given task (excluding synchronizations or DMA waits) is independent of the activity in the rest of the system. Therefore, the simulator does not "waste" time on accurately simulating the details of the internal execution of a core. On the other hand, TaskSim does simulate in great detail (cycle-accurate) the data transfers (that is, buses contention, caches and memory) and the synchronization signals exchanged among processors.

Simulation Setup

Figure 3.4 shows the simulation setup we have used with TaskSim. The simulator's inputs are traces describing the relationship among application's tasks and a configuration file with the architecture parameters. The simulator's outputs are also traces that describe the execution in the simulated architecture and a performance statistics file.

To ensure the accuracy of our simulations in Chapter 6, we ran ClustalW-PW on a Cell BE Blade using 8 SPEs and compared the execution time with that obtained from simulating a system with equal configuration. This allowed the accuracy verification when simulating multiple cores from a trace generated

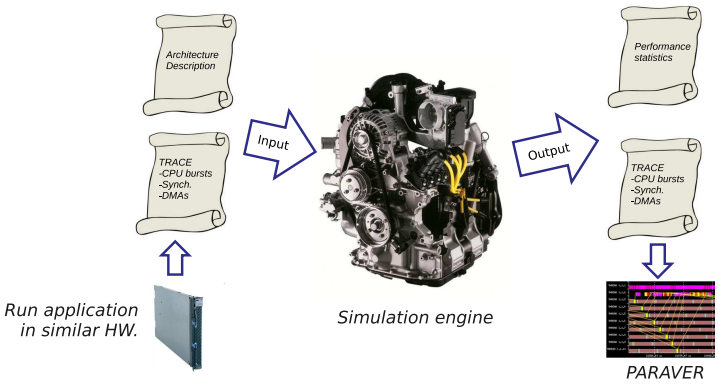


Figure 3.4: The TaskSim simulation scenario.

using only one. Results showed that difference between the two measurements is under 2%, considered adequate for the purpose of our study.

The simulator output's traces (like the ones shown in Section 6.4.1) are analyzed using the performance visualization tool Paraver [13].

3.3 Related Work

3.3.1 ClustalW Implementations

Various implementations of bioinformatics applications on different platforms have been reported in the literature. Some of them are based on Single-Instruction Multiple-Data (SIMD) augmented general purpose processors [78, 84] to exploit the fine-grained parallelism present in the sequence alignment applications. However, due to the exponential growth of biological databases, coarse grain parallelism needs to be exploited too.

Many studies about bioinformatics workloads target parallel machines combining the SIMD approach with multiple processing nodes. This in order to additionally distribute the job among the different nodes. Most of these studies focus on performance evaluation and parallelization on large high-performance supercomputers [86]. These alternatives, however, are expensive and exhibit severe limitations especially in terms of power consumption.

The use of heterogeneous multicore architectures on a single chip, e.g. the Cell BE, combines the parallelism benefits of multiprocessor systems, with the lower power consumption and higher speed interconnects of the systems

on a chip. However, these alternatives have not been completely studied as a solution for bioinformatics applications. Sachdeva et. al [79] present preliminary results on the viability of Cell BE for bioinformatics applications (ClustalW, Ssearch and Hmmer), all performing sequence alignment. In the case of Ssearch, a preliminary evaluation is reported that uses the SPUs for a pairwise alignment of only 8 sequence pairs that fit entirely in the LS memories. We believe that in order to get valid conclusions and given the different programming strategies and models that Cell BE offers, it is important to analyze the architecture behavior under the realistic conditions that are more demanding, such as using actual databases containing many sequences with various sizes.

Various works are found in literature with respect to ClustalW parallelizations. They have used different implementation platforms: from multiprocessor machines and GPUs to FPGAs and custom made ASICs. In [26, 30], results of distributed memory parallelizations (using *MPI*) of ClustalW are presented. The PW and PA stages were parallelized targeting computer clusters and reported speedups without any analysis. In [30], the performance scalability is fairly good up to 8 processors while in [26] it lasts up to 16 processors. The main reason for this difference is that the input sequences used in [26] are about 4 times longer on average. This results in the PW phase taking much longer than the other phases. As a consequence and since PW is the most scalable ClustalW phase, the overall performance scalability improves. The main performance bottleneck is the long computation time needed for each PW alignment because of the use of sequential codes running on a rather old processor (Pentium III). More recently, other works [52,53,80,93] have studied ClustalW's performance on the Cell BE [55]. In [93] ClustalW is parallelized for Cell BE. PW scales close to linearly as well and the inside of PW is parallelized with SIMD which greatly reduces its execution time. As a consequence, PA becomes the predominant stage and due to its more limited parallelism, the overall application's scalability saturates much more quickly. Their work is mainly focused on various programming optimizations while our interest is on discovering architectural limitations for a wider range of bioinformatics applications.

Vandierendonck et al. [93] have produced a highly optimized version of PA specifically tuned for using 6 SPEs. Two master SPEs independently compute the forward (FWD) and backward (BWD) loops while each of them get help from two other SPEs (four SPE helpers in total) that take care of computing the *prfscore* function, needed in the inner FWD/BWD loops. Most of the code running on the SPEs is vectorized and data communication is double-buffered. Although the speedup achieved with respect to the sequential code running

on the PPE is significant (5.8X), the parallelization is fixed, that is, the code is only able to run with 6 SPEs, not with more, not with less. This does not allow to study scalability behavior when the aim is to look forward into future architectures where more cores will be available. Different to what has been done in previous works, our aim in Chapter 5 is to measure and analyze how the contribution to performance changes for different PA parts when varying input size and number of cores. For this purpose, we have parallelized PA in a flexible way that can use an arbitrary number of SPEs. Although the forward and backward loops remain *2-way* parallel, the *prfscore* is now *n-way* parallel. Details, benefits and drawbacks of doing this are discussed in Chapter 5.

Results of previous work show that despite of a non-trivial programming model, specific Cell BE features like the Local Stores (LS) and the SIMD processing can achieve significant performance improvements for bioinformatics applications, including ClustalW. Besides, ClustalW-PW is shown to scale well up to the 8 SPE cores available in a Cell BE. Our work (Chapter 6) aims to extend the analysis of parallelism in the bioinformatics domain to future architectures containing hundreds of cores. While previous works report that ClustalW-PW has no scalability issues (for tens of cores), we show that when using hundreds of processors (in a CMP), performance will saturate due to several reasons (Section 6).

In [29] *pthreads* have been used to parallelize ClustalW using a shared memory model. Experiments reported focused more on varying the number of input sequences and their length. However, a machine supporting only 4 threads is used and the analysis is minimal. SGI has also announced an *OpenMP* parallelization for shared memory machines but the source code is not available.

Results reported in most of the previous work focus on describing application mapping decisions onto commercial hardware and present speedups compared to the competitor (e.g. GPU vs. Cell BE). However, all lack in-depth analyses of hardware resource utilization that are needed in order to find the real bottlenecks. We instead, aim at not only efficiently mapping ClustalW to a scaled multicore but also at finding the true bottlenecks that would limit performance scalability. This analysis and the related findings form the main contribution of Chapter 6. We envision this knowledge as very useful for designers of future multicore systems targeting applications from the bioinformatics domain.

The SARC architecture used in our study is presented in [75], where results are reported for the comparison of one pair of very long sequences, using FASTA-SSEARCH34 [4, 71]. We complement those experiments by looking at a different problem: the multiple sequence alignment with ClustalW [90]. Our

results show that although ClustalW-PW also tolerates high memory latencies (similar to the observations in [75, 82, 83]), the bandwidth requirements are quite different. Moreover, we study a manycore based on ASIC accelerators while previous work [75] used ASIPs instead.

Since the release of the Cell BE processor, there have been a number of articles reporting experiences of mapping bioinformatics applications to this new architecture. Some of them have used sequence alignment applications and explored parallelization strategies and manual code optimizations [27,74,79,92], but none of them have evaluated the instruction set efficiency. Those works have shown that both parallelization and the use of specialized cores can provide significant speedups, yet staying within a general-purpose platform. However, the traditional focus on multimedia and networking applications have not allowed bioinformatics to efficiently benefit from current architectures. In [52], we investigated the limitations of the Cell BE architecture when targeting bioinformatics. Some of the limitations mentioned there have been later considered by adding new instructions (Section 5.4).

Hardware support for bioinformatics has been explored in the form of fully custom accelerators, most of them using FPGAs [42, 47, 68, 69, 95] and some ASICs [35, 85]. Although capable of achieving very significant speedups, FPGA solutions suffer from low power efficiency and programmability issues. On the other hand there is also the case of Anton [85]. D.E. Shaw Research is about to finish the implementation of this new special-purpose machine to target molecular dynamics simulations. Although containing a programmable processor core, Anton's architecture is fully application-specific aiming at very high speedups for a single application regardless of the high cost of such a solution.

To the best of our knowledge, there has not been previous works on instruction-set extensions targeting bioinformatics applications. It should be noticed however, that in a timid attempt Intel's SSE4.2 (the newest version of their SIMD instruction-set extensions) featured one instruction aimed at genome mining (an application closely related to sequence alignment). Although this fact shows a new interest to offer support for bioinformatics applications from the general-purpose stream, still much is to be investigated so that efficiency gains are significant.

Table 3.1: Performance comparison of various SW and HW implementations of Smith-Waterman.

Reference	Performance	Speedup
Cell BE [52]	0.4GCUPS	1×
Cell BE [44]	2GCUPS	5×
GPU [58, 61]	5GCUPS	12.5×
FPGA [24]	14GCUPS	35×
CGRA [61]	76.8GCUPS	192×
FPGA [14]	688GCUPS	1720×

Comparing per-core performance

There have been efforts on optimizing sequence alignment kernels at a finer grain on software and hardware. In [52], the SIMD capabilities of the Cell BE are exploited, achieving 0.4GCUPS per SPE. GCUPS (Giga (10^9) Cell Updates Per Second) is a common metric used for Smith-Waterman implementations that allows us to compare with other reported results. An optimized version that avoids misaligned memory accesses is presented in [43,44], achieving 2GCUPS. GPU implementations [58, 61] have reported 5GCUPS and FPGA designs [24] 14GCUPS. Liu et.al. presented in [61] a careful comparison of his coarse-grain reconfigurable architecture (CGRA) performance (76.8 GCUPS) with the mentioned GPU and FPGA results. Lastly, an even higher performance (688GCUPS) has been recently reported by a commercial FPGA-based product [14], likely using more hardware resources though. In Chapter 6 we study a multicore architecture where each core has the performance of either a general-purpose processor [43, 44, 52] or an application-specific accelerator [14, 24, 58, 61]. Table 3.1 shows the difference in throughput of various implementations of the kernel targeted in Chapter 6.

3.3.2 HMMER Implementations

HMMERCELL [62] is the port of *hmmsearch v2.32* to the Cell BE architecture. In [62], detailed information on the implementation and parallelization strategy is provided, along with raw performance data where it is benchmarked against commodity x86 architectures. Compared to the AMD Opteron platform (2.8 GHz, 1-4 cores) and the Intel Woodcrest platform (3.0 GHz, 1-4

cores), a single Cell BE is reported to be up to thirty times faster than a single core Intel or AMD processor.

In this thesis, we build a model of HMMER for a master-worker parallelization scheme and use HMMERCELL as a validation example (Chapter 4). Besides, we evaluate HMMERCELL performance in much more detail. We have manually instrumented the code to obtain runtime traces have thus broken down performance into its constituent phases. These are then modeled and profiled in order to analyze their behavior for various workloads. Bottlenecks to scalability are shown and their impact is evaluated. Furthermore, in Chapter 4, HMMERCELL is mapped onto a larger multicore architecture (SARC [75]) to study performance scalability. After bottlenecks are identified, two optimized versions are proposed and evaluated.

HMMER has been ported to various architectures. In [70], an FPGA implementation of HMMER is investigated. As in HMMERCELL, the computationally intensive kernel of the Viterbi algorithm is the main focus. Similar to HMMERCELL, the FPGA is used as a filter: the sequences with a promising score require reprocessing on the host machine. A thirty fold speedup over an AMD Athlon64 3500+ is reported. This result is comparable to the performance of HMMERCELL.

Further parallel versions of HMMER for different architectures have been proposed by researchers. MPI-HMMER was created to take advantage of computer clusters [96]. Similar to HMMERCELL, one node is assigned a manager role and the rest of the machines are workers over which the workload is distributed. To cope with overhead from message passing, sequences are grouped in larger bundles and sent as one message. Through double buffering, communication latency is minimized. An eleven-fold speedup is reported when using sixteen machines. In [97], MPI-HMMER is analyzed and found to be scalable up to 32-64 nodes, depending on workload. PIO-HMMER is introduced, addressing I/O-related bottlenecks through the use of parallel I/O and optimized post-processing. The manager distributes an offset file with sequences to each node and they read the sequences from their local database. Furthermore, nodes only report significant results back to the manager. The resulting scaling capability is much improved, as up to 256 machines can be used effectively. Other authors have parallelized HMMER *hmmpfam* kernel for shared-memory machines [89] and for computer clusters in HSP-HMMER [76], using MPI. In [54], HMMER is reported to scale up to 1024 processors on a Blue Gene/L system. Although that paper does not report kernel performance (GCUPS), we can safely assume it is significantly slower. The Blue Gene/L platform uses

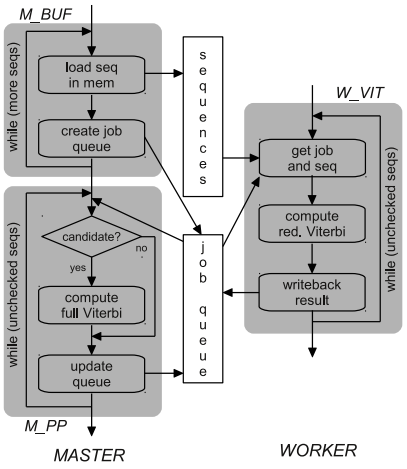


Figure 3.5: HMMERCELL program phases.

PowerPC 440 cores that, compared to the Cell SPEs [55] we use, do not support SIMD processing and run at much lower clock frequency rates. Having a superior performance per worker, poses a higher pressure on the shared resources like the master processor and thus the scalability is more difficult to achieve. The aim of our work is to study the parallel performance of HMMER for on-chip multicores with high throughput workers (Chapter 6). Furthermore, the use of on-chip multicores provides great energy savings compared to discrete multiprocessors and clusters.

HMMERCELL program structure

Since the HMMERCELL execution time is almost exclusively formed by the execution of the Viterbi function, the parallelization strategy focuses on this function. In order to optimally utilize the Cell BE architecture, a few key techniques have been used. First of all, parallelism is used at two levels: coarse-grain parallelism by spawning SPE threads and fine-grain parallelism within the SPEs, by using a highly efficient SIMDized version of the Viterbi algorithm [62]. Secondly, due to the small SPE Local Store, the use of small memory footprint version of the Viterbi algorithm is required. Hence, SPEs do not provide a full alignment but only produce an alignment score. High scoring alignments are reprocessed on the PPE to obtain the actual alignment.

In Figure 3.5, an overview is given of the HMMERCELL internal functioning. The PPE and the SPEs assume the master and worker rolls respectively in our

case study. The important phases are:

- **M-BUF:** the master buffers the protein sequences by loading them from disk to main memory and creates tasks for the workers by adding entries in a job queue.
- **W-VIT:** once a worker gets a job from the queue, it copies the assigned protein from main memory to its LS, performs the reduced Viterbi algorithm and writes the result score back to main memory.
- **M-PP:** during the post-processing phase, the master runs the full Viterbi algorithm to recover the alignment of proteins that have scored above the threshold.

For the sake of clarity, Figure 3.5 does not show the loading of the HMM as this is done only once, at the beginning, and therefore negligible.

3.4 Summary

This chapter is divided into three parts. The first one discussed the multicore architectures, focusing on the two studied in this thesis: Cell BE and SARC. The second part described the two simulators used for the instruction-set extensions and the manycore experiments: CellSim and TaskSim respectively. The last part of this chapter has discussed the related work. It was divided in two subsections according to the applications used in this thesis: ClustalW and HMMER. We mostly discussed thread-level parallelizations but also included SIMD and FPGA implementations.

4

Performance Estimation Model

In sequence alignment tools there is always a trade-off between sensitivity and speed. For instance, the Smith-Waterman algorithm [87] finds the optimal alignment between two sequences but it is impractical in some scenarios and people have to use BLAST [8] instead. As discussed in Chapter 2, HMMER [7] uses a different (probabilistic) approach to align sequences that aims at achieving high sensitivity. Furthermore, it has been recently improved and it is claimed [7] to already achieve the same performance as BLAST.

In this chapter we analyze a parallel version of HMMER and propose a performance model that captures the program behavior. This model helps us to understand the program structure, reveal the bottlenecks and compute the maximum effectively usable cores for a given workload. We also present profiling results of real executions that reveal the scaling capabilities in number of cores and input data size.

Advancements in microprocessor technology have resulted in continuously increasing computational power, through miniaturization and growing transistor budgets. However, single threaded performance growth is stagnating because of frequency, power and memory scaling issues. These "walls" are the reason for the current paradigm shift towards multicore architectures. This shift attempts to provide the expected growth in processing capabilities. One example of such multicore architecture is the Cell BE, a processor with special architectural components and organization that has opened a new path in processor design. In this chapter we have used Cell as a case study to validate our proposal, however, our analysis is ultimately more general.

The suitability and effectiveness of the multicore paradigm for bioinformatics applications is still an open research question. In this chapter, we develop an analytical model of a *master-worker* parallelization of HMMER for multicore architectures. As a case-study, we use HMMERCELL (a port of HMMER to

the Cell architecture) to investigate its scalability behavior. Through profiling on the Cell processor we set the coefficients and finally we are able to validate our model. In essence, main contributions of this chapter are:

- A performance prediction model that is shown to be highly accurate (with error within 2%);
- A detailed, quantitative characterization of program phases and their influence on overall performance;
- A careful study of the scalability bottlenecks.

The remaining of the chapter is organized as follows. In Section 4.1 we recall the HMMERCELL phases and validate our performance model. Section 4.2 describes the experimental methodology used for profiling and for the model construction and validation. Section 4.3 shows the profiling results. Section 4.4 discusses this chapter findings and in Section 4.5 we draw a summary of the work.

4.1 HMMER Analytical Model

Here we present an analytical model that estimates the total execution time of a HMMER parallel version that uses the master-worker paradigm. Based on theoretical expectations and code inspection, we model the required time for each program phase separately and then combine these phases together. This results in an accurate model for HMMER performance on multicore platforms.

First, we recall the HMMERCELL phases and derive the different functions of the model. Then, the model is applied to our implementation platform, the Cell Broadband Engine, to define the numeric values of the different constants. The analytical results are validated and used to show how the model can be used to derive the maximum effectively usable SPE count. More information on the program phases and the profiling results are presented in Section 4.3.

4.1.1 HMMERCELL Program Phases

For the sake of clarity, we recall here the main HMMERCELL program phases that are explained in more detail in Section 3.3.2 and shown in Figure 3.5. The PPE and the SPEs assume the master and worker rolls respectively and the main phases are:

- **M-BUF:** the master buffers the protein sequences by loading them from disk to main memory and creates tasks for the workers by adding entries in a job queue.
- **W-VIT:** once a worker gets a job from the queue, it copies the assigned protein from main memory to its LS, performs the reduced Viterbi algorithm and writes the result score back to main memory.
- **M-PP:** during the post-processing phase, the master runs the full Viterbi algorithm to recover the alignment of proteins that have scored above the threshold.

4.1.2 Model Derivation

The following parameters are used in our HMMER model:

- T_M, T_W: master-worker processor time;
- T_{M_BUF}, T_{M_PP}, T_{W_VIT}: execution time of phases;
- l_i: length of a specific sequence;
- $\bar{l}$: average length of sequences in the test set;
- m: length of the profile HMM H;
- n: number of sequences in the test set S;
- P_{PP}: chance for protein sequence to score above the threshold and thus requiring post-processing on the master;
- q: number of workers used;
- α, β, γ, δ, C_α, C_β, C_γ, C_δ: model coefficients.

The required time (t) for each phase to process a single sequence is expressed in Equations 4.1-4.3 and is based upon expectations from theory and program inspection. Function I_{PP} acts as an indicator, returning 1 when an alignment between a sequence s_i and the model H is significant for a test set of size n and otherwise returning 0. Such a sequence requires post-processing on the master node, which in our case means recomputing the alignment using the full Viterbi algorithm on the PPE.

$$t_{M\_BUF} = \alpha \cdot l_i + C_\alpha \quad (4.1)$$

$$t_{W\_VIT} = \beta \cdot m \cdot l_i + C_\beta \quad (4.2)$$

$$t_{M\_PP} = (\gamma \cdot m \cdot l_i + C_\gamma) \cdot I_{PP} \quad (4.3)$$

Aggregating these equations for individual sequences to the entire test set (containing n sequences) results in Equations 4.4-4.6. T_{W_VIT} states the time required for the Viterbi computations of all the sequences combined. The indicator function I_{PP} has been replaced by the probability function P_{PP}, giving the average chance for a sequence in test set S to require post-processing. Predicting the result of indicator function I_{PP} is difficult, as it requires knowledge of the biological match between the protein model and a specific sequence. Probability P_{PP} however, can be estimated based on the program theory [6] and the overall *traceback* count of a representative test set. The HMMER algorithm dynamically computes a threshold score for a hit based on the statistical significance that in turns depends on the database size and a parameter known as the *E-value*. The larger the database (n), the higher the probability that a hit occurs just by chance and then it is desirable to increase the threshold as this happens. This behavior is modeled by the decremental characteristic of P_{PP} as defined here.

$$T_{M\_BUF} = n \cdot (\alpha \cdot \bar{l} + C_\alpha) \quad (4.4)$$

$$T_{W\_VIT} = n \cdot (\beta \cdot m \cdot \bar{l} + C_\beta) \quad (4.5)$$

$$T_{M\_PP} = n \cdot (\gamma \cdot m \cdot \bar{l} + C_\gamma) \cdot P_{PP} \quad (4.6)$$

$$with \quad P_{PP} = \frac{\delta \cdot \ln n + C_\delta}{n}$$

To combine the previous equations into an integrated model of HMMER performance, the interrelation between the functions should be taken into account. The dependencies between these three functions are depicted in Figure 4.1. W_VIT starts after M_BUF, as at least one sequence should be buffered before processing by the workers can commence. W_VIT ends after M_BUF, as the last sequence to be buffered must be processed as well. M_PP starts when M_BUF finishes, as both buffering and post-processing are performed on the master node. M_PP ends after W_VIT, as the last processed sequence must be checked by the master.

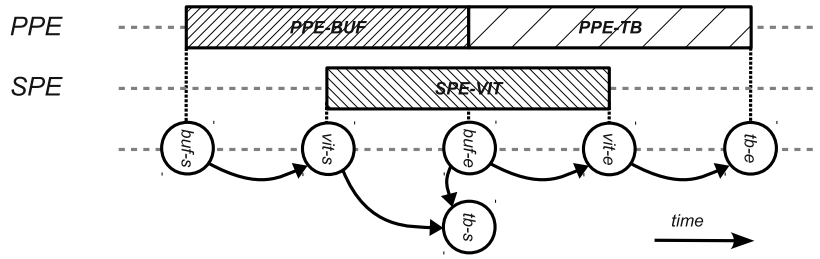


Figure 4.1: Relationship of dependence between functions.

When a test set contains many thousands of sequences, processing time of any individual sequence is insignificant when compared to total execution time. This observation allows for two simplifications: first, the above dependencies between functions can be approximated as follows: M_BUF and W_VIT can be assumed to start at the same time, M_PP starts when M_BUF completes, and M_PP must finish after W_VIT. The model also assumes that when M_BUF finishes there are *hits* already available for it to post-process. This is reasonable considering the M_BUF long latencies and because *hits* will usually be randomly distributed in the database.

On the other hand, load balancing between workers is assumed to be perfect, as all processes will finish at approximately the same time. This approximation and hence the accuracy of the model relies on the assumption that the test set contains a large number of sequences, so that the granularity of individual sequence processing becomes very small. This is reasonable, for example a relevant workload such as the UniProt database contains around half a million sequences. Using these assumptions, execution time is modeled as per Equations 4.7-4.9:

$$T_M = T_{M\_BUF} + T_{M\_PP} \tag{4.7}$$

$$T_W = (T_{W\_VIT})/q \tag{4.8}$$

$$T_{TOTAL} = \max(T_M, T_W) \tag{4.9}$$

4.1.3 Model Parametrization

The previous section shows the generalized form of a performance model for an application parallelized using the master-worker paradigm. The coefficients α-δ, C_α-C_δ and function P_{PP} are specific to the actual implementation of HMMER. Here, we show the actual values for our implementation on the Cell BE

architecture. Using linear and logarithmic regression, the parametrized values are derived from the profiling results. The extra processing time required by reprocessing high scoring sequences on the master node is incorporated in the coefficient's values.

$$T_{M\_BUF} = n \cdot \left(\frac{0.19}{10^3} \cdot \bar{l} + \frac{5.52}{10^3}\right) \tag{4.10}$$

$$T_{W\_VIT} = n \cdot \left(\frac{0.59}{10^3} \cdot \frac{m}{10^2} \cdot \bar{l} + \frac{0.88}{10^3}\right) \tag{4.11}$$

$$T_{M\_PP} = n \cdot \left(\frac{2.25}{10^3} \cdot \frac{m}{10^2} \cdot \bar{l} + \frac{35.7}{10^3}\right) \cdot P_{PP} \tag{4.12}$$

$$with \quad P_{PP} = \frac{21.9 \cdot \ln n - 73.2}{n}$$

Combining Equations 4.7-4.9 and 4.10-4.12, total execution time is approximated by:

$$\max \begin{cases} T_M \Rightarrow n \cdot [(\frac{0,19}{10^3} + \frac{2,25}{10^3} \cdot \frac{m}{10^2} \cdot P_{PP}) \cdot \bar{l} \\ \qquad\qquad +(\frac{5,52}{10^3} + \frac{35,7}{10^3} \cdot P_{PP})] \\ T_W \Rightarrow n \cdot (\frac{0,59}{10^3} \cdot \frac{m}{10^2} \cdot \bar{l} + \frac{0,88}{10^3})/q \end{cases} \tag{4.13}$$

4.1.4 Maximum Effective SPE Count

Equation 4.13 can be used to derive the number of workers (or in the case of Cell: SPEs) that can be effectively used in scenarios that are constrained by the master's buffering performance (in our case: the PPE). In such situations, the number of workers that will saturate the master's buffering capability can be estimated by setting T_{M_BUF} equal to T_W, which results in the maximum effective number of workers q:

$$q \approx \frac{T_{W\_VIT}}{T_{M\_BUF}} = \frac{0,59 \cdot \frac{m}{10^2} \cdot \bar{l} + 0,88}{0,19 \cdot \bar{l} + 5,52} \tag{4.14}$$

From this equation, it follows that the number of usable workers is solely dependent on HMM model size. Table 4.1 gives the maximum number of usable workers for various HMM sizes when using sequences with typical length. Profiling results in Section 4.3 confirm the data from Table 4.1.

Table 4.1: Maximum effectively usable workers.

HMM Length	100	200	300	400	500
q (max worker count)	3	6	9	12	15

4.1.5 Model Validation

To validate our model, additional tests have been performed with new randomly selected data sets of 20.000 and 40.000 sequences (the size is constrained to fit in our blade user quota, but large enough to be significant for the experiments). These test sets have been checked to have an average sequence length near 355 symbols in order to ensure the same behavior as with the full Siwss-Prot database. Sequences are compared against four different HMMs with length 150 and 450 (two representative lengths as seen in Figure 4.2). The execution time of each of the HMMER phases is shown in Table 4.2 for both the empirical execution and the model prediction. The last two columns show the percentage error between prediction and estimation.

Our model was able to accurately estimate the execution time of M_BUF and W_VIT, average deviation between result and expectation was 1.5% and 1.7% respectively. On the contrary, the estimation for M_PP was unreliable, as the number of sequences that require post-processing depends on the biological fit between data set and the HMM, and because the time for post-processing varies considerably for each sequence. However, the M_PP model inaccuracy will only affect overall performance estimation if the application is constrained by the M_PP phase, which only occurs if a high fraction of sequences requires post-processing. However, as *traceback* count scales logarithmically in test set size, this fraction is marginal for realistic test sizes. Furthermore, for shared memory architectures where the M_PP phase does not need to compute the full Viterbi algorithm, the significance of the phase is even less than in our case-study. Thus, M_PP contribution to total execution time is negligible. Overall, the average error of our model was below 2%.

4.2 Experimental Methodology

Experiments are performed on an IBM QS21 Blade featuring two Cell processors (and hence 16 SPEs) running at 3.2GHz and having 4GB of RAM. The

Table 4.2: Validation results.

Test	Empirical Results			Model Results			Diff	
n, m, q	BUF	VIT	TB	BUF	VIT	TB	BUF	VIT
20k, 150a, 1	1498	6349	581	1459	6301	176	-2.6%	-0.8%
20k, 150a, 8	1492	797	581	1459	788	176	-2.2%	-1.2%
20k, 150b, 1	1442	6345	336	1459	6301	176	1.2%	-0.7%
20k, 150b, 8	1492	797	335	1459	788	176	-2.2%	-1.2%
20k, 450a, 1	1441	18440	777	1459	18868	519	1.3%	2.3%
20k, 450a, 8	1448	2303	776	1459	2359	519	0.8%	2.4%
20k, 450b, 1	1441	18436	1032	1459	18868	519	1.3%	2.3%
20k, 450b, 8	1446	2305	1031	1459	2359	519	0.9%	2.3%
40k, 150a, 1	3071	12747	1031	2934	12673	196	-4.7%	-0.6%
40k, 150a, 8	3023	1605	714	2934	1584	196	-3.1%	-1.3%
40k, 150b, 1	2927	12748	427	2934	12673	196	0.2%	-0.6%
40k, 150b, 8	3026	1603	427	2934	1584	196	-3.1%	-1.2%
40k, 450a, 1	2925	37021	509	2934	37949	577	0.3%	2.4%
40k, 450a, 8	2931	4621	507	2934	4744	577	0.1%	2.5%
40k, 450b, 1	2924	37021	1531	2934	37949	577	0.3%	2.4%
40k, 450b, 8	2929	4629	1551	2934	4744	577	0.2%	2.4%

code has been compiled with GCC4.1.1 and -O3 flag. Only one PPE was used in our experiments as we intend to study scalability in the number of SPEs.

For the profiling and model validation tests, profile HMMs from the Pfam database [88] and sequence data sets from the UniProtKB/Swiss-Prot database [18] were used. Figure 4.2 shows the current model and sequence length distribution for Pfam and Swiss-Prot databases. HMM queries of 5 different sizes (100 to 500) have been selected from Pfam. Protein test sets of 20000 and 40000 sequences have been chosen at random from Swiss-Prot. We consider these test sets representative as we verified that the size distribution of the samples matched the one of the complete database (shown in Figure **??**). The statistical significance of the test sets is also expected since the samples represent approximately 5% and 10% of the database size. For the validation experiments, newly random protein test sets were created following the same methodology.

Profiling results were obtained by analyzing runtime traces from an instrumented version of the application. HMMERCELL was manually instrumented using Extrae tracing library [17]. The generated traces have been inspected with Paraver [13], a visualization environment for trace files. To model application behavior, parametric functions for each phase were created. Their

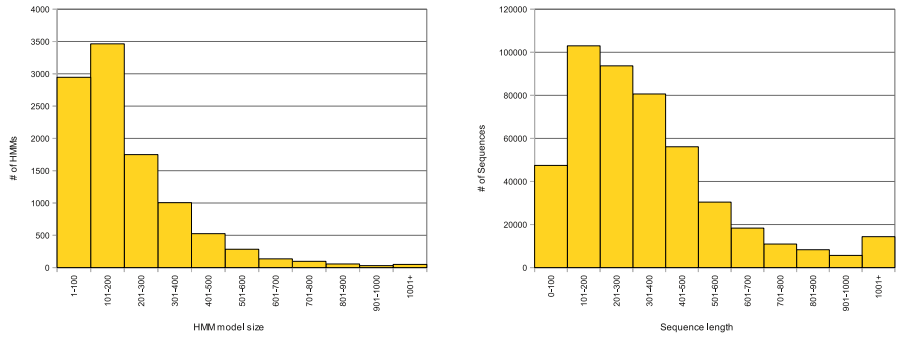


Figure 4.2: Distribution of protein database element sizes.

dependence on the input and the choice for linear or logarithmic scaling depends on theory, profiling results and inspection of the HMMER source code. Based on these equations, the formulas were parametrized by fitting the profiled performance data, using linear or logarithmic regression.

4.3 Cell BE Profiling Results

This section complements the HMMER scalability analysis by presenting profiling analysis. Besides reporting performance scalability on the number of workers, results also show scaling behavior with respect to input sizes. Profiling results were obtained using 5 distinct HMMs with lengths from 100 to 500 positions. For general performance tests, two sequence sets consisting of 20000 and 40000 randomly selected sequences with length distribution identical to the UniProt database were used. Figure 4.3 gives an overview of HMMERCELL performance where some basic characteristics on how HMMERCELL reacts to changes in input parameters are revealed: the use of a longer HMM model size requires correspondingly longer execution time; in general, the use of additional SPEs leads to shorter execution times; and only a certain number of SPEs can be used effectively, depending on the workload. Due to management overhead, using more SPEs results in identical or even deteriorated performance.

In order to evaluate the scaling capability of HMMERCELL, the behavior of each of the important program phases (**M_BUF, W_VIT, M_PP**) was analyzed in isolation. The goal of these experiments is to understand the behavior of each phase, their dependence on the input parameters, how they contribute to

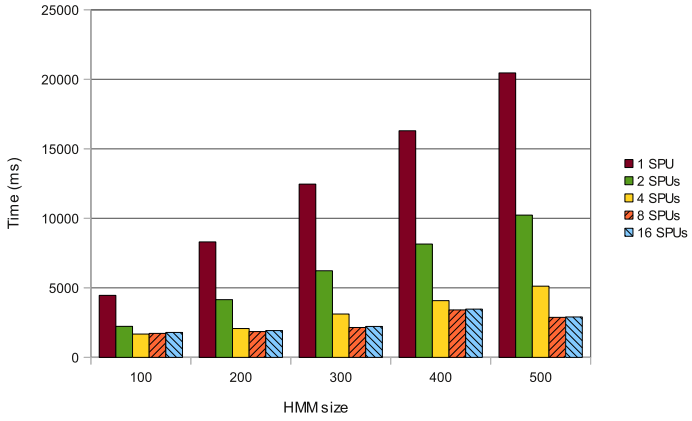


Figure 4.3: HMMERCELL execution time overview.

aggregated HMMERCELL performance, and to understand the role and impact of various bottlenecks to scaling capability. In the following subsections, profiling results are discussed for each phase, showing their scaling behavior in HMM size and sequence length. Each phase has two plots showing its scaling behavior. The figures on the left show the execution time scaling with respect to sequence length and for various HMM sizes (i.e. sequence length on the horizontal axis, lines represent different HMM sizes). The plots on the right show the execution time scaling with respect to the HMM model length (HMM size on the horizontal axis, lines represent different sequence lengths). The vertical axes represent execution time except in Figure 4.7 where it is number of tracebacks.

4.3.1 The PPE Buffering Phase

In Figure 4.4, the scaling behavior of the M_BUF phase is shown. This phase of the program, which runs on the PPE, is responsible for loading sequences from disk into main memory, for converting them to HMMER's internal format and for creating jobs (for the SPEs) by adding the corresponding entries to the job queue. From the graphs in the figure, it is clear that the M_BUF computation time scales linearly in the sequence length and is independent of the HMM model size. This is in-line with expectations: loading a single HMM (even if it is a long one) takes negligible time compared to loading the many database sequences. In fact, when we talk about the M_BUF phase we

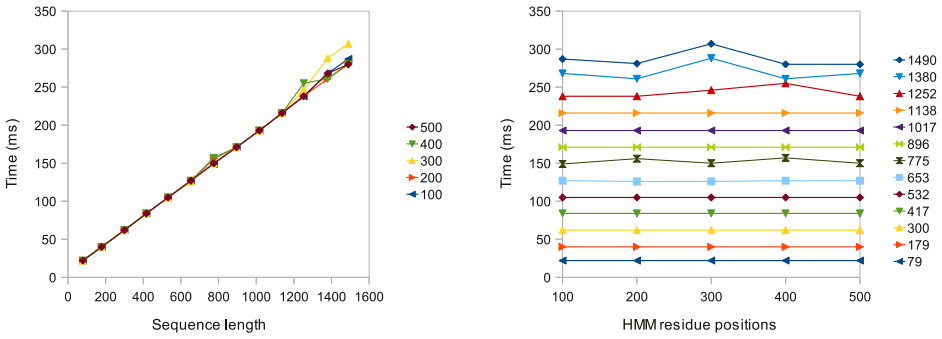


Figure 4.4: Scaling characteristics of the PPE buffering phase (M_BUF) parametrized for the HMM size and the sequence size in the left and right figures respectively.

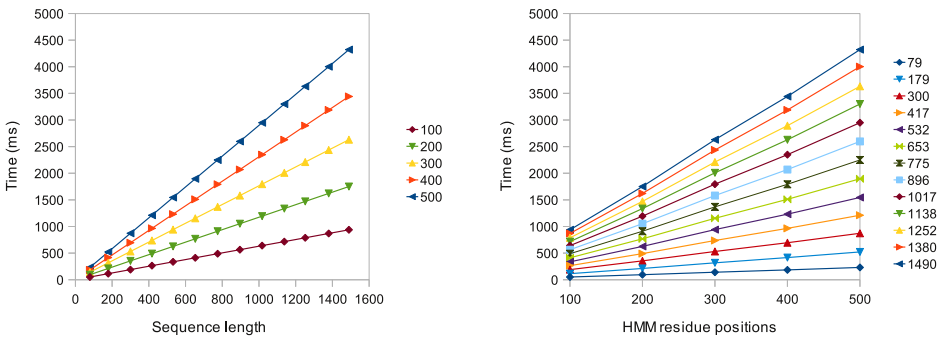


Figure 4.5: Scaling characteristics of the SPE Viterbi phase (W_VIT) parametrized for the HMM size and the sequence size in the left and right figures respectively.

discard the loading of the HMM. More in-depth profiling revealed that 40% of M_BUF time is spent on I/O while most of the rest is spent on formatting the sequences before they can be processed. Therefore, an increased I/O bandwidth and a faster formatting of sequences (either by parallelization or by a faster processor) would be the way of speeding this phase up.

4.3.2 The SPE Viterbi Phase

Figure 4.5 shows the scaling behavior of the most computationally intensive part, the W_VIT. During this phase, the SPEs process the PPE-created jobs

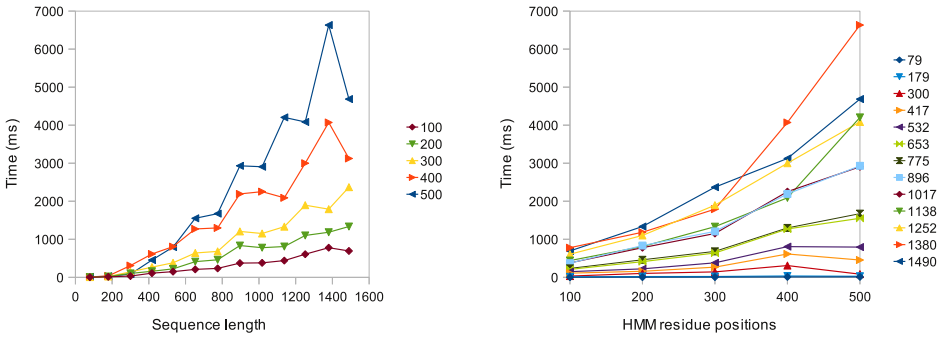


Figure 4.6: Scaling characteristics of the PPE Viterbi Traceback phase (M_PP) parametrized for the HMM size and the sequence size in the left and right figures respectively.

in M_BUF. In each job, a sequence is aligned to the HMM using the Viterbi algorithm. A special version of the algorithm with smaller memory footprint is used so that all data structures fit inside the small SPEs' LS (256KB). In this version, intermediate values are discarded and only the alignment score is produced, which is sent back to the PPE.

From the figures, it is clear that W_VIT computation time scales linearly both in the length of the sequence and in the size of the HMM profile. Again, this confirms expectations, as the Viterbi algorithm scales linearly in sequence length and linear for models cast in profile HMM form.

4.3.3 The PPE Traceback Phase

Figure 4.6 depicts the M_PP scaling behavior. This phase checks the results that have been produced by the SPEs and performs the full Viterbi algorithm on the PPE for those sequences that have a high alignment score and hence might form a potential match to the model. Compared to the previous phases, M_PP behavior is less regular. The reason for this is that whereas M_BUF and W_VIT are performed for each sequence in the test set, M_PP only performs the Viterbi calculations for a subset of sequences, namely those whose alignment score exceeds a certain threshold. The actual number of tracebacks depends on the underlying biological semantics, i.e. how many sequences in the test set fit well to the model.

Behavior of M_PP is further analyzed by breaking it down in two components:

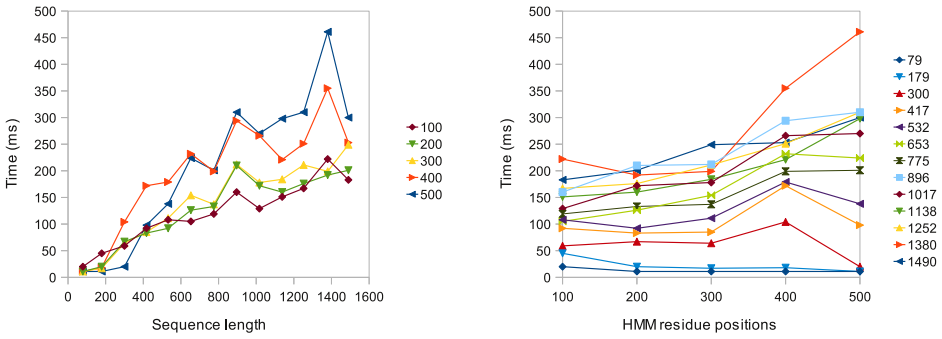


Figure 4.7: M_PP analysis: total number of tracebacks, parametrized for the HMM size and the sequence size in the left and right figures respectively.

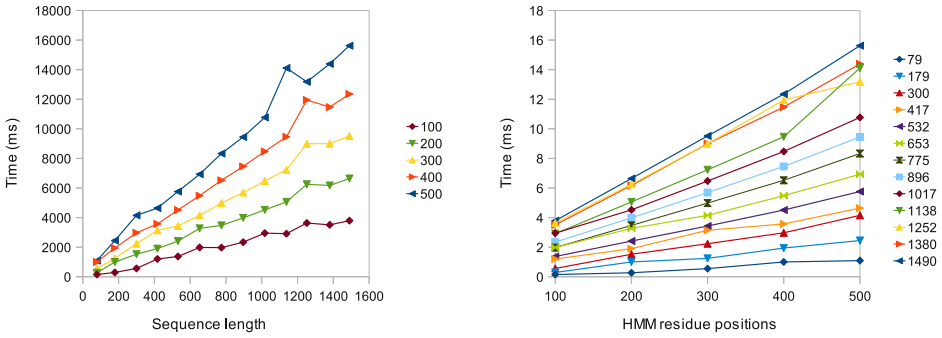


Figure 4.8: M_PP analysis: average time per individual traceback, parametrized for the HMM size and the sequence size in the left and right figures respectively.

the number of tracebacks performed for a test set (Figure 4.7) and the average time required for an individual traceback (Figure 4.8), given different sequence and HMM combinations. Of course, when the number of tracebacks is multiplied by the time per traceback, the total time spent in the traceback phase is produced.

Of these two components, the results for average time per individual traceback are as expected: execution time for a single sequence scales more or less linearly in both sequence length and HMM size. The full Viterbi algorithm requires a large data structure in memory and traversing this memory hierarchy is the reason for the observed staggered scaling. Hence, the erratic results

in total traceback time are mainly caused by the fluctuations in the number of tracebacks. Some correlation between length and traceback count can be observed: generally speaking, longer sequences result in more hits, since local alignment is performed. Subsections of a sequence are allowed to form a match to the model, hence the longer the sequence the larger the probability of a matching subsection. HMM size has no clear effect on performance.

The number of tracebacks is affected by two factors: the particular combination of HMM and sequence set, i.e. their biological match; and the number of sequences in the test set, as the E-value depends on the test set size. The number of tracebacks required varies largely between HMMs, even for those having identical length. Results between different sequence sets of equal size vary much less. An alignment requires a higher bit score to be counted as significant when comparing against a larger database. A logarithmic relationship between test set size and traceback count is present [6].

4.4 Discussion

The *hmmsearch* kernel was parallelized according to the master-worker pattern into three stages: buffering, Viterbi processing and post-processing. The phase that performs the Viterbi calculations is the most time-consuming portion of HMMER and is primarily responsible for overall program behavior. Hence, inspection of this part and its parallelization strategy is very important. Offloading the Viterbi calculations onto the workers is effective: the workload is regular, the computation-to-communication ratio is high, and in theory the number of workers that can be efficiently used is only limited by the number of sequences. However, the master should be able to create jobs fast enough. This implies that for any given workload a certain worker count exists that will saturate the master. In this respect, our model shows that the HMM model size determines how many workers can be used before the master's buffering capability is exceeded. This is explained by the fact that (as seen in Figures 4.4-4.5) only the HMM size has a different impact on M_BUF and W_VIT. For short HMMs for instance, worker jobs are small compared to the M_BUF phase, resulting in the master not being able to keep up with preparing jobs. Notice that by letting the workers format the input sequences themselves would improve scalability as less work needs to be done by the master in the buffering phase (see Chapter 6).

The Cell BE explicit memory architecture with SPEs having a small LS requires the use of a smaller footprint Viterbi algorithm. As a consequence,

the full Viterbi kernel should be included in M_PP. This phase becomes another potential bottleneck and is shown to introduce inherent uncertainty in the model. However, as the PPE buffering phase and the SPE Viterbi phase are both linearly dependent on the number of sequences in the workload, they are the most influential to overall performance. Because M_PP execution time becomes less significant for larger workloads, its impact on the overall model accuracy becomes negligible for realistic test sets. It should be noted however, that for future chips containing more cores, the impact of the M_PP phase will grow unless it is also parallelized in some sort of multi-master strategy. In our case study with the Cell BE, the model was overall found to be highly accurate, with only 2% error when compared to execution on real hardware.

The need for Viterbi in M_PP can be avoided on a shared memory system. That is, W_VIT could perform the Full Viterbi algorithm instead of the reduced version because in such an architecture a core is not limited to compute on a small scratchpad memory. On the other hand, increasing the scratchpad memory sizes could solve the problem for the case of short sequences or HMMs. A significantly larger scratchpad memory is usually impractical due to large amount of silicon needed when it is replicated among the multiple cores and because the increased latency defeats the purpose of having it at all.

A drawback of Cell BE and heterogeneous processors with an explicit memory architecture in general, is that there is a direct impact on the parallelization strategy. The advantage is of course that for suitable applications depending on their compute-intensive nature, performance can be very high. However, the ratio between master and workers has to be balanced for the target application. For HMMERCELL, we found that three SPEs saturate the PPE for typical HMM sizes. The proposed model can be used to estimate the optimal ratio between PPE and SPEs for different workloads. In general, modeling the behavioral characteristics is useful: it is a valuable aid for decision-making during design space exploration as it can show the optimal ratio between job creation and consumption. The proposed model can also be used for runtime scheduling.

4.5 Summary

In this chapter we have presented an analytical model of HMMER aimed at master-worker parallelization schemes. The model has been deduced from program inspection and later compared against execution of HMMERCELL on a real Cell BE processor. The model and the profiling results have given

us deep insight in the HMMER scalability details. The model prediction for M_BUF and W_VIT phases has been found to be highly accurate, with only 1.5% and 1.7% error on average. Although M_PP was not accurately estimated by the model, we have shown that for realistic test cases it does not affect the overall prediction. The total execution time estimation only had 2% error.

The findings in this chapter are relevant for other bioinformatics applications as well. Most bioinformatics applications contain an abundance of coarse-grained parallelism and the master-worker pattern is a useful strategy to divide the work over multiple cores. For optimal scaling behavior, the master core should be relieved of as many other tasks as possible and control tasks should also be parallelized. In the case of a Cell BE blade, the two PPEs offer together four hardware threads that could be used to divide up the M_BUF work. Even better, the SPEs could take care of the sequence formatting work in M_BUF. However, parallelizing M_BUF would only speedup its sequence formatting part and the I/O bottleneck would still remain.

Although using HMMER and the Cell BE processor for the experiments, the study presented in this chapter has a more general scope. Our ultimate goal is to understand the interaction between bioinformatics workloads and heterogeneous multicore architectures.

Note. The content of this chapter is based on the the following papers:

S. Isaza, E. Houtgast, G. Gaydadjiev. **Sequence Alignment Application Model for Multi- and Manycore Architectures**. *International Journal on Information Technologies and Security,* ISSN 1313-8251, pp. 3-20, 2011.

S. Isaza, E. Houtgast, G. Gaydadjiev. **HMMER Performance Model for Multicore Architectures**. *Proceedings of the* 14^{th} *Euromicro Conference on Digital System Design, DSD - IEEE,* pp. 257-261, Oulu, Finland, Aug. 2011.

5

Cell BE Performance Optimizations

The Cell Broadband Engine microprocessor has been one of the first and most prominent examples of the multicore paradigm in the computer architecture field. Two types of processing elements, a specific memory hierarchy and a fast interconnect were incorporated on a single chip to form a radically new heterogeneous multicore. The Cell BE was initially designed to target the game consoles market but due to its novel features, researchers rapidly started using it for other purposes, notably for scientific high-performance computing. With some programming effort, the Cell BE is able to exploit multiple dimensions of parallelism typically found in scientific applications, especially in bioinformatics.

This chapter presents our experiences with mapping and optimizing sequence alignment applications on the Cell BE. Although the multiple types of parallelism (ILP, DLP, TLP) found in these applications make them good candidates to be mapped onto the Cell BE, here we discuss experimental results that reveal the existence of several limitations. While manually mapping the target applications, we identified different types of bottlenecks that affect different aspects of the programs execution on the Cell BE. Special attention is also put on how performance scales with an increasing number of cores and the associated bottlenecks. The main contributions of this chapter are the performance improvements achieved for ClustalW through: fine-grain code optimizations, parallelization to multiple SPEs and adding application-specific instructions to the SPE instruction-set, in the last part of the chapter.

This chapter is organized as follows: Section 5.1 explains the ClustalW implementation on the Cell BE. In Section 5.2 we describe the experimental methodology. Section 5.3 presents the experimental performance results, focusing on the limitations we found when porting the application and on the scalability. Section 5.4 describes the proposed SPE instructions and the performance gains

as measured on the CellSim simulator. Section 5.5 summarizes the chapter.

5.1 ClustalW and its Implementation on the Cell BE

This section introduces the ClustalW workload and discusses specific issues related with its Cell BE implementation. We recall here that our intention is not the development of highly optimized Cell BE specific versions of the targeted applications. Our main focus is on the analysis of the limitations that Cell BE presents at several levels in order to hopefully guide the architecture design of future multicore systems for bioinformatics applications.

5.1.1 ClustalW Analysis

Unlike pairwise sequence alignment (e.g., Ssearch), multiple sequence alignment (MSA) applications like ClustalW build a single alignment from a set of sequences that are expected to have some biological relationship. While for pairwise alignments it is still computationally feasible to produce optimal alignments with dynamic programming algorithms, for MSA it is prohibitive and heuristics must be applied to avoid an explosion in time and space complexity. Recalling from Chapter 2, the time complexity of ClustalW is $O(n^4 + l^2)$, where n is the number of sequences and l their average length. Using a technique called *progressive alignment* [90], ClustalW performs the multiple alignment in three main steps: 1) All-to-all pairwise alignment (PW), 2) Creation of a phylogenetic guide tree (GT), 3) Use of the phylogenetic tree to carry out the progressive multiple alignment (PA).

According to profiling results of the original code, the function that performs the alignments in the first step (PW), i.e. *forward_pass*, consumes most of the running time (around 60% to 80%). This function calculates a similarity score among two sequences implementing a dynamic programming algorithm based on Smith-Waterman. In order to perform the alignment of all-to-all pairs of sequences, *forward_pass* is called $n(n-1)/2$ times. As opposed to the first step, the final alignment step (PA) performs only $n-1$ alignments.

It is important to mention that *forward_pass* iterations are data independent making parallelization very appealing. Furthermore, the main function's loop can be at least partially vectorized to explore data-level parallelism in the same fashion as described elsewhere for Ssearch [84].

The second phase in ClustalW (GT) has been shown [52, 80, 93] to take an

almost negligible amount of time in most cases, certainly in the ones we are interested in, where sequences are in the order of a thousand symbols. Lastly, PA progressively aligns sequences following the tree structure. It walks the tree by aligning the most related sequences first. After traversing the full tree, the MSA is produced along with a global score.

Figure 5.1a shows the basic PA operation. The main function in PA is *prfalign*. It first computes the two 2D arrays that contain the *profiles* and then call the *pdiff* function to align them. *pdiff* computes the alignment with a dynamic programming algorithm [64] that uses two independent loops: one that moves forward (FWD) and one that moves backwards (BWD). The arrow between the two corresponding boxes in Figure 5.1a emphasizes the fact that there are no data dependencies. A third small loop (not shown for simplicity) collects results and a recursive call to *pdiff* follows. Every recursive *pdiff* instance incrementally reduces the loops' scope until a final score is computed. This condition is checked for at the beginning of *pdiff* as shown in Figure 5.1a. *ptracepath* takes care of updating the alignment that will subsequently be aligned to a sequence or to another alignment (by further calls to *prfalign* and *pdiff*), by walking the tree nodes.

5.1.2 Cell BE Implementation

Here we describe the Cell BE port of the two ClustalW phases that consume most time, i.e., PW and PA. While in PW there is a lot of very regular coarse-grain parallelism, in PA parallelism is much more limited at the coarse-grain level and, therefore, our implementation tries to exploit a more finer granularity.

We ported the *forward_pass* (PW) Altivec code to the SPE instruction-set and implemented a number of optimizations. DMA transfers are used to exchange data between main memory and the SPEs LS. Saturated addition and maximum instructions were replaced with 9 and 2 SPE instructions respectively. The first optimization consisted in reducing the precision from 32-bit to 16-bit so that twice as many sequence residues could be packed in a single SIMD register for processing. In principle, this allows us to double the computation throughput but the requirement of an overflow check in software causes some overhead. This precision reduction makes sense because the residues are encoded with just a few bits (2 for nucleotides and 5 for amino acids) and because the temporary scores fit within 16 bits for most of the iterations. The overflow check makes sure that the program rolls back to 32 bit precision when needed so as to ensure the program remains semantically correct.

Inside the inner loop of the kernel there are instructions responsible for loading the sequence elements to be compared and using them to index a matrix that provides the comparison score. This is a random scalar memory access that is performed within a loop also containing a complex branch for checking boundary conditions. This type of operations are very inefficient in the SPEs. We have unrolled this loop and manually evaluated the boundary conditions outside the inner loop. Section 5.3.1 discusses the impact of these optimizations.

ClustalW-PW also exhibits coarse-grain parallelism. Every sequence-to-sequence alignment can be isolated as a job that is performed individually by an SPE. The PPE distributes the jobs and waits to collect the results. A first such a version was implemented using a simple round-robin strategy for load distribution. Since some alignments take much longer than others (due to the sequence lengths), this strategy is inefficient as it creates load unbalancing. A second strategy uses a table of flags that SPEs can raise to indicate idleness. This way the PPE can make better decisions on where to allocate the tasks. As explained in the previous section, the parallelization of *forward_pass* in multiple threads does not have dependencies and therefore there is no need to optimize much the load balancing nor the communication efficiency. Section 5.3.1 shows the scalability results of this strategy.

PA has a less straightforward parallelism, as shown in Figure 5.1b. The FWD and BWD loops are independent (see Figure 5.1a) so we concurrently compute them using two SPEs, as in [93]. Figure 5.1b shows the basic interaction between the PPE and one SPE where only FWD is indicated. An equivalent pair of threads (one in the PPE and one in the SPE) computes BWD in the same way.

We noticed that the *prfscore* calls inside FWD/BWD can be fully computed in advance as they only depend on the input *profiles* that do not change inside *pdiff*. This has two benefits. On one hand, it allows for a relatively easy distribution of work among all available SPEs. On the other one, *prfscore* results can be stored in a 2D array that is to be used by all recursive instances of *pdiff*. This saves the unnecessary repeated *prfscore* calls in the original algorithm that was aimed at reducing the amount of memory needed. In consequence, all *prfscore* calls are precomputed by an arbitrary number of SPEs. A *prfscore matrix* is computed and then used as input for FWD and BWD. That is, in the inner body, FWD and BWD do not compute the scores anymore but rather get them from main memory. These DMA transfers happen once for every row and are double-buffered. Therefore, the *prfscore* stage will use N SPEs while

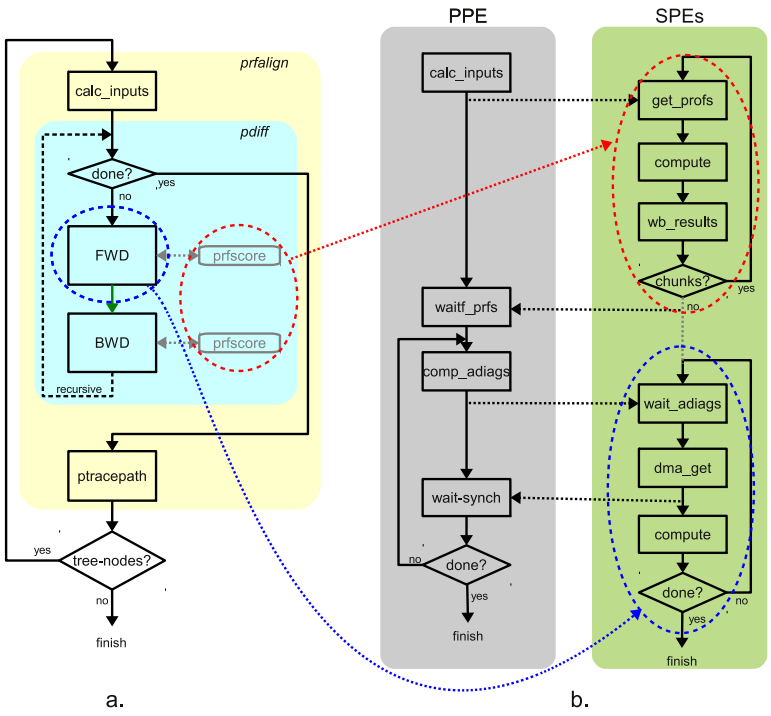


Figure 5.1: PA workflow. a.) Baseline sequential version. b.) Parallel version showing the PPE and one of the SPEs.

in the FWD/BWD stage only two will be running.

Although not shown in Figure 5.1, a check is done on the *profiles*' sizes before running FWD/BWD. This is to avoid sending too tiny pieces of FWD/BWD work when inside recursive *pdiff* instances.

Figure 5.2 illustrates the computation of the *prfscore matrix*. The inputs are two 2-dimensional arrays called *profiles* (A and B in Figure 5.2) where each one represents either a sequence or the result of a previous alignment. The *prfscore matrix* will have *M* rows and *N* columns where position "*i,j*" is computed as the dot-product between rows *i* and *j* of the two input arrays respectively. This work is then performed by the SPEs together. *Profile A* is split in sections that are assigned to the SPEs in a round-robin fashion. *Profile B* has to be divided in chunks due to the LS size limitation. While processing one chunk, the next one is being fetched to hide the transfer latency.

Since the SPE FWD/BWD code uses SIMD and because of the data dependencies (like in Smith-Waterman), the *prfscore matrix* needs to be reorganized

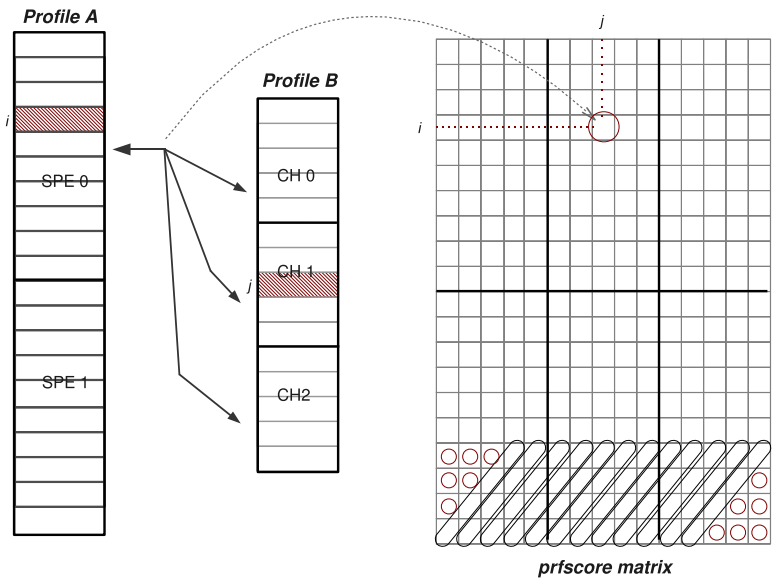


Figure 5.2: Generation of the *prfscore matrix* by dot-products of all rows in the two input *profiles*.

in a way that elements of 4-wide anti-diagonals form vectors, as shown in the lower part of Figure 5.2. For that purpose, we split the matrix in two parts (one that concerns FWD and one for BWD) that are concurrently reorganized by two PPE threads. These also handle synchronization with their corresponding consumer SPE threads. Furthermore, the corner elements that do not belong to any 4-wide antidiagonal are fetched one-by-one from the SPEs. The two PPE threads benefit from the SMT capabilities.

5.2 Experimental Setup

In this chapter we used ClustalW v1.83, written in C. As a starting point, we selected Altivec-SIMD implementations [31] of the most intensive kernel in the program and ported it to the SPE instruction-set. A thread-level parallelization scheme was also implemented using *libspe2*. The experiments were performed on an IBM QS21 Blade featuring two Cell BE processors running at 3.2GHz and 4GB of RAM. The source code has been manually instrumented to measure the time that various processing or waiting phases take.

We have used the default application parameters and the inputs were taken

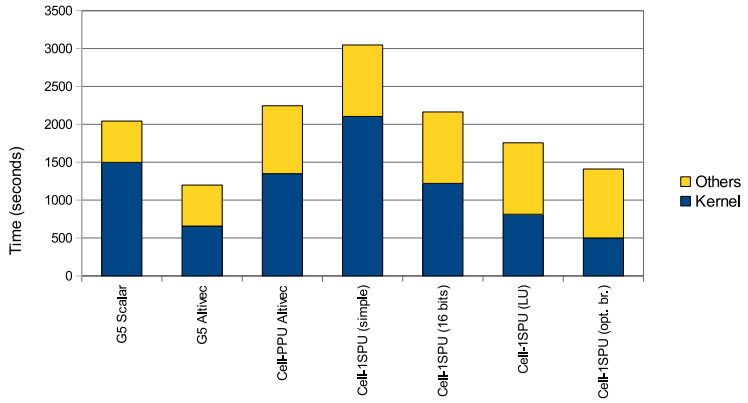


Figure 5.3: ClustalW performance for different platforms and optimizations.

from the BioPerf benchmark suite [22]. The code segments running on the PPE were compiled with ppu-gcc 4.1.1 with -O3 -maltivec options. The code in the SPE side was compiled using spu-gcc with -O3 option.

5.3 Results

This section presents the experimental results. The analysis is divided into pairwise alignment (ClustalW-PW), progressive alignment (ClustalW-PA) and overall application's performance. The last part of this section makes a qualitative discussion on the Cell BE limitations we found. The results corresponding to the instruction-set extensions are presented in Section 5.4.

5.3.1 Pairwise Alignment

In this section we show experimental results of the ClustalW-PW execution, the ClustalW phase that performs pairwise alignments. The results show the impact of a number of optimizations and the performance scalability with an increasing number of cores. We discuss the optimizations applied, the scalability achieved and several issues related to the SPE instruction-set.

Figure 5.3 shows a comparison of ClustalW running on various single-core platforms as compared to different versions using a single SPE. The first two bars show the execution on a PowerPC 970 (also known as G5) processor. It is

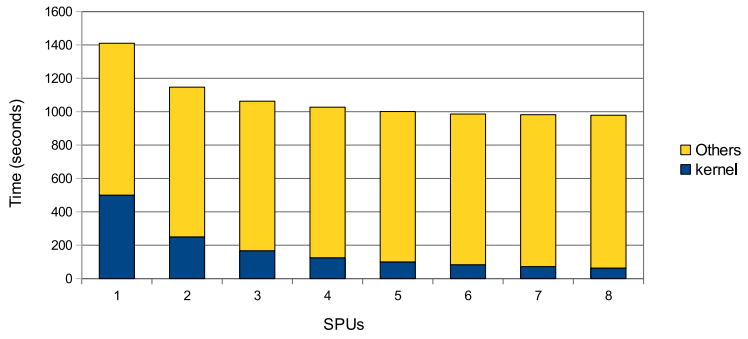


Figure 5.4: ClustalW speedup using multiple SPEs.

an out-of-order superscalar processor that runs at 1.4GHz and supports Altivec instructions. Since the clock frequency of the G5 is more than twice as low as the Cell BE, it is clear that in terms of cycles it outperforms any *Cell 1SPE* version. This is mostly due to the G5's superior ability (larger cache, out-of-order, branch prediction, etc) to execute scalar sections of the code present outside and inside the *forward_pass* function.

The fourth bar shows the straightforward SPE implementation of ClustalW, where only thread creation, DMA transfers and mailboxes are implemented for basic operation and no attention is given to optimizing the kernel code. The fifth bar shows the speedup achieved when reducing the precision from 32 bits to 16 bits. By doing so, we are able to double the throughput as twice as many elements can be packed in a SIMD register. Although for most of the loop iterations this works, the program needs to constantly check if an overflow has occurred (this without any hardware support) so that it rolls back to the 32 bit computation. Overall, a significant speedup (1.7×) is obtained with this optimization.

The next two bars show results for unrolling a small loop present within the inner loop of the kernel. An accumulative 2.6× speedup is achieved. Finally, the last two versions went further into optimizing that small loop by removing the boundary conditions involved in a scalar branch and handling them explicitly outside the loop. This final optimization provided about 4.2× speedup (accumulative) with respect to the initial version.

Figure 5.4 shows the scalability of ClustalW-PW kernel when using multiple SPEs. The black part of the bars reveals a perfect scalability (8× for 8 SPEs). This is due to the relatively low amount of data transferred and the indepen-

dence between every instance of the kernel.

After the successful reduction of the execution time for *forward_pass*, significant application speedups are only possible by accelerating other parts of the program. The progressive alignment phase is now the portion consuming most of the time, discussed in the Section 5.3.2.

5.3.2 Progressive Alignment

Previously we showed how the PW phase is highly scalable so now we focus on the PA part. This section presents a detailed profiling analysis of the different PA phases. We have also produced a scalable parallelization of the *prfscore* function within PA and have merged it with the parallel forward (FWD) and backward (BWD) loops. We also measured and analyzed how the different parts of PA affect the performance when increasing the number of cores and the input data size.

Results presented here include a number of optimizations we applied. Double-buffering is used to copy the input profiles to the local stores. On the PPE side, two threads are created to take care of the forward (FWD) and backward (BWD) loops concurrently. These two threads are supported through SMT in the PPE. The code is also modified by The *prfscore* function is moved earlier in the program execution so that all instances are processed at once and in parallel using all the SPEs.

Table 5.1 indicates the meaning of the different phase names used in the graphs and the text.

prfscore behavior - SPE side

The *prfscore* parallelization requires only minimal synchronization and the PPE just waits until SPEs are done. Therefore, here we analyze the behavior from the SPE perspective only. Figure 5.5 shows the time share of the *prfscore* processing in the SPEs. Looking at the DMA transfers, there are two contrasting results. While fetching the input data (*get-profs*) from main memory appears negligible, the time for writing results back (*wb-results*) is considerable and gets worse with more SPEs. Although both types of DMA transfers are double-buffered, output data is much larger than input data (see Figure 5.2). Moreover, as more SPEs come into play, more data is being transferred simultaneously, thus putting pressure in the communication infrastructure. On the other hand, even when using 16 SPEs, most of the time is spent in performing

Table 5.1: PA phase names used.

Name	Meaning	PPE/SPE
calc-inputs	compute the two input *profiles*	PPE
wait-prfs	wait until the *prfscore matrix* is computed	PPE
comp-adiags	reorganize *prfscore matrix* in antidiagonals	PPE
wait-synch	synchronize reception of antidiagonals	PPE
others	*ptracepath* and other minor phases	PPE
wait-adiag	FWD/BWD waiting for an antidiagonal to be computed	SPE
total-dma	DMA waiting time for corner elements and antidiagonals	SPE
compute	compute the *prfscore matrix* and the FWD/BWD loops	SPE
get-profs	DMA waiting time for input *profiles*	SPE
wb-results	DMA waiting time for matrix data	SPE

useful computations (orange).

Figure 5.6 shows that up to 16 SPEs the overall scalability is relatively good. However, *wb-results* phase will soon become the main bottleneck if adding more cores.

Results shown for the *prfscore* analysis are with 313 sequences as input. Very similar results were obtained with the other input sets. That is, the three *prfscore* phases grow at the same pace with respect to the number of sequences processed. Since we are interested in analyzing scalability when using an increasing number of cores, the baseline for the speedup is the execution time using one SPE (not PPE).

FWD/BWD behavior - SPE side

Unlike for the *prfscore* analysis, for FWD/BWD we use the number of sequences in the input set as the independent variable (X axis in Figures 5.8 and 5.7). The FWD/BWD code in the SPEs is heavily affected by the *comp-adiags* phase in the PPE. As a consequence, only 40%-50% of the time is spent on computing. Around 50% of the time the SPEs are idle waiting for the PPE to reorganize the current row.

Eventhough the corner elements represent only 0.3% of the *prfscore matrix*,

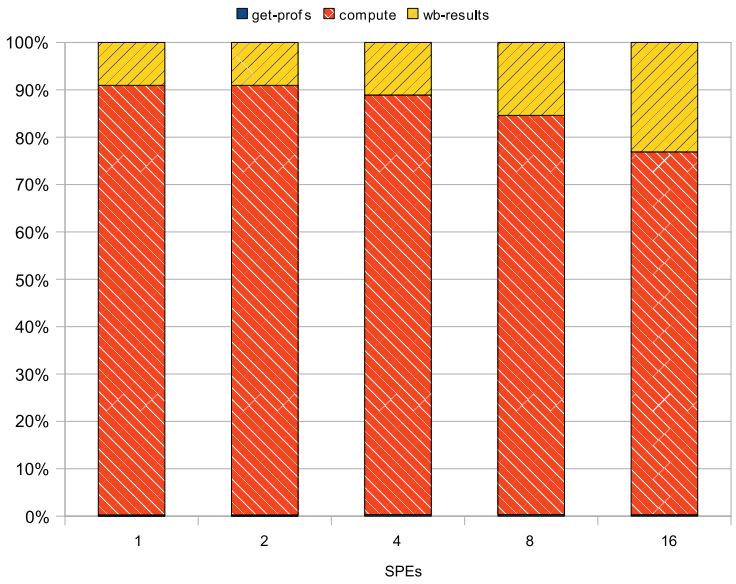


Figure 5.5: Execution time distribution for *prfscore* phases in the SPEs.

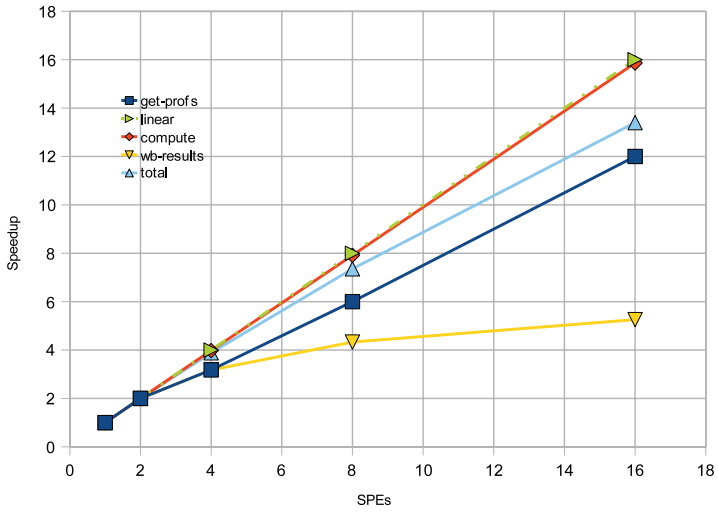


Figure 5.6: Scalability of *prfscore* processing on the SPEs.

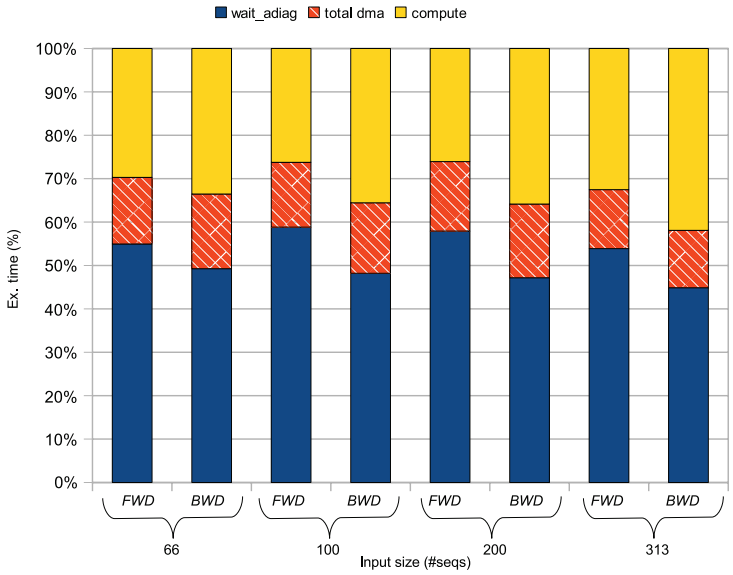


Figure 5.7: Execution time distribution of PA phases in the SPEs.

its associated DMA latency is consuming nearly 15% of the FWD/BWD SPE time. One way to improve this is by flattening corners in vectors so that they can be accessed faster from the SPEs.

Lastly, Figure 5.8 shows that increasing the number of sequences does not significantly deviate the growth lines from the linear behavior.

PA behavior - PPE side

Now we look at the PPE side behavior with respect to the SPE phases already discussed. In Figure 5.9 the time share reveals interesting results. After efficiently parallelizing *prfscore*, the portion it takes now is very small. While the SPEs are busy with FWD/BWD, the PPE spends the time reorganizing the *prfscore matrix*. The most important observation here is that when using the largest input size, the preparation of inputs becomes the most time consuming part. Besides that, new phases become significant.

In Figure 5.10 we show how fast is the growth of the execution time for the PA phases when the input size increases. Two phases do not follow a linear growth. Moreover, *calc-inputs* time grows faster than n^2 (see the dashed lines representing the linear and squared functions). Computing the input *profiles*

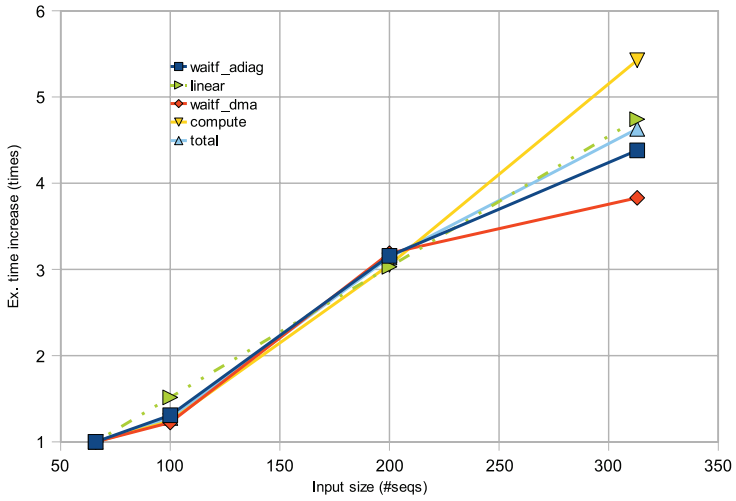


Figure 5.8: Time growth of PA phases in the SPEs.

becomes then the highly predominant applications bottleneck when increasing the number sequences to be analyzed.

In Figure 5.9, bars are grouped to show the behavior of the two SMT threads running on the PPE. As expected (because they reorganize half matrix each) the workload is well balanced among the two.

Although not shown here, one of the procedures that becomes predominant in the *others* category is the *ptracepath* function. *ptracepath* is called after *pdiff* finished aligning two *profiles* and it is in charge of reconstructing the path information so that the alignment can be stored.

5.3.3 Overall application's performance

After putting together the parallelizations and optimizations described before, Figure 5.11 shows the overall ClustalW performance. Although PW dominates the execution time when using a single core, it quickly decreases due to its linear scalability. PA runtime gets also reduced but in a much more limited way and GT remains negligible.

With respect to increasing the input size, Figure 5.12 shows that both PW and PA grow at a similar rate, faster than linearly.

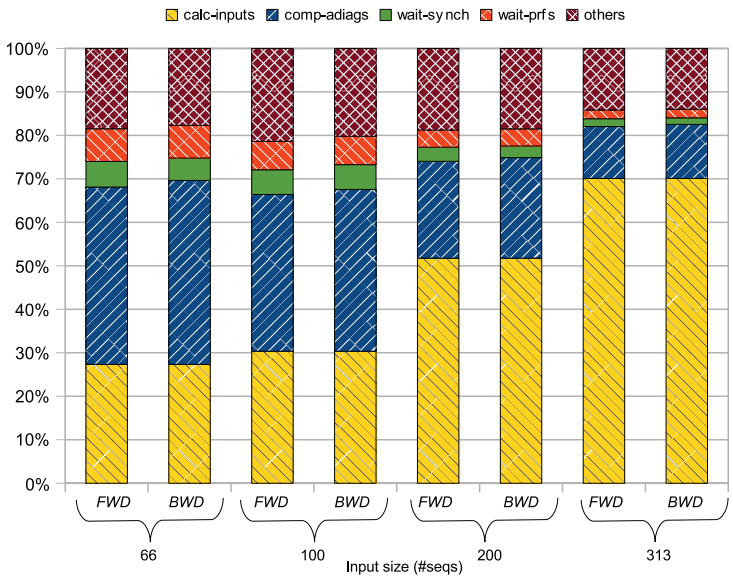


Figure 5.9: Execution time distribution of PA phases in the PPE.

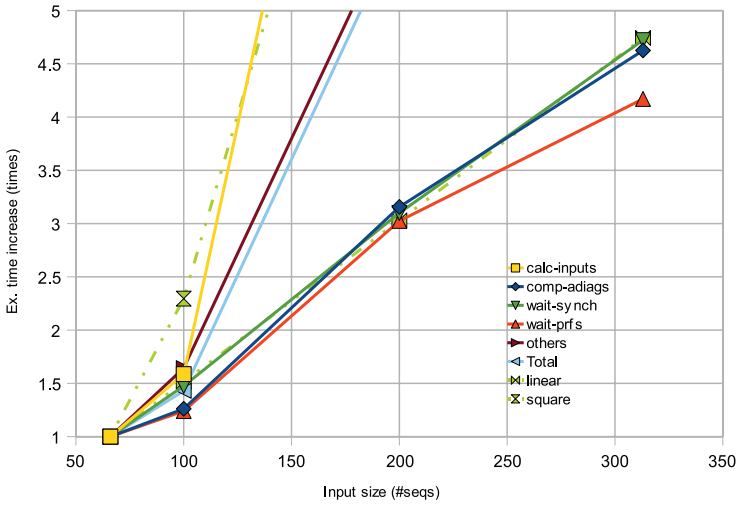


Figure 5.10: Time growth of PA phases in the PPE.

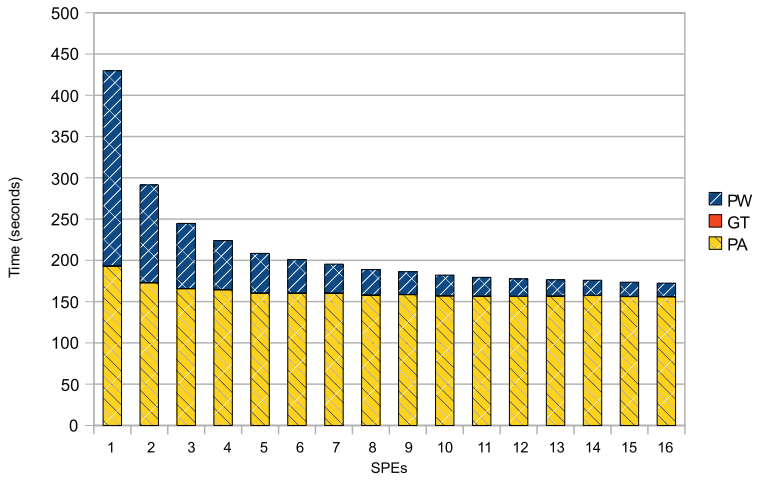


Figure 5.11: Execution time of ClustalW phases with varying number of SPEs.

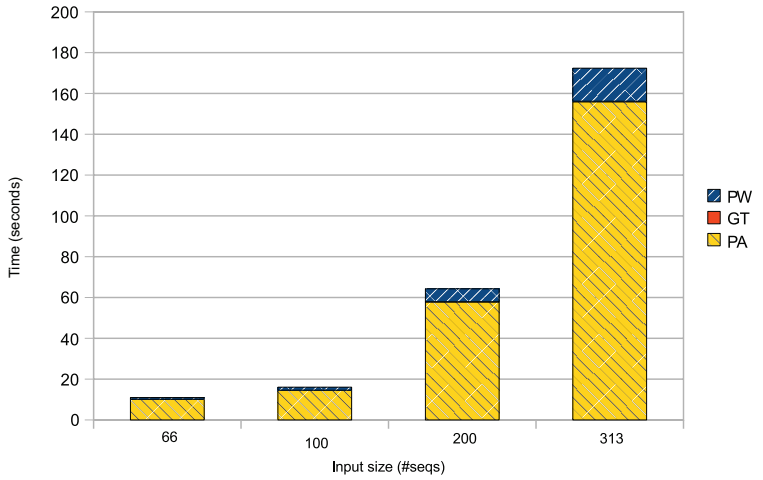


Figure 5.12: Execution time of ClustalW phases with varying number of input sequences.

5.3.4 Limitations

Here we list and discuss some limitations we found at both architecture- and micro-architecture levels.

* *Unaligned data accesses:* The lack of hardware support for unaligned data accesses is one of the issues that can limit the performance the most. When the application needs to do unaligned loads or stores, the compiler must introduce extra code that contains additional memory accesses plus shuffling instructions for data reorganization. If this sort of situation appears in critical parts of the code (as is the case in ClustalW), the performance will be dramatically affected. The loading of the sequence symbols and the access to the scoring matrix require unaligned accesses within the inner loop of *forward_pass* in ClustalW. Later, other authors have proposed coding optimizations that can alleviate this overhead [92].

* *Scalar operations:* Given the SIMD-only nature of the SPEs ISA and the lack of unaligned access support, scalar instructions may cause performance degradation too. Since there are only vector instructions, scalar operations must be performed employing vectors with only one useful element. Apart from power inefficiency issues, this works well only if the scalars are in the appropriate position within the vector. If not, the compiler has to introduce additional instructions to make the scalar operands aligned and perform the instruction. This limitation can be responsible for a significant efficiency reduction. Again, loading the sequence symbols and the scores requires scalar processing if more advanced coding tricks are not used.

* *Saturated arithmetics:* These frequently executed operations are present in Altivec but not in the SPE ISA. They are used to compute partial scores avoiding that they are zeroed when overflow occurs with unsigned addition. This limitation may become expensive depending on the data types. For *signed short*, 9 additional SPE instructions are needed. The 16bit version of *forward_pass* uses saturated addition to prevent the accumulated score overflowing.

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

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

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

* *Local Store size:* For our parallelization of *forward_pass*, the limited LS size was not an issue. However, when more pressure is put on data transfers, multi-bufering is required and it the memory requirements increase significantly. It has also been shown [52] that the LS size is critical for other applications such as Ssearch because it is not always possible to ensure that each database sequence, query sequence and temporal computations fit in the LS. Partitioning the Smith-Waterman matrix in blocks is a useful approach to alleviate this issue [52].

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

5.4 Enhancing the SPE Instruction-set

Based on the knowledge acquired while porting the ClustalW-PW kernel on the SPE, we have proposed the addition of a few instructions to speedup the execution of the inner loop body computations. This section describes the simulator used, the proposed instructions and the acceleration results achieved for the targeted kernel.

5.4.1 Proposed Instructions

Based on Smith-Waterman, the *forward_pass* function computes a similarity measure of two sequences. In our experiments we have used the vectorized and optimized version of *forward_pass* from Section 5.1.2. *forward_pass* is composed of two nested loops that iterate over the entire length of the input sequences. The body of the inner loop is then the target of our analysis for new SIMD instructions.

Computing the maximum between two or more operands is a fundamental op-

Table 5.2: Functionality of the proposed SPE instructions

Instruction	Operation
MAX Rt,Ra,Rb	if(Ra>Rb) Rt=Ra
	else Rt=Rb
MAX3Z Rt,Ra,Rb,Rc	if(Ra>Rb&&Ra>Rc)
	temp=Ra
	else if(Rb>Rc) temp=Rb
	else temp=Rc
	if(temp>0) Rt=temp
	else Rt=0
SUBMX Rt,Ra,Rb,Rc	temp=Ra-Rb
	if(Rc<temp) Rt=temp
	else Rt=Rc

eration often used in sequence-alignment kernels. Consequently, we added the MAX instruction to the SPEs ISA to replace an operation that would otherwise need two SPE instructions. The analyzed kernel uses the Smith-Waterman recurrent formula (see Chapter 2) that subtracts the penalties from the upper and left scores in the dynamic programming matrix. Then, it computes the maximum value with saturation at zero. We also added two instructions to speed up this processing: MAX3Z computes the maximum of three input vectors and 0, and SUBMX computes $max\{a - b, c\}$. Table 5.2 list the proposed instructions and their functionality.

In short, MAX and MAX3Z find maximums, SUBMX finds a maximum after a subtraction and ADDS perform saturated (signed) addition.

5.4.2 Simulation Setup

The evaluation of our instruction-set extensions has been done using the CellSim [1] simulator described in Chapter 3. The proposed instructions have been added to the CellSim's source code, and, in order to make easy use of the new instructions, we have added them as intrinsics to the CellSim GCC (SPU) compiler. The code running on the SPE has been instrumented for the CellSim profiler so as to measure the cycles taken by the function of interest.

Since our proposed instructions are independent of the use of multiple cores and due to the long simulation times, we have only simulated the ClustalW

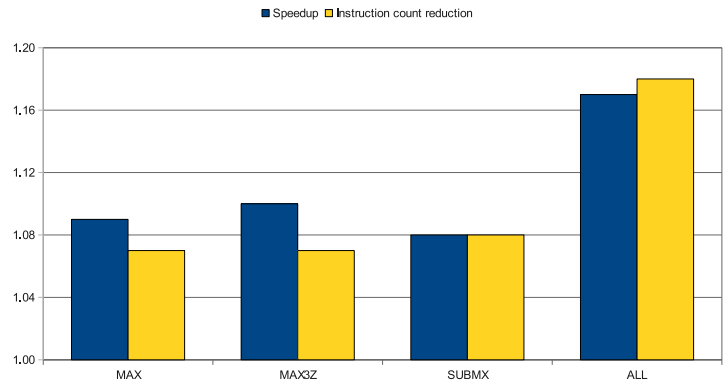


Figure 5.13: Speedup and dynamic instruction count reduction of the proposed instructions compared to the original SPE

most important function (*forward_pass*), running on one SPE and processing two 50-symbols long input sequences. The previous conditions do not affect the analysis since the number of loop iterations and computations only depends on the input size, not on the data content. That is, the speedup will remain the same for any other input data.

New instructions have been chosen by first profiling and then by manual inspection of the kernel code.

5.4.3 Acceleration Results

Using the CellSim profiler we have instrumented the source code running on the SPE. Numbers reported correspond to the cycles consumed by ClustalW kernel *forward_pass*.

Figure 5.13 shows the speedups and instruction count reductions for the *forward_pass* function, the most time consuming kernel of ClustalW. For MAX and MAX3Z, the speedup is larger than the reduction in executed instructions. The reason is that these two instructions replace a sequence of instructions that create dependencies. Collapsing them into a single instruction saves many dependency stall cycles, further contributing to the total speedup. Overall, the new instructions improve performance by 17%.

It should be noticed that the total acceleration achieved does not equals the sum of individual contributions. This is because the use of MAX3Z overlaps

some instances of MAX.

In the parts of the loop body that do not benefit from the new instructions, many cycles are consumed by data reorganization instructions (masks creation, rotates, shifts, shuffles). These instructions are inserted by the compiler in order to make possible scalar manipulations and unaligned memory accesses, not supported by the SPE hardware.

5.5 Summary

In this chapter we have described the mapping and some optimization alternatives of a representative bioinformatics application, namely ClustalW, targeting the Cell BE. We have also presented a qualitative analysis of the architectural shortcomings identified during this process. Our study revealed various architectural aspects that negatively impact Cell BE performance for bioinformatics workloads. More precisely, the missing HW support for unaligned memory accesses and the programming model appear to be the most critical for ClustalW.

Apart from the PW phase analysis, we have gone further and analyzed the second most compute intensive part (PA) that becomes the bottleneck when PW is optimized and parallelized. We have carefully profiled and analyzed the PA behavior and parallelized the *prfscore* computations so that an arbitrary number of SPEs can be used. Results show that the parallel *prfscore* version scales close to linearly with both the number of SPEs and the number of input sequences. On the other hand, the FWD/BWD part gets affected by the limited task-level parallelism and the slow matrix reorganization process in the PPE. Cores with stride memory access capabilities would be able to tackle this bottleneck by reducing the number of load operations required.

Medium to fine grained parallelization for PA does not greatly benefit from adding more cores as only *prfscore* allows scaling. FWD/BWD remains fixed to use two cores and other sequential parts in the PPE (like *calc-inputs* and *ptracepath*) become the bottlenecks. These phases suffer from the poor PPE ability to extract ILP.

Due the its computational complexity, *calc-inputs* dominates the PA time share for input sets of 200 or more sequences.

Given that the PW phase is computation-bound, in this chapter we have also proposed three new instructions for the SPE instruction-set, targeting sequence alignment kernels. We have manually included them in the *forward_pass* function from PW and showed acceleration results of 17%. Furthermore, the pro-

posed instructions are fundamental not only in the *forward_pass* kernel but in various other critical functions of ClustalW, and in other applications of the same field like Ssearch and Hmmer [7].

Note. The content of this chapter is based on the the following papers:

A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez, F. Sanchez, A. Azevedo, C. Meenderinck, C. Ciobanu, S. Isaza, G. Gaydadjiev. **The SARC Architecture**. *Micro, IEEE,* 30(5):16-29, Sep.-Oct. 2010.

S. Isaza, F. Sanchez, G. Gaydadjiev, A. Ramirez, M. Valero. **Scalability Analysis of Progressive Alignment on a Multicore**. *Proceedings of the* 3^{th} *International Workshop on Multicore Computing Systems, MuCoCoS - IEEE*, pp.889-894, Krakow, Poland, Feb. 2010.

S. Isaza, F. Sanchez, G. Gaydadjiev, A.Ramirez, M. Valero. **Preliminary Analysis of Cell BE Processor Limitations for Sequence Alignment Applications**. *Proceedings of the* 8^{th} *International Symposium on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS - Springer LNCS*, pp. 53-64, Samos, Greece, Jul. 2008.

6

Performance Scalability on Manycores

In this chapter, we investigate various manycore architecture configurations to study performance scalability. First, we map ClustalW onto a manycore that uses either general-purpose processors or application-specific accelerators. This allows us to project state-of-the-art processors and accelerators performance into scaled future systems and determine the requirements in terms of shared hardware resources. Although the fundamental pros and cons of the two approaches are known (flexibility and ease of programming vs. higher throughput and energy efficiency), this chapter aims at investigating the performance consequences of using each of them and analyzing the sensitivity to various hardware components (e.g. memory, cache, bus). Experimental results show that an accelerator-based manycore can still achieve high efficiency when using hundreds of cores. Furthermore, because those accelerators can be run at lower frequencies, they will be able to stay within the allowed power budget (and will dramatically reduce energy consumption as well).

In any case, performance will be largely determined by the available parallelism in the application, which in some cases may depend on the input data characteristics. In this chapter we take input data sets from BioPerf [22] and characterize them against three additional input sets that we build ourselves. Conclusions are made on the performance implications when exploiting the application's parallelism.

Furthermore, this chapter also studies the scaling and parallel behavior of HMMER. We use HMMERCELL (a port of HMMER to the Cell architecture [55]) as baseline for the analysis and propose two optimized versions based on the bottlenecks found. We simulate a manycore with up to 512 processors in order to measure performance. We compare the three parallel versions, report their relative speedup and their performance scalability with the number of cores.

The main contributions of this chapter are:

- The characterization of various types of input data sets and their impact on performance when exploiting parallelism;
- The quantification of bandwidth and synchronization requirements;
- The performance scalability comparison of a manycore based on general-purpose processors against one based on application-specific accelerators;
- The load balancing improvement in ClustalW for high number of cores processing a small input data set;
- An improved HMMER parallelization and the analysis of program bottlenecks.

The use of a hybrid high-level/cycle-accurate simulator setup allows us to abstract out the parts of the system that do not affect our analysis. This, in turn, enables high speed simulations enabling the modeling of systems with hundreds of cores.

The content of this chapter is organized as follows. Section 6.1 describes the target applications and their parallel versions and Section 6.2 the experimental methodology for the simulations. Section 6.3 analyzes the input data sets and their impact on programs' behavior. Finally, Section 6.4 discusses the results and Section 6.5 summarizes the chapter.

6.1 Application's Description and Parallelism

In this section we recall the basic characteristics of ClustalW and HMMER applications, more extensively described in Chapter 2. Furthermore, we discuss the coarse-grain parallelism available that allows us to harness a manycore platform like SARC.

6.1.1 ClustalW

Recalling from Chapter 2, ClustalW [90] is a widely used software to perform MSA. It uses a heuristic known as *progressive alignment* in three steps:

Table 6.1: Characterization of various ClustalW input sets.

		Profiling $n(l)$*							
		P13569		P10091		NM_000249	BioPerf-A	BioPerf-B	BioPerf-C
	$O(Time)$	500(940)	318(1,035)	**500(355)**	318(355)	83(1,641)	50(52)	66(1,083)	318(1,045)
PW	$O(n^2l^2)$	98.5%	97.7%	98.5%	97.9%	90.5%	82%	61%	82%
GT	$O(n^4)$	Negl.	Negl.	Negl.	Negl.	Negl.	Negl.	Negl.	Negl.
PA	$O(n^3 + nl^2)$	1.5%	2.3%	1.5%	2.1%	9.5%	18%	39%	18%
Score		8,809,293	16,144,863	9,785,331	12,803,740	9,778,088	120,625	1,298,083	-4,528,287
Norm. Std. Dev.		0.44	0.41	0.04	0.05	0.49	0.15	0.71	0.75
Max. Speedup		66.67×	43.48×	66.67×	47.62×	10.53×	5.56×	2.56×	5.56×

* n is the number of sequences in a set and l is the average length.

PW, GT and PA. PW is responsible for computing similarity scores for all sequence pairs; GT uses these scores to cluster sequences in a tree structure; and PA computes the final alignment by walking the tree, adding (aligning) one sequence (or profile) to the MSA at a time. In PW, a similarity score is computed for all sequence pairs. This score is computed by aligning the two sequences (without actually recovering the alignment but just the score), based on the Smith-Waterman [87] algorithm. Table 6.1 shows the computational complexity of each of the ClustalW phases (where n is the number of sequences and l is the average sequence length) and profiling results of the original (sequential) ClustalW code. We see that even for a large[1] n, PW takes most of the time in a uni-processor execution. For an input of n sequences, PW must compute $n(n-1)/2$ alignments, called tasks (or jobs) here. On the other hand, GT only grows significantly for the case of many short input sequences. For most of the cases, including the test inputs used in our experiments, GT time is negligible. Although PA takes a small but non negligible portion, this chapter focuses on the much more demanding PW phase.

Parallel ClustalW-PW

Tasks in PW are independent of each other which enables the parallel use of, in principle, as many processors as the available number of tasks ($n(n-1)/2$). To extract the task parallelism we followed a master-worker scheme. One master processor is in charge of submitting tasks to a queue by taking pairs of sequences as they come from the database. Each job queue entry contains an ID and a memory pointer to the task parameters: the pointers to the input sequences, their lengths and the gap penalties. Worker processors poll the queue until they obtain a job and then proceed to fetch the input data from main memory (using DMA). Once data is loaded in the Scratchpad Memory (SPM), a worker core performs the computations and writes back results to main memory on completion (see Chapter 3 for details on the architecture). Most of the communication traffic in the system will be due to workers fetching input data. During the write-back of results, only a score needs to be transferred, which is negligible compared to the input sequences. The master must wait for the completion of all the scheduled jobs before the next phase in ClustalW (GT) can start.

Table 6.1 shows the profiling results for various input data sets and the corresponding theoretical speedup limit for the parallelization of PW, as obtained

[1] Although current databases contain millions of sequences, current MSA heuristics are not reliable for aligning more than a few thousands.

from applying Amdahl's law [21]. Although PA can be parallelized to some extent as well [26,61], its tasks have dependencies that limit the achievable degree of parallelization. Hence, the total application's speedup will be largely determined by the characteristics of the input dataset.

6.1.2 HMMER

HMMER [37] is a tool specifically aimed at protein sequence analysis. Groups of protein sequences thought of as belonging to the same family are modeled with profile Hidden Markov Models (HMMs). Our experiments focus on one tool within the HMMER suite: *hmmsearch*. With this program, an HMM can be compared to a protein sequence database (see Chapter 3). The goal is to find the proteins from the database that are most similar to the protein family represented by the HMM query. To perform this comparison, the Viterbi algorithm is used to generate an alignment score. The *hmmsearch* output is a list of high scoring sequences and their alignment to the HMM. Execution time is dominated by the Viterbi decoding phase, which is performed once for each sequence in the database. Profiling of the original HMMER code running on a single processor shows that for all but the simplest workloads, this phase accounts for 98+% of total running time.

Parallel *hmmsearch*

For our experiments, we use HMMERCELL [62] as baseline: HMMER v2.3.2 ported to the Cell BE processor [55]. Since the execution time of *hmmsearch* is almost exclusively formed by the execution of the Viterbi function, the parallelization strategy focuses on this program phase and follows a master-worker scheme. The master processor creates parallel jobs that are consumed by the workers. Parallelism is used at two levels: coarse-grain parallelism by spawning threads and fine-grain parallelism by using SIMD processing within the Viterbi kernel. Due to memory limitations caused by the small Local Stores, the Viterbi algorithm is modified to use a smaller memory footprint. Hence, worker processors do not provide a full alignment but only produce an alignment score. High scoring alignments are reprocessed (by running the full Viterbi) on the master processor to obtain the actual alignment (more details on HMMERCELL can be found in [62]).

After extensive profiling and tracing of HMMERCELL we present here two optimized versions that are more scalable and target the larger and more general SARC manycore architecture (see Chapter 3). Figure 6.1 shows a diagram

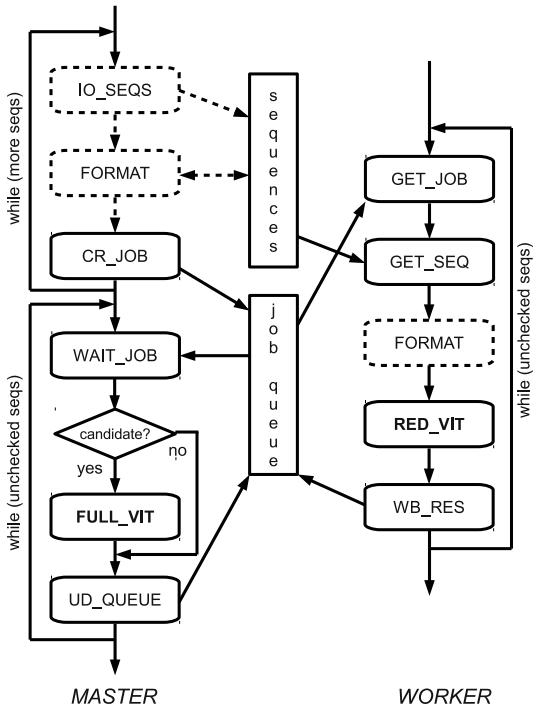


Figure 6.1: Parallel HMMER diagram.

of the parallel HMMER functioning and Table 6.2 lists the phases involved. For every sequence in the database, two steps are involved at first. A sequence is load from the disk into main memory (IO_SEQS) and then it is formatted (FORMAT) for appropriate Viterbi processing. In HMMERCELL, both phases are sequentially handled by the master processor which creates a centralized bottleneck. Since the Swiss-Prot database amounts to 234MB, we notice that in a realistic scenario, the database can be permanently held in memory. Thus, the application is relieved from the I/O bottleneck and should be able to efficiently use more processors. The second observation is that the FORMAT phase is independent for every sequence in the database and therefore can be performed by the workers in parallel. The three parallel versions described are listed here for reference:

- **BASE**: Baseline HMMERCELL parallelization with I/O reads and sequential formatting of sequences.
- **M_FORMAT**: Removing the I/O bottleneck by holding database in

main memory. FORMAT is performed by the master.

- **W_FORMAT**: Besides removing I/O, the FORMAT phase is performed by the workers in parallel.

The processing done in FORMAT consists in replacing every amino acid by its index in the alphabet. Hence, moving this simple operation to the workers side only implies a negligible increase in code size and no extra memory for the sequence as it can be overwritten when formatted.

For the sake of clarity, Figure 6.1 does not show the loading of the HMM as this is done only once, at the beginning.

Table 6.2: HMMER program phases.

Phase name	Description
IO_SEQS	Read sequences from disk.
FORMAT	Format sequence for proper processing.
CR_JOB	Create an entry in the job queue.
WAIT_JOB	Wait for an available job to process.
FULL_VIT	Compute full Viterbi algorithm.
UD_QUEUE	Update entry status as done.
GET_JOB	Get a job entry from the queue.
GET_SEQ	DMA sequence from memory to SPM.
RED_VIT	Compute reduced Viterbi algorithm.
WB_RES	Write-back result in queue.

6.2 Experimental Methodology

For our analysis we have used the *TaskSim* simulator [75, 77] that models a parametrizable multicore architecture (SARC). Both TaskSim and SARC are described in Chapter 3. To obtain the traces that feed the simulator, we have manually instrumented a ClustalW (v.1.83) port to the Cell BE. Thereafter the instrumented code was run on an IBM QS21 Blade, running at 3.2GHz with 4GB of RAM. The code has been compiled with GCC4.1.1 and -O3 flag. Only the master and one worker processor (that is, a PPE and a single SPE in Cell BE) were used to generate the trace. This in order to create a single pool of tasks that can be dynamically scheduled to any of the simulated worker processors.

Table 6.3: List of parameter names and value ranges.

Parameter name	Range of values
Number of workers	1,2,4... 1024
Workers' throughput	1×,100×
DRAM latency (cycles)	1,64,128,256,512,1024,2048,4096
Number of MICs	1,2,4,8,16,32
L2 cache size	0KB,64KB,128KB,512KB,1MB
L2 cache banks	1,2,4,8
GDB rings	1,2,4,8,16,INF
SQ latency (cycles)	1,32,128,256,512

To ensure our experimental simulations' accuracy, we ran ClustalW-PW on the Cell BE Blade using 8 SPEs and compared the execution time with that obtained from simulating a system with equal configuration. This allowed accuracy verification when simulating multiple cores from a trace generated using only one. Results showed that the simulation error is under 2%, considered adequate for the purpose of our study.

The intention of the experiments presented here is to evaluate the sensitivity of the application's performance to various critical architectural parameters and to determine the minimum configuration that achieves the maximum speedup. Table 6.3 lists the parameters along with the value ranges used in the simulations. For every figure shown in Section 6.4, only one parameter is varied. All other parameters are set to appropriate values so as to emphasize the effect of the one being analyzed.

The simulation output also contains traces (like the ones graphically shown in Section 6.4.1) that we analyze using Paraver [13].

Later, Section 6.3 discusses the choice and influence of the input data sets.

6.2.1 Workers

One special parameter that needs discussion is the *workers' throughput*. One of the key aspects that determines system performance is obviously the data processing power of an individual worker. However, increasing workers' throughput does not always result in an equal increase in application's performance.

Shared hardware resources like the buses or the memory may become the bottleneck.

As explained before, the hot-spot in ClustalW-PW uses the Smith-Waterman algorithm. Because that kernel has been the focus of many research papers and even commercial products in recent years, it is possible for us to compile an overview of the wide range of achievable single-core performances. These are described in Chapter 3 and Table 3.1. We compare simulation results of a multicore based on general-purpose processors ($1\times$ throughput) with one using accelerators ($100\times$ throughput). These accelerators are dedicated hardware structures that implement a *systolic array* architecture to exploit the parallelism available in the Smith-Waterman processing [61]. Based on Table 3.1 we use $100\times$ as a conservative value. We run experiments with different architectural parameters in order to find out the minimal configuration that achieves maximum performance using $100\times$ accelerators.

6.3 The Influence of the Input Data Sets

In this section we analyze various input data sets and discuss their correlation with the applications' behavior.

6.3.1 ClustalW

Various inputs with different characteristics are used in our simulations (see Table 6.1). First, we take the three input data sets (A, B and C) provided in BioPerf [22] that have been used in some of the related work [52, 53, 80, 93]. Although not clearly stated by the BioPerf authors, the sequences seem to be randomly selected from the databases. As a result, sequences are highly dissimilar as confirmed in the low alignment scores obtained with ClustalW. Table 6.1 also shows the sequence length standard deviation (normalized with the average length) as a measure of how different sequence lengths are within each input set.

A different methodology is followed to generate additional (more realistic) test sets, as reported in previous work [26]. We randomly pick protein P10091 CYSA_MARPO (365 amino acids) as query to perform a BLAST search against the SwissProt database. Since ClustalW is commonly used to align sequences that are expected to have some similarity or evolutionary relationship, we take the first 500 protein hits returned by the search to use as input for the MSA. We apply the same methodology to protein P13569 (1480 amino

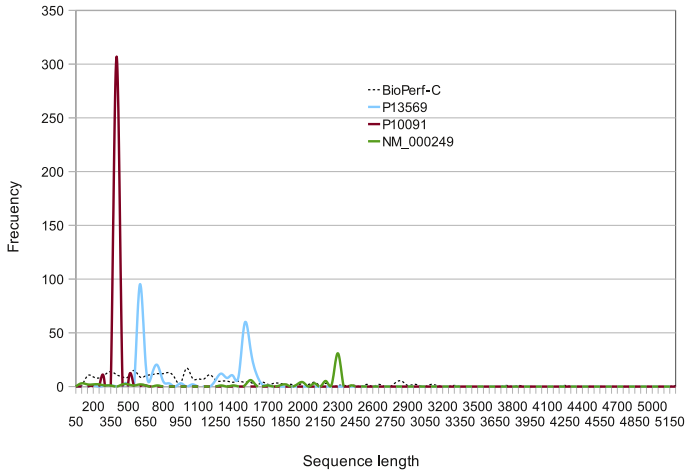
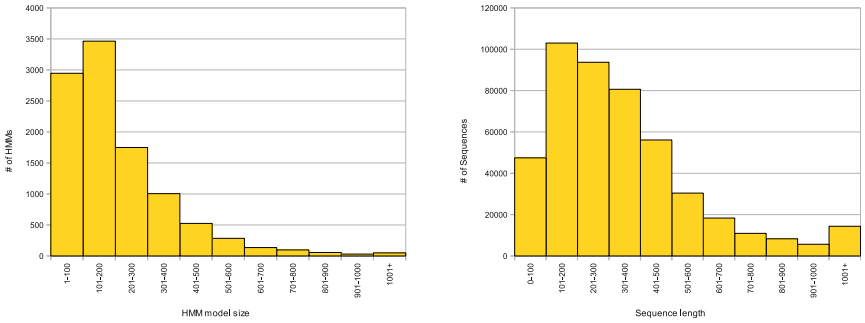


Figure 6.2: Sequence length histogram for different input test sets.

acids) and to DNA sequence NM_000249 (2662 nucleotides) with the GenBank database.

Figure 6.2 shows a sequence length histogram of input data sets analyzed. The taller the peaks in a given set, the more homogeneous the sequences' lengths are. Data from BioPerf is clearly the most heterogeneous in sequences' length and, as confirmed by the alignment scores from Table 6.1, also dissimilar in the biological sense. By looking at Table 6.1 and Figure 6.2, a correlation can be observed between alignment scores, length standard deviation and profiling results. Test sets with a low score and high length variation spend less time on the most parallel part of the program, i.e. PW. The reason for this phenomenon has to do with the calculation of boundaries in the recursive structure of the PA alignment algorithm, which is out of the scope of these experiments. Although in a uniprocessor scenario this does not matter, in a manycore it has great performance implications as the parallelizable portion is smaller. The last row in Table 6.1 shows the maximum application speedup achievable if PW times is reduced to zero.

In our attempt to simulate a realistic scenario, we use 500 sequences corresponding to protein P10091 with lengths most similar to the average sequence length of the SwissProt database (355 amino acids). In PW, a set of 500 sequences produces 124,750 coarse grain tasks that can be run in parallel, which is enough to stress a system with up to 1024 cores as it is the case for our study. Only for the experiments in Section 6.4.1 we use sequences from BioPerf in



(a) Pfam (b) Swiss-Prot

Figure 6.3: Entry length histogram of protein databases.

order to show an issue that arises in such execution scenario.

6.3.2 HMMER

Protein sequence data sets used for the HMMER simulations are from the Swiss-Prot database [15] and HMMs from Pfam [88]. Figure 6.3 shows the current model and sequence length distribution for Pfam and Swiss-Prot databases. Based on this information, input test sets have been chosen. Two randomly picked HMMs of 100 and 500 states are used to study a typical and a demanding execution scenario. In order to keep traces under a reasonable size, we have used a randomly selected subset of Swiss-Prot database for the simulations (20,000 sequences). Every sequence creates a job and so we have enough jobs to stress our manycore system.

Due to the algorithm structure used in *hmmsearch*, where there is a single kernel responsible for the vast majority of processing (Viterbi), changes in the input data do not have an impact on the program's parallelism, as in ClustalW.

6.4 Simulation Results

In this section we discuss the simulation results. For ClustalW we study the scalability of an accelerator-based manycore and discuss the performance gap with general-purpose cores. For HMMER we focus on the program bottlenecks and compare three parallelization options.

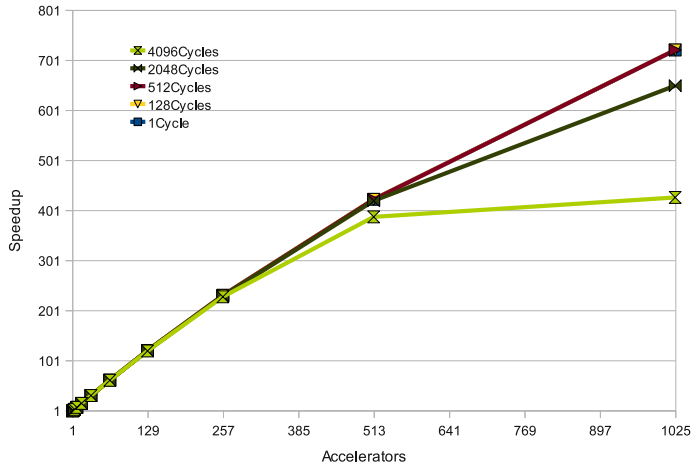


Figure 6.4: Sensitivity to memory latency. Other parameters are: 32 MICs, No cache, INF Rings.

6.4.1 ClustalW

First, we study the sensitivity of various architectural parameters for the accelerator-based multicore. Then, we look at the load balancing problem of a general-purpose-based manycore that arises when the input set is small relative to the number of cores. Finally, we run a set of simulations where we compare the performance achieved by the accelerators with that of the general-purpose cores.

Accelerator-based Multicore

Tasks in ClustalW-PW are very coarse and have a high computation-to-communication ratio, that is, a large amount of processing is done on a relatively small data set. Despite of this, the use of accelerators manages to lower this ratio and the shared resources (e.g. memory, buses, SQ) requirements increase. In this section we analyze the impact of the architectural parameters from Table 6.3, using up to 1024 accelerators ($100\times$). In Figures 6.4 - 6.9, the baseline is a single accelerator core.

Figure 6.4 shows the impact of varying the latency to access main memory. This simulation has an artificial setup in which we replace the shared cache and the DDR memory system with an ideal conflict-free memory having a configurable latency between 1 cycle and 4096 cycles. For reference purposes,

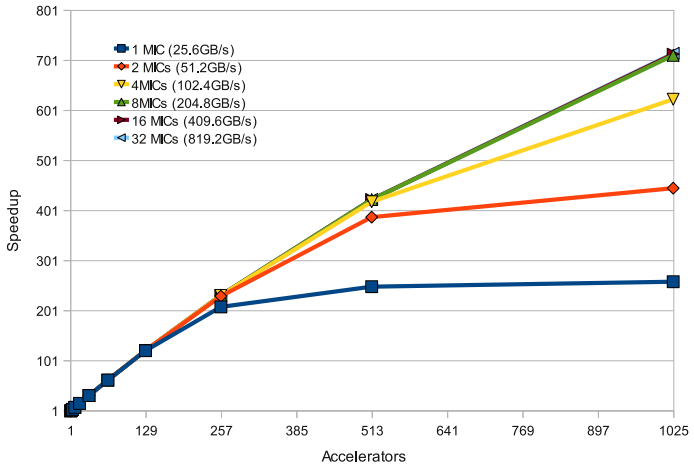


Figure 6.5: Sensitivity to memory bandwidth. Other parameters are: No cache, INF Rings.

notice that the average DDR3 latency is between 150 and 250 cycles. On top of the raw memory latency, the time to traverse the buses has to be taken into account as well. Results show that the application tolerates well even large latencies. Only by increasing latency beyond 2048 cycles, performance degradation is observed. This result is expected since most data transfers are done in large chunks ($\sim$ 1KB) to fetch the sequences to the accelerators' SPMs.

Figure 6.5 shows the sensitivity to memory bandwidth. We vary the number of MICs between 1 and 16. Every MIC provides 25.8GB/s with two DRAM channels of 12.8GB/s each. We disable the cache and set the GDB rings to infinite in order to isolate the effect of the off-chip memory bandwidth accesses. Results show that for 1024 accelerators, 8 dual-channel MICs (204.8GB/s) are needed to sustain the traffic and achieve nearly the maximum performance. For 512 and 256 cores, 4 MICs (102.4GB/s) and 2 MICs (51.2GB/s) are enough though. Below 128 cores, bandwidth is not an issue as the application is still compute bound. With 1024 accelerators, going from 1 to 4 MICs increases the speedup from 259$\times$ to 623$\times$.

In Figure 6.6 we investigated different L2 cache sizes and compare against having no cache. For accessing off-chip memory, only a single-channel MIC (12.8GB/s) is configured so that the benefit of the cache size can be better appreciated. The GDB is set to infinite rings for the same reason. We can see that a small cache of only 32KB is enough to achieve close to the maximum

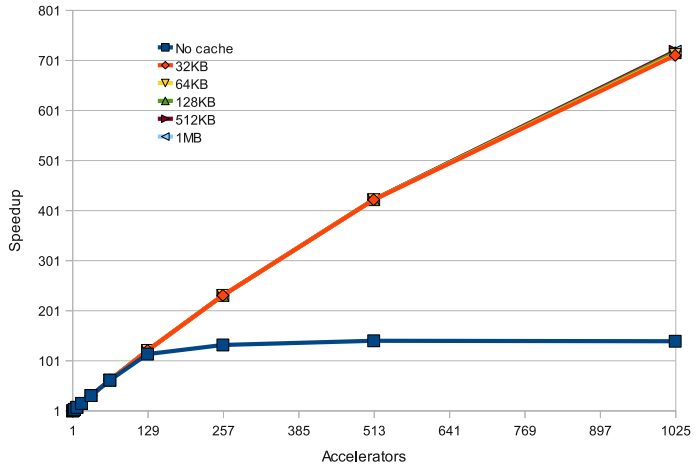


Figure 6.6: Sensitivity to cache size. Other parameters are: 4 cache banks, 1 MIC, 1 DRAM, INF Rings.

speedup. This is because in aligning all-to-all sequences, workers reuse a lot of the already small dataset. When increasing the cache size from 32KB to 1MB, the hit rate improves from 91% to 99%. However, 32KB are enough to achieve 98.8% of the 1MB cache performance (for 1024 cores).

However, having no caches dramatically degrades performance. Notice that the no-cache curve in Figure 6.6 has lower performance than the one MIC curve in Figure 6.5. This is because in Figure 6.5, MICs are always set to have two DRAM channels, as opposed to the single channel in Figure 6.6. The performance gain of using caches comes mostly from the bandwidth provided by the 4 cache banks, for a total cache bandwidth of 102.4 GB/s. This is confirmed in Figure 6.7 where we can see that the sensitivity is higher with respect to the number of banks, rather than the cache size. Notice that results from Figure 6.6 have important implications. They tell us that a tiny 32KB cache can capture most of the memory traffic that otherwise would have to go off-chip. From Figure 6.5 we can see that around 8 dual-channel MICs are needed to achieve similar speedups. Therefore, a small cache can save a lot of hardware complexity and more important, a lot of energy.

Next we simulate several GDB ring numbers in order to see the bandwidth sensitivity from the bus perspective. Figure 6.8 shows that the configuration with one ring (25.6GB/s) can only serve 256 accelerators at most. We can also see that all the traffic can be captured with an interconnect able to provide

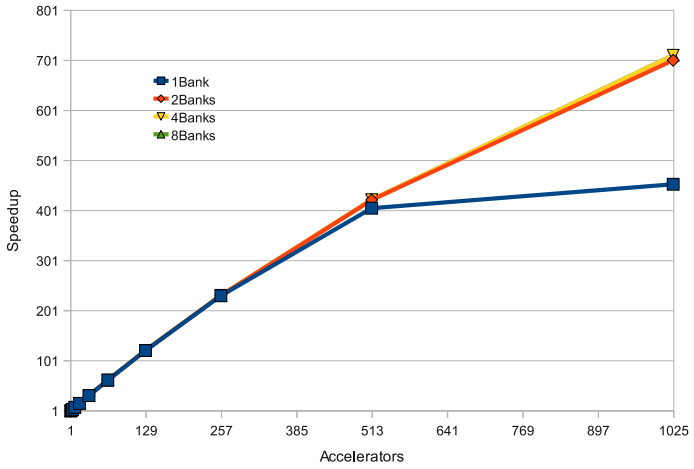


Figure 6.7: Sensitivity to cache banks. Other parameters are: 1MB cache, 1 MIC, 1 DRAM, INF Rings.

102.4GB/s (4 rings) of aggregate bandwidth. Figure 6.8 shows that the maximum speedup can improve from 336× with one ring to 713× with 4 rings.

Afterwards, Figure 6.9 shows the tolerance to synchronization latency. Because it is a centralized and shared resource, it is a potential bottleneck when using many accelerators. Results show that up to 128 cycles are tolerated by the application. If we consider an SQ that can hold all the total tasks in our setup, a latency below 128 cycles is quite possible. There are 124,750 tasks with each entry taking 8 Bytes (an ID and a pointer). This multiplied by the two queues needed makes a total size of 1MB, small enough to have a latency smaller than 128 cycles (the Cell BE LS has 256KB with 6 Cycles latency). When having smaller tasks, the SQ bottleneck can be reduced by adding queues per cluster or per core. This result shows that using main memory to synchronize (increasing latency) could significantly affect performance.

Finally, due to low pressure imposed on shared hardware resources by the general-purpose cores, the minimal system bandwidth configuration (1 MIC, 1 ring, 1 cache bank) was able to sustain the maximum performance.

Load Balancing with GP Workers

In ClustalW-PW, tasks' duration directly depends (quadratically) on the length of the two processed sequences. When the input set is heterogeneous in se-

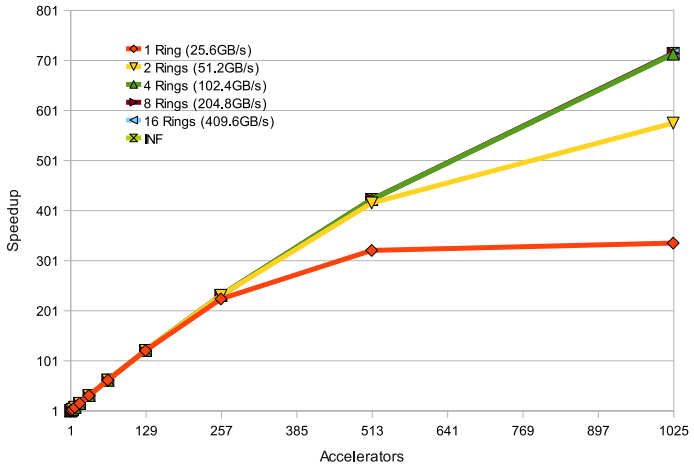


Figure 6.8: Sensitivity to GDB bandwidth. Other parameters are: No cache, 32 MICs.

quence length, as it is the case with the BioPerf sets (see Table 6.1 and Figure 6.2), the variation in tasks' duration will be very high, up to two orders of magnitude. On the other hand, if many processors are involved and the input set is not large enough, each processor will end up with only a few tasks. With the default task scheduling policy, this leads to a few processors finishing much later than others because they had to process much larger tasks.

To look into this issue, we took 200 sequences from input C in BioPerf (due to their heterogeneity) and run simulations with up to 1024 cores. This results in 19,900 tasks, every core getting roughly 19 tasks to process on average. The lower part of Figure 6.10 shows how the default distribution of tasks behaves. It is a screenshot obtained with Paraver that allows us to see the behavior of all processors in time. Every horizontal line represents one core's state. The load unbalancing is appreciated in the right part of the plot.

In order to improve the load balancing we applied a simple task sorting strategy. We noticed that we can roughly estimate all tasks durations in advance just by looking at the sequence lengths. Using that information, we sort the tasks so that longer ones are scheduled first, distributed among different workers. The upper part of Figure 6.10 shows a significant improvement in the load balancing and consequently in performance. However, some time should be spent in the sorting computation, creating an overhead [2] (see gray segment

[2]The overhead time was measured on an Intel Core 2 Duo running the *qsort* function from the Standard C Library. Using a parallel sorting algorithm can reduce this overhead.

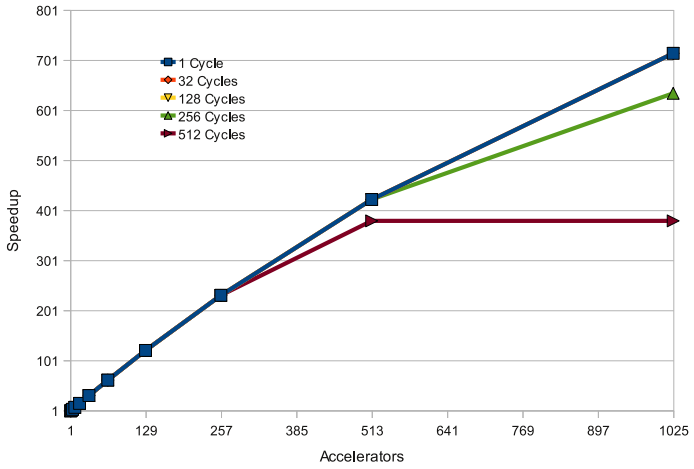


Figure 6.9: Sensitivity to SQ latency. Other parameters are: No cache, 32 MICs, INF rings.

in left part of upper Figure 6.10). Figure 6.11 shows the scalability improvement of task sorting for this type of input set. These results correspond to general-purpose cores. For accelerators, because the sorting is not parallel nor accelerated, the sorting overhead becomes comparable with tasks computation and hence it is not worth. In the end, task sorting can accelerate processing time by 28%. The maximum speedup of $1\times$ workers (Figure 6.11) increases from $674\times$ to $758\times$.

Accelerators vs. General-purpose

Figure 6.12 shows the performance scalability of ClustalW-PW with respect to the number of cores used and compares the case of using accelerators with that of general-purpose cores. For both simulation sets we put the configuration that achieved the maximum performance, as discussed in Section 6.4.1. Speedup curves use one master and a single general-purpose worker as baseline. The vertical axis uses logarithmic scale in order to fit both performance results in a single plot while still appreciating their performance gap. Notice that due to the logarithmic scale, the flattening of the curves does not mean saturation. Dashed lines show the linear ideal scaling as a reference. Results show that even when using accelerators, we can scale performance significantly and efficiently use the 1024 cores.

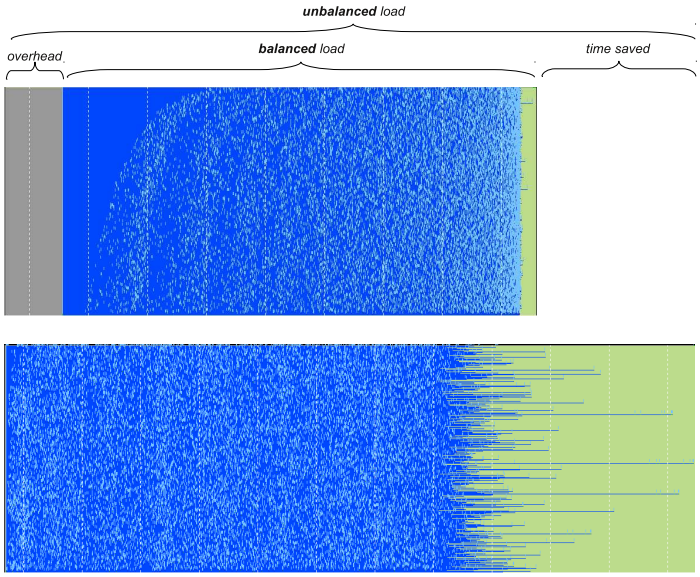


Figure 6.10: Load balancing with 200 sequences as input, with 1024 general-purpose workers. The vertical axis represents the cores and the horizontal axis represents time. Dark (blue) segments are computations and light ones (gray and green) show idleness.

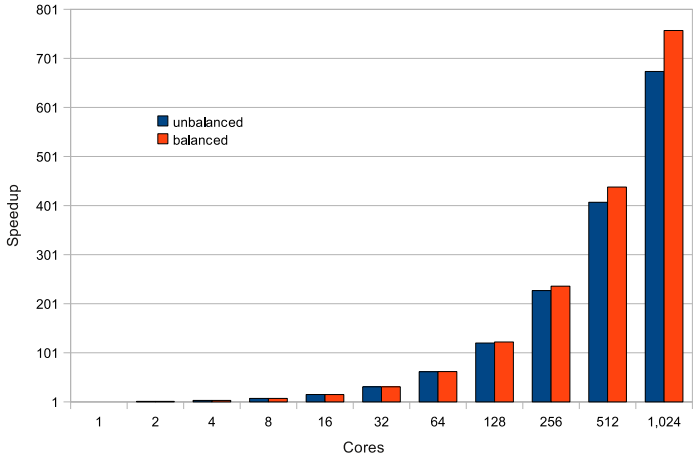


Figure 6.11: Performance scalability with and without task sorting using general-purpose cores. 200 sequences from BioPerf are used as input set.

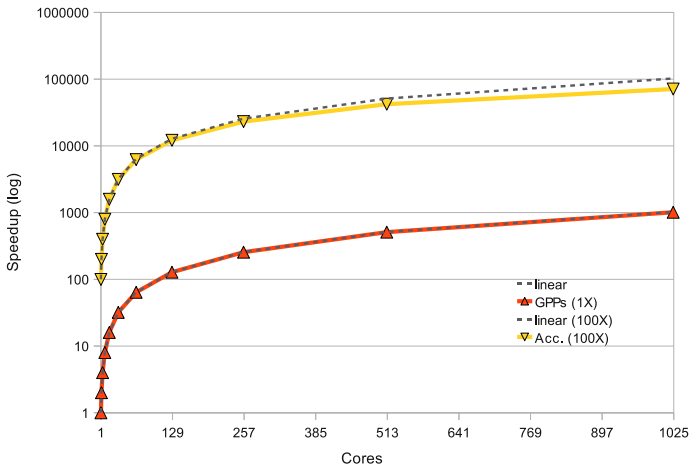


Figure 6.12: Performance comparison for up to 1024 workers of two types: general-purpose cores ($1\times$) and application-specific accelerators ($100\times$). The vertical axis is in *Log* scale and the baseline is a single general-purpose core.

Table 6.1 shows that for the input set used in the simulations, the theoretical maximum speedup of ClustalW is $66.67\times$ when only PW is parallelized. Since the maximum speedup achieved for PW was $720\times$, this translates into $61.1\times$ total application speedup. However, PA parallelism has been reported [26, 61] to achieve around $4\times$ speedup on average. This would increase the speedup impact of our accelerator approach from $61.1\times$ to $195\times$.

Although power analysis is out of the scope of this thesis, it is important to notice that the performance gap that appears in Figure 6.12 is likely to be higher when considering a real chip. The simultaneous functioning of hundreds of general-purpose cores running at the typical frequencies of about 3GHz will not fit within the power budget of current CMOS technologies. Such cores will have to be run at lower frequencies, with a direct impact on performance. On the other hand, application-specific accelerators are designed to run at lower frequencies wile still providing high computational performance. In fact, their advantage comes from the fact the circuits are designed to efficiently match a particular application or application domain, not from running complex hardware structures at very high frequencies.

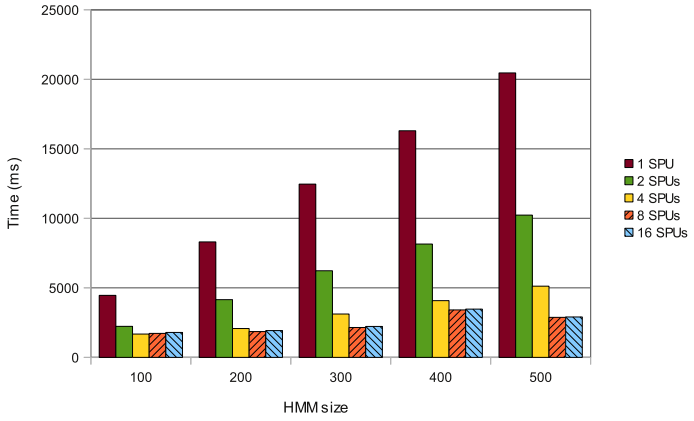
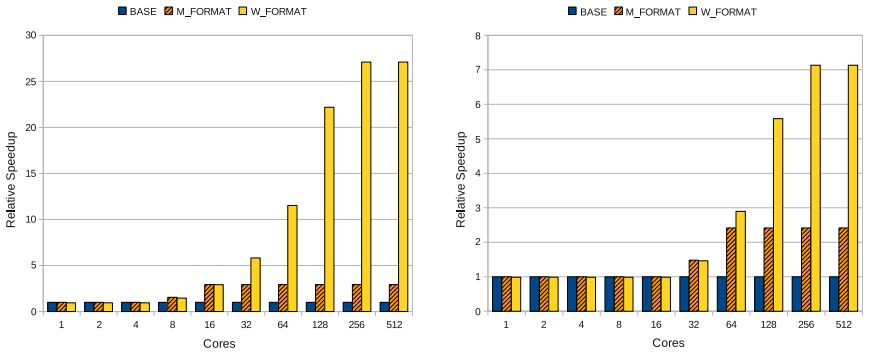


Figure 6.13: HMMERCELL execution time overview.

6.4.2 HMMER

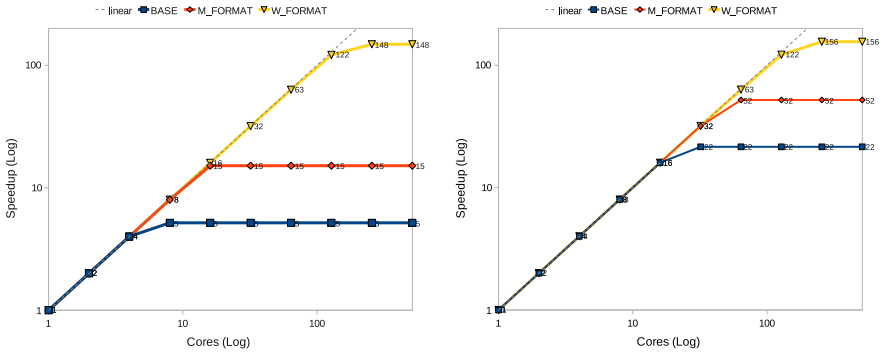
In this section we first show profiling results of HMMERCELL in order to have an overview of performance scalability with various types of inputs. We use 5 distinct HMMs with lengths from 100 to 500 positions, thus covering the average and demanding scenarios as follows from Figure 6.3(a) (larger HMMs need more processing). Then we create a sequence set consisting of 20.000 randomly selected sequences with length distribution identical to the Swiss-Prot database. Figure 6.13 gives an overview of HMMERCELL performance where some basic characteristics on how HMMERCELL reacts to changes in input parameters are revealed: the use of a longer HMM model size requires correspondingly longer execution time; in general, the use of additional SPEs leads to shorter execution times; and only a certain number of SPEs can be used effectively, depending on the workload. Due to management overhead, using more SPEs results in identical or even deteriorated performance.

Afterwards, we investigate the performance of the two optimized versions (see Section 6.1) and compare it with BASE on the SARC architecture. Figure 6.14 shows the speedup provided by the M_FORMAT and W_FORMAT versions over BASE, for various processor counts. We analyze both the average case (100 states HMM) and a more demanding one (500 states HMM). On Figure 6.14(a) we see that for the short HMM there is no benefit when using 4 or less cores. For few cores, the I/O overhead does not affect performance because the execution time is largely dominated by the computation of the Viterbi



(a) Short HMM (100 states) (b) Long HMM (500 states)

Figure 6.14: Speedup relative to the BASE version.



(a) Short HMM (100 states) (b) Long HMM (500 states)

Figure 6.15: HMMER performance scalability with a single worker core as baseline.

kernel (RED_VIT). On the other hand, a large HMM imposes an even higher demand on the RED_VIT processing and since it is executed in parallel, more cores (16) can be efficiently used (Figure 6.14(b). By holding the database in main memory and thus avoiding the I/O bottleneck, a $3\times$ and $2.4\times$ speedup is obtained for short and long HMMs respectively. Finally parallelizing the FORMAT phase brings the largest improvement. Speedups of up to $27\times$ and $7\times$ are achieved for each case, compared to the BASE version. For short HMMs, the impact is significantly larger due to the fact that the sequential parts amount to a bigger portion of the execution. Compared to the M_FORMAT version, W_FORMAT reaches $9.3\times$ and $3\times$ speedups for the short and long HMMs respectively.

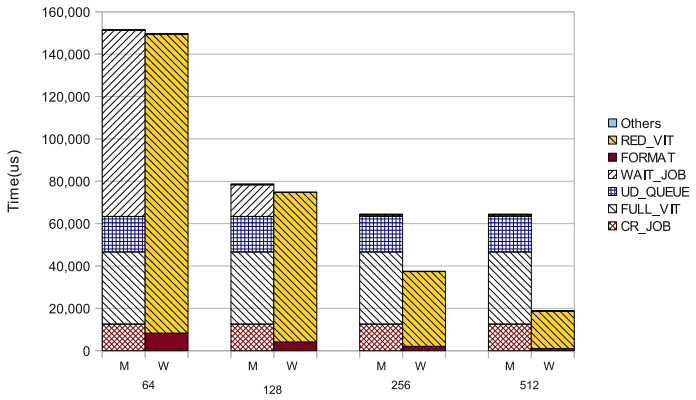


Figure 6.16: Time share of HMMER program phases for a short HMM.

Figure 6.15 shows the performance scalability in the number of cores using *Log-Log* axes. For the short HMM (Figure 6.15(a)), the BASE version can only take advantage of 8 workers for a maximum speedup of $5\times$. M_FORMAT increases maximum speedup to $15\times$ with 16 cores and W_FORMAT increases the number of utilizable cores to 256 to achieve $148\times$ speedup. Figure 6.15(b) shows that for a large HMM, performance scales a bit more (up to $156\times$). This is because a longer HMM makes the parallel portion of work larger.

In Figure 6.16, we plot the cumulative time distribution of phases in the best parallel version (W_FORMAT). This, in order to study the performance saturation that occurs when using more than 256 cores. We look at four scenarios: using 64, 128, 256 and 512 cores. For each case, one bar represents the master and the other one the average behavior of the workers. First of all, we see that on the master side, the WAIT_JOB time gets reduced with more cores that are able to finish the jobs faster. The other phases in the master stay unchanged. On the workers side, most of the time is spent on RED_VIT and FORMAT phases, indicating a high utilization rate. With the increase in the number of cores, phases get reduced proportionally. The most important trend to observe in Figure 6.16 is that the reduction in total execution time (determined by the master), stops at 256 cores due to fixed duration of three phases: CR_JOB, FULL_VIT, UD_QUEUE. These, stay unchanged in all cases as they only depend on the input data and not on the number of cores being used, as in Amdahl's law.

Figure 6.16 also reveals that workers spend most of the time in computing the

Viterbi kernel (and formatting) and do not manage to saturate the memory nor the buses. This was confirmed by running further simulations with different system configurations. These experiments showed that increasing the GDB bandwidth, adding more cache and cache banks, and adding more MICs to increase the memory bandwidth did not change the time it takes the workers to complete all jobs.

6.5 Summary

This chapter has presented a study of a novel manycore architecture targeting the multiple sequence alignment problem in ClustalW and the protein database search in HMMER. A hybrid high-level/cycle-accurate simulator has been used to model a system with up to 1024 cores. We started by characterizing various input datasets and found out that they have a significant influence in the effectiveness of parallelizing ClustalW.

We have simulated a manycore architecture composed of one master processor and 1024 application-specific accelerators for ClustalW-PW. Results show that while high latencies are tolerated by the application, the system bandwidth determines the performance. This is mostly due to the use of DMAs to fetch chunks of data from main memory to the accelerator' SPMs. Simulations demonstrated that a small shared cache, as long as it is partitioned in (four) banks, is able to capture most of the memory traffic and provide the necessary bandwidth. Instead of using main memory to synchronize, we also showed that a realistic latency for the SQ is able to handle all the accelerators and all the available tasks in our already large input set.

On the other hand, the general-purpose cores have much lower throughput than the accelerators and therefore, they do not put pressure on the rest of the system. A minimalistic configuration was able to provide the maximum performance. However, we showed that when using heterogeneous inputs of small to medium size, the distribution of the load gets unbalanced. Estimating the duration of tasks in advance, allowed us to sort them and schedule long ones first. This simple strategy improved performance by 28% for general-purpose cores. However, because the task sorting is performed on a single processor, its overhead may become too big and hence not worth when using accelerators.

Simulations have shown the "minimum" performance gap between general-purpose and accelerator cores for our case study. We make the observation that if implemented in real chips, the power wall will have the effect of fur-

ther increasing the performance gap. This due to the fact that accelerators run at much lower frequencies, while general-purpose cores will be forced to apply frequency scaling, directly affecting performance. Lastly, we estimated that with PA parallelization, the total application speedup would increase from 61.1× to 195× using accelerators.

In this chapter we have also analyzed the performance scalability of several parallel versions of the HMMER software for protein sequence alignment. We started by analyzing HMMERCELL, a Cell port of HMMER. By manually instrumenting the code and looking into the execution traces, we were able to find out the bottlenecks when the application is ported to a larger multicore like SARC. Based on our findings, we proposed two optimized versions that were aimed at alleviating the bottlenecks: holding the database in memory to avoid I/O access and parallelizing the pre-formatting of sequences. Simulation results showed that the M_FORMAT version was up to 3× faster while the W_FORMAT parallelization achieved up to 27× speedup, compared to HMMERCELL. Compared to using a single worker processor, W_FORMAT scaled performance up to 156× with 256 cores. Adding more cores or increasing the memory bandwidth did not improve performance. This was because the system is then bottlenecked by the master processing the FULL_VIT phase and collecting results from all workers. Parallelizing FULL_VIT in a system with larger scratchpad memories is the way to increase scalability further.

We strongly believe that our study bring useful considerations for the design of future manycore systems-on-a-chip targeting bioinformatics workloads.

Note. The content of this chapter is based on the the following papers:

S. Isaza, F. Sanchez, F. Cabarcas, A. Ramirez, G. Gaydadjiev. **Parametrizing Multicore Architectures for Multiple Sequence Alignment**, *Proceedings of the* 8^{th} *ACM International Conference on Computing Frontiers*, pp. 1-10, Ischia, Italy, May 2011.

S. Isaza, E. Houtgast, F. Sanchez, A. Ramirez, G. Gaydadjiev. **Scaling HMMER Performance on Multicore Architectures**. *Proceedings of the* 4^{th} *International Workshop on Multicore Computing Systems, MuCoCoS - IEEE*, pp. 618-623, Seoul, Korea, Jun. 2011.

7

Conclusions and Future Work

In this thesis we have addressed the challenges of performance scalability for bioinformatics applications on current and future multicore architectures. First we introduced a number of concepts in bioinformatics and multicore architectures as necessary background for understanding the remainder of the thesis. Then, we presented the related work and discussed how this thesis improves it. In the following chapters we presented the thesis contributions that have been divided into three parts: analytical modeling of performance scaling, mapping onto the Cell BE and analysis of its limitations, and, forecasting scalability bottlenecks in future manycores. In the following we summarize the work done and present our conclusions. Last, we propose possible directions for future work.

7.1 Summary and Conclusions

When introducing this thesis in Chapter 1, we provided several reasons why our work is pertinent. An exponential growth of biological databases combined with the computational complexity of the algorithms used has created an enormous need for computational power in the bioinformatics field. In parallel to that, the computer architecture domain has shifted to the multicore paradigm mostly due to various technology limitations. In order to sustain, or hopefully improve, the performance growth of bioinformatics applications in the future, their scalability by parallelism must be understood in light of the coming manycore era. This thesis contributes to understand the parallelism and scalability of sequence alignment applications and proposes directions to increase their performance by means of hardware support and program optimizations. Briefly stated, the objectives of this thesis have been defined as: modeling the performance scaling, discovering the strengths and bottlenecks

of contemporary multicores, and, estimating the bottlenecks in future many-core systems, for biosequence alignment applications.

In Chapter 2, we introduced the bioinformatics theory and applications. We briefly described the molecular biology problems that can be solved with computer programs and showed the exponential growth of biological databases that pushes for an increase in processing power. Then, we defined sequence alignment and described in detail the algorithms commonly used to compute them. On the other hand, the multiple sequence alignment scenario was explained along with its differences with pairwise alignment. A description of the applications targeted in this thesis, i.e., ClustalW and HMMER, was also included in this chapter.

Chapter 3 consisted of three parts: the multicore architectures description, the simulators setups and the related work. First, the Cell BE and the SARC architectures were explained, followed by the their corresponding simulation platforms used in this thesis, i.e., CellSim and TaskSim. The last part of the chapter presents other works with respect to ClustalW and HMMER acceleration.

In Chapter 4, we presented an analytical model of HMMER aimed at master-worker parallelization schemes. The proposed model was deduced from program inspection and later compared against execution of HMMERCELL on a real Cell BE processor. Results have shown that the model prediction for M_BUF and W_VIT phases is highly accurate, with only 1.5% and 1.7% error on average. Although M_PP was not accurately estimated by the model, we showed that for realistic test cases it does not affect the overall prediction. The total execution time estimation only had a 2% average error.

These findings are relevant for other bioinformatics applications as well. Most bioinformatics applications contain an abundance of coarse-grained parallelism and the master-worker pattern is a useful strategy to divide the work over multiple cores. For optimal scaling behavior, the master core should be relieved of as many other tasks as possible and control tasks should also be parallelized. In the case of a Cell BE blade, the SPEs could take care of the sequence formatting work in M_BUF. However, parallelizing M_BUF would only speedup its sequence formatting part and the I/O would still be limited by the hardware channels. Later, in Chapter 6, we presented simulations that show the gains obtained from removing each of the two bottlenecks.

In an attempt to have a first hand experience with state-of-the-art multicores, Chapter 5 has presented the mapping and some optimization alternatives of ClustalW, a representative bioinformatics application, targeting Cell BE. We have also presented a qualitative analysis of the architectural shortcomings

identified during this process. Our study revealed various architectural aspects that negatively impact Cell BE performance for bioinformatics workloads. More precisely, the missing HW support for unaligned memory accessesand the programming model appeared to be the most critical. However, the precise behavior of the ClustalW was found to be different when compared to other sequence alignment applications. For instance, SSearch needs a large amount of temporal storage that exceeds the SPE's capacity. This significantly increases the interconnect traffic and it has been shown how the blocking factor affects performance. For ClustalW, we found that it was strongly computationally bound and therefore very sensitive to any inefficiency or improvement in the inner loop of the most intensive kernel (Pairwise Alignment - PW). The last part of Chapter 5 reports the speedups we have achieved (17%) by adding a few simple instructions to the SPE baseline instruction-set. These results have shown that with minimal domain-specific hardware, significant performance improvements can be obtained.

Since ClustalW is a more complex application, we went further and analyzed the second most computationally intensive part (Progressive Alignment - PA) that becomes the bottleneck when PW is optimized and parallelized. We have carefully profiled and analyzed the PA behavior and parallelized the *prfscore* computations so that an arbitrary number of SPEs can be used. Results have shown that the parallel *prfscore* version scales close to linearly with both the number of SPEs and the number of input sequences. On the other hand, the FWD/BWD part gets affected by the limited task-level parallelism and the slow matrix reorganization process on the PPE. Cores with stride memory access capabilities would be able to tackle this bottleneck by reducing the number of required load operations. Medium to fine-grain parallelization for PA does not greatly scale performance with adding adding more cores as only *prfscore* allows scaling. FWD/BWD remains fixed to use two cores and other sequential parts in the PPE (like *calc-inputs* and *ptracepath*) become the bottlenecks. These phases suffer from the poor PPE ability to extract ILP and from the lack of parallelization. We also found that as the input size grows, *calc-inputs* dominates the PA time share. This clearly shows that it should be the next function to parallelize in ClustalW-PA.

In Chapter 6, we have used a hybrid high-level/cycle-accurate simulator to model a future system with up to 1024 cores. For ClustalW, the cores have been configured as accelerators for the PW phase. Results have shown show that while high latencies are tolerated by the application, the system bandwidth determines the performance. This is mostly due to the use of DMAs to fetch chunks of data from main memory to the accelerator' SPMs. Simulations have

demonstrated that a small shared cache, as long as it is partitioned in (four) banks, is able to capture most of the memory traffic and provides the necessary bandwidth. Instead of using main memory to synchronize, we also showed that a realistic latency for the SQ is able to handle all the accelerators and all the available tasks in our already large input set.

When using general-purpose processors instead of accelerators, we found a load balancing issue for certain types of input data. Then, we demonstrated that by estimating the duration of tasks in advance, we could sort them and schedule long ones first. This simple strategy improved performance by 28% for general-purpose cores. However, because the task sorting is performed on a single processor, its overhead becomes too big and, hence, not worth when using accelerators.

Simulations have shown the "minimum" performance gap between general-purpose and accelerator cores for our case study. We have made the observation that if implemented in real chips, the power wall will have the effect of further increasing the performance gap. This is due to the fact that accelerators run at much lower frequencies, while general-purpose cores will be forced to apply frequency scaling, directly affecting performance.

We have found that inputs sequences with different characteristics have a significantly different impact of ClustalW parallelism and consequently on performance. Lastly, we estimated that with PA parallelization and the input set chosen, the total application speedup would increase from 61.1× to 195× using accelerators. The best configuration of the manycore architecture achieved 720× speedup for ClustalW-PW.

In Chapter 6 we also analyzed the performance scalability of several parallel versions of HMMER. Using HMMERCELL as a baseline, we proposed two optimized versions that were aimed at alleviating the bottlenecks found in Chapter 4: holding the database in memory to avoid the I/O access and parallelizing the pre-formatting of sequences. Simulation results showed that the M_FORMAT version was up to 3× faster while the W_FORMAT parallelization achieved up to 27× speedup, compared to HMMERCELL. Compared to a system with a single worker processor, W_FORMAT scaled performance up to 156× with 256 cores. Adding more cores or increasing the memory bandwidth did not improve performance. This was because the system is then bottlenecked by the master processing the FULL_VIT phase and collecting results from all workers. Parallelizing FULL_VIT in a system with larger scratchpad memories is one way to increase scalability further.

We strongly believe that the collection of experiments, proposals and discus-

sions in this thesis provide very useful insights into the design of future manycore systems targeting bioinformatics workloads.

7.2 Open Issues and Future Directions

The research work presented in this thesis is not complete mainly because of two reasons: the large amount of bioinformatics applications that need to be studied and the complexity of setting up experiments involving large applications and complex architectures. In particular, dealing with parallel programming is widely recognized as hard and time consuming. This very last section of the thesis aims at outlining a few research directions and particular experiments that can complement this thesis.

The performance model presented in Chapter 4 was shown to be highly accurate for the chosen case-study. However, it should be noticed that the model can be further improved by incorporating the saturation effects of the interconnect. Based on the data traffic and available bandwidth, an analytical model could be proposed and added to the existing model. For applications in which the computation-to-communication ratio is not as high as in the ones studied in this thesis, such an improvement is critical to obtain an accurate model. On the other hand, validating the model against more parallel bioinformatics applications that also use the master-worker strategy (e.g. ClustalW and HMMER) is an interesting additional exercise that can demonstrate the usability of the proposed model.

In Chapter 5, we proposed a few new instructions for the SPE instruction-set and reported speedups based on experimental results. Since the proposed instructions deal with the core computations in the inner part of a dynamic programming kernel, they can also be used in kernels with the same type of processing such as SSearch, HMMER and even in ClustalW-PA. Incorporating our proposed instructions into those other functions and applications, and, proposing additional instructions is part of our ongoing work.

The methodology used in Chapter 6 for studying the performance scalability in manycores can be extended to other kernels and applications. In particular, an interesting study for the future work is to investigate the scalability of ClustalW-PA, Clustal Omega and HMMER3. On the other hand, in order to respect power limitations, future manycore systems will have to use frequency scaling very often. One interesting research direction is the development of simulation models, as extensions of the one used in Chapter 6 for instance, to incorporate the impact of frequency scaling on performance scalability.

Bibliography

[1] CellSim: Modular Simulator for Heterogeneous Multicore Processor Architectures. http://pcsostres.ac.upc.edu/cellsim/doku.php.

[2] European Bioinformatics Institute, ClustalW Web Server. http://www.ebi.ac.uk/Tools/clustalw/.

[3] European Bioinformatics Institute, ClustalW2 Web Server. http://www.ebi.ac.uk/Tools/clustalw2/.

[4] European Bioinformatics Institute, FASTA Web Server. http://www.ebi.ac.uk/Tools/sss/fasta/.

[5] FASTA Web Server at University of Virginia. http://fasta.bioch.virginia.edu.

[6] HMMER User's Guide. ftp://selab.janelia.org/pub/software/hmmer3/3.0/Userguide.pdf.

[7] Howard Hughes Medical Institute, HMMER Web Site. http://hmmer.janelia.org/.

[8] National Center for Biotechnology Information, BLAST Web Server. http://blast.ncbi.nlm.nih.gov.

[9] National Institute of Genetics, DNA Data Bank of Japan. http://www.ddbj.nig.ac.jp/.

[10] SARC Project. http://www.sarc-ip.org/.

[11] The Human Genome Project. http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml.

[12] UNISIM. http://unisim.org/site/.

[13] Barcelona Supercomputing Center, Paraver, Sep. 2010. http://www.bsc.es/paraver.

[14] Convey Computer Announces Record-Breaking Smith-Waterman Acceleration of 172X, May 2010. http://www.conveycomputer.com/Resources/.

[15] Swiss Institute of Bioinformatics, UniProtKB/Swiss-Prot Protein Knowledgebase Release 2010-11 Statistics, November 2010. http://www.expasy.org/sprot/relnotes/relstat.html.

[16] The European Molecular Biology Laboratory, Nucleotide Sequence Database, 2010. www.ebi.ac.uk/embl/.

[17] Barcelona Supercomputing Center, Extrae tool user's Guide, 2011. http://www.bsc.es/ssl/apps/performanceTools/files/docs/extrae-2.1.1-userguide.pdf.

[18] UniProt: Universal Protein Resource, 2011. www.uniprot.org.

[19] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[20] S. F. Altschul, T.L, A. Schffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Serach Programs. *Nucleic acids research*, 25:3389–3402, 1997.

[21] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485. ACM, 1967.

[22] D. Bader, Y. Li, T. Li, and V. Sachdeva. BioPerf: A Benchmark Suite to Evaluate High-Performance Computer Architecture on Bioinformatics Applications. In *Proceedings of the IEEE International Workload Characterization Symposium.*, pages 163 – 173, Oct. 2005.

[23] P. Bellens, J. M. Perez, R. M. Badia, and J. Labarta. CellSs: a programming model for the cell BE architecture. In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 1–10, 2006.

[24] K. Benkrid, Y. Liu, and A. Benkrid. A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment. *IEEE Trans. on VLSI Systems*, 17(4):561 –570, 2009.

[25] D. Benson, I. Karsch, D. Lipman, J. Ostell, B. Rapp, and D. Wheeler. GenBank. *Nucleic acids research*, 28(1):15–18, 2000.

[26] K. bin Li. ClustalW-MPI: ClustalW Analysis Using Distributed and Parallel Computing. *Bioinformatics*, 19:1585–1586, 2003.

[27] F. Blagojevic, A. Stamatakis, C. Antonopoulos, and D. Nikolopoulos. RAxML-Cell: Parallel Phylogenetic Tree Inference on the Cell Broadband Engine. In *Proceedings of the 21st IEEE/ACM International Parallel and Distributed Processing Symposium*, pages 1–10, March 2007.

[28] M. Brudno, R. Steinkamp, and B. Morgenstern. The chaos/dialign www server for multiple alignment of genomic sequences. *Nucleic Acids Research*, 32(suppl 2):W41–W44, 2004.

[29] K. Chaichoompu, S. Kittitornkun, and S. Tongsima. MT-ClustalW: multithreading multiple sequence alignment. In *IPDPS 06: Proceedings of the 20th International Parallel and Distributed Processing Symposium*, pages 1–6, 2006.

[30] J. Cheetham, F. K. H. A. Dehne, S. Pitre, A. Rau-Chaplin, and P. J. Taillon. Parallel CLUSTAL W for PC Clusters. In *Proceedings of International Conference on Computational Science and Its Applications (ICCSA)*, pages 300–309, 2003.

[31] Y. Cheng. Optimizing ClustalW1.83 using AltiVec Implementation. Technical report, 2006.

[32] F. Crick. Central Dogma of Molecular Biology. *Nature*, 227:561–563, August 1970.

[33] L. Darden and J. Tabery. Molecular Biology. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2010 edition, 2010.

[34] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

[35] A. Di Blas, D. M. Dahle, M. Diekhans, L. Grate, J. Hirschberg, K. Karplus, H. Keller, M. Kendrick, F. J. Mesa-Martinez, D. Pease, E. Rice, A. Schultz, D. Speck, and R. Hughey. The ucsc kestrel parallel processor. *IEEE Trans. Parallel Distrib. Syst.*, 16:80–92, January 2005.

[36] R. Durbin, S. R. Eddy, A. Krogh, and G. J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, ISBN 0-521-62971-3, 1998.

[37] S. R. Eddy. Profile Hidden Markov Models. *Bioinformatics*, 14(9):755–763, 1998.

[38] R. Edgar. MUSCLE: Multiple Sequence Alignment Software. http://www.drive5.com/muscle/.

[39] R. Edgar. MUSCLE: a Multiple Sequence Alignment Method with Reduced Time and Space Complexity. *BMC Bioinformatics*, 5(1):113+, August 2004.

[40] L. Eisenberg. The Tree of Life. http://www.evogeneao.com/tree.html.

[41] I. Elias. Settling the Intractability of Multiple Alignment. *Journal of Computational Biology*, 13(7):1323–1339, 2006.

[42] P. Faes, B. Minnaert, M. Christiaens, E. Bonnet, Y. Saeys, D. Stroobandt, and Y. V. de Peer. Scalable Hardware Accelerator for Comparing DNA and Protein Sequences. In *Proceedings of the First International Conference on Scalable Information Systems*, pages 1–6, May 2006.

[43] M. Farrar. Striped Smith-Waterman Speeds Database Searches Six Times over other SIMD Implementations. *Bioinformatics (Oxford, England)*, 23(2):156–161, January 2007.

[44] M. Farrar. Optimizing Smith-Waterman for the Cell Broadband Engine. http://sites.google.com/site/farrarmichael/smith-watermanfortheibmcellbe, 2008.

[45] N. C. for Biotechnology Information. MSA Package for Multiple Sequence Alignment. http://www.ncbi.nlm.nih.gov/CBBresearch/Schaffer/msa.html.

[46] J. P. Gogarten, W. F. Doolittle, and J. G. Lawrence. Prokaryotic Evolution in Light of Gene Transfer. *Molecular Biology and Evolution*, 19(12):2226–2238, 2002.

[47] M. Gokhale and P. Graham. *Reconfigurable Computing*. Springer, Dordrecht, The Netherlands, ISBN 978-0-378-26105-8, 2005.

[48] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *IEEE Micro*, pages 10–24, 2006.

[49] S. A. Gupta SK, Kececioglu J. Improving the practical space and time efficiency of the shortest-paths Approach to sum-of-pairs multiple sequence alignment. *Journal of Computational Biology*, 2:459–472, 1995.

[50] J. Henikoff, S. Henikoff and S. Pietrokovski. Blocks+: A Non-redundant Database of Protein Alignment Blocks Derived from Multiple Compilations. *Bioinformatics*, 15(6):471–479, 1999.

[51] S. Henikoff and J. Henikoff. Amino Acid Substitution Matrices from Protein Blocks. *Proceding in Natural Academic Science*, 89, 1992.

[52] S. Isaza, F. Sanchez, G. Gaydadjiev, A. Ramirez, and M. Valero. Preliminary Analysis of the Cell BE Processor Limitations for Sequence Alignment Applications. In *Proceedings of the 8th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS '08, pages 53–64, 2008.

[53] S. Isaza, F. Sanchez, G. Gaydadjiev, A. Ramirez, and M. Valero. Scalability Analysis of Progressive Alignment on a Multicore. In *Proc. of IEEE Int. Workshop on Multicore Computing Systems*, pages 889–894, February 2010.

[54] K. Jiang, O. Thorsen, A. Peters, B. Smith, and C. P. Sosa. An Efficient Parallel Implementation of the Hidden Markov Methods for Genomic Sequence-Search on a Massively Parallel System. *IEEE Trans. Parallel Distrib. Syst.*, 19:15–23, January 2008.

[55] J. Kahle, M. Day, H. Hofstee, C. Johns, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM Systems Journal*, 49(4/5):589–604, 2005.

[56] G. Karp. *Cell and Molecular Biology: Concepts and Experiments*. Willey, fith edition, 2008.

[57] A. Krogh, M. Brown, I. Mian, K. Sjlander, and D. Haussler. Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

[58] L. Ligowski and W. Rudnicki. An Efficient Implementation of Smith Waterman Algorithm on GPU using CUDA, for Massively Parallel Scanning of Sequence Databases. In *IPDPS '09: Proceedings of the 2009 IEEE International Parallel&Distributed Processing Symposium*, pages 1–8, 2009.

[59] D. Lipman, S. Altschul, and J. Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences*, 86:4412–4415, 1989.

[60] D. Lipman and W. Pearson. Rapid and Sensitive Protein Similarity Searches. *Science*, 227(4693):1435–1441, 1985.

[61] P. Liu and A. Hemani. A Coarse Grain Reconfigurable Architecture for Sequence Alignment Problems in Bioinformatics. In *SASP: 8th IEEE Symposium on Application Specific Processors*, pages 50–57, 2010.

[62] J. Lu, M. Perrone, K. Albayraktaroglu, and M. Franklin. HMMer-Cell: High Performance Protein Profile Searching on the Cell/B.E. Processor. In *ISPASS '08: IEEE International Symposium on Performance Analysis of Systems and software*, pages 223 –232, 2008.

[63] C. Meenderinck. *Improving the Scalability of Multicore Systems: With a Focus on H.264 Video Decoding*. PhD thesis, Delft University of Technology, 2010.

[64] E. W. Meyers and W. Miller. Optimal alignments in linear space. *Bioinformatics*, 4:11–17, 1988.

[65] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[66] U. of California Santa Cruz. SAM. http://compbio.soe.ucsc.edu/sam.html.

[67] U. of Gottingen. CHAOS/DIALIGN. http://dialign.gobics.de/.

[68] T. Oliver, B. Schmidt, and D. Maskell. Hyper Customized Processors for Bio-Sequence Database scanning on FPGAs. In *Proceedings of 17th International Symposium on Field Programmable Gate Arrays*, pages 229–237, February 2005.

[69] T. Oliver, L. Yeow, and Schmidt. High performance database searching with hmmer on FPGAs. In *Proceedings of the 6th Workshop on High Performance Computational Biology*, pages 1–7, February 2007.

[70] T. Oliver, L. Yeow, and B. Schmidt. Integrating FPGA Acceleration into HMMER. *Parallel Computing*, 34(11):681–691, 2008.

[71] W. R. Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11:635–650, 1991.

[72] W. R. Pearson and D. J. Lipman. Improved Tools for Biological Sequence Comparison. In *Proc. Natl. Acad. Sci (USA).*, pages 2444–2448, April 1988.

[73] M. F. Perutz, M. H. F. Wilkins, and J. D. Watson. DNA Helix. *Science*, 164:1537–1539, June 1969.

[74] F. Petrini, G. Fossum, J. Fernandez, A. Varbanescu, M. Kistler, and M. Perrone. Multicore Surprises: Lessons Learned from Optimizing Sweep3D on the CellBE. pages 1–10, 2007.

[75] A. Ramirez, F. Cabarcas, B. H. H. Juurlink, M. Alvarez, F. Sanchez, A. Azevedo, C. Meenderinck, C. B. Ciobanu, S. Isaza, and G. Gaydadjiev. The SARC Architecture. *Micro, IEEE*, 30(5):16 –29, sept.-oct. 2010.

[76] B. Rekapalli, C. Halloy, and I. Zhulin. HSP-HMMER: A Tool for Protein Domain Identification on a Large Scale. In *ACM Symposium on Applied Computing*, pages 766–770, 2009.

[77] A. Rico, F. Cabarcas, A. Quesada, M. Pavlovic, A. Vega, C. Villavieja, Y. Etsion, and A. Ramirez. Scalable Simulation of Decoupled Accelerator Architectures. UPC-DAC-RR-2010-10. Technical report, Technical University of Catalonia, 2010.

[78] T. Rognes. *Rapid and Sensitive Methods for Protein Sequence Comparison and Database Searching*. PhD thesis, Institue of Medical Microbiology, University of Oslo, 2000.

[79] V. Sachdeva, M. Kistler, E. Speight, Tzeng, and T.H.K. Exploring the viability of the Cell Broadband Engine for bioinformatics applications. In *Proceedings of the 6th Workshop on High Performance Computational Biology*, pages 1–8, 2007.

[80] V. Sachdeva, M. Kistler, E. Speight, and T.-H. K. Tzeng. Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications. *Parallel Computing*, 34(11):616–626, 2008.

[81] N. Saitou and M. Nei. The Neighbor-joining Method: A New Method for Reconstructing Phylogenetics Trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[82] F. Sanchez. *Exploiting Multiple Levels of Parallelism in Bioinformatics Applications*. PhD thesis, Technical University of Catalonia, 2011.

[83] F. Sanchez, F. Cabarcas, A. Ramirez, and M. Valero. Scalable Multicore Architectures for Long DNA Sequence Comparison. *Concurrency and Computation: Practice and Experience*, 2011.

[84] F. Sanchez, E. Salami, A. Ramirez, and M. Valero. Performance Analysis of Sequence Alignment Applications. pages 51–60. IEEE Computer Society, 2006.

[85] D. E. Shaw, M. Deneroff, R. Dror, J. Kuskin, R. Larson, J. Salmon, C. Young, B. Batson, K. Bowers, J. Chao, M. Eastwood, J. Gagliardo, J. Grossman, R. Ho, D. Ierardi, I. Kolossvry, J. Klepeis, T. Layman, C. McLeavey, M. Moraes, R. Mueller, Y. S. E. Priest, J. Spengler, M. Theobald, B. Towles, and S. Wang. Anton: A Special-Purpose Machine for Molecular Dynamics Simulation. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 1–12, June 2007.

[86] S. Smith and J. Frenzel. Bioinformatics Application of a Scalable Supercomputer-on-chip Architecture. 1:385–391, 2003. Proceedings of the International Conference on Parallel and Distributed Processing Techniques.

[87] T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[88] E. L. Sonnhammer, S. R. Eddy, and R. Durbin. Pfam: a Comprehensive Database of Protein Domain Families based on Seed Alignments. *Proteins*, 28(3):405–420, 1997.

[89] U. Srinivasan, P.-S. Chen, Q. Diao, C.-C. Lim, E. Li, Y. Chen, R. Ju, and Y. Zhang. Characterization and Analysis of HMMER and SVM-RFE Parallel Bioinformatics Applications. In *IEEE International Symposium on Workload Characterization*, pages 87 – 98, 2005.

[90] J. Thompson, D. Higgins, and T. Gibson. CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-specific Gap Penalties and Weight Matrix Choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[91] J. Thompson, D. Higgins, and T. Gibson. CLUSTAL W and Clustal X version 2.0. *Bioinformatics*, 23:2947–2948, 2007.

[92] H. Vandierendonck, S. Rul, M. Questier, and K. D. Bosschere. Experiences with Parallelizing a Bio-informatics Program on the Cell BE. In *Proceedings of the 3th International Conference on High Performance and Embedded Architectures and Compilers*, pages 161–175, 2008.

[93] H. Vandierendonck, S. Rul, M. Questier, and K. D. Bosschere. Accelerating Multiple Sequence Alignment with the Cell BE Processor. *The Computer Journal*, 53(6):814–826, 2010.

[94] A. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[95] J. Walters, X. Meng, V. Chaudhary, T. Oliver, L. Yeow, D. Nathan, B. Schmidt, and J. Landman. MPI-HMMER-Boost: Distributed FPGA Acceleration. *Journal of VLSI Signal Processing*, pages 223–238, 2007.

[96] J. Walters, B. Qudah, and V. Chaudhary. Accelerating the HMMER Sequence Analysis Suite using Conventional Processors. In *AINA '06: International Conference on Advanced Information Networking and Applications*, page 6 pp., 2006.

[97] J. P. Walters, R. Darole, and V. Chaudhary. Improving MPI-HMMER's Scalability with Parallel I/O. In *IPDPS '09: IEEE International Symposium on Parallel & Distributed Processing*, pages 1 –11, 2009.

[98] L. Wang and T. Jiang. On the Complexity of Multiple Sequence Alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[99] J. Watson and F. Crick. Molecular structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, 171:737–738, April 1953.

List of Publications

International Journals

1. S. Isaza, E. Houtgast, G. Gaydadjiev. **Sequence Alignment Application Model for Multi- and Manycore Architectures**. *International Journal on Information Technologies and Security*, ISSN 1313-8251, pp. 3-20, 2011.

2. A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez, F. Sanchez, A. Azevedo, C. Meenderinck, C. Ciobanu, S. Isaza, G. Gaydadjiev. **The SARC Architecture**. *Micro, IEEE,* 30(5):16-29, Sep.-Oct. 2010.

Peer-reviewed International Conferences and Workshops

1. S. Isaza, E. Houtgast, G. Gaydadjiev. **HMMER Performance Model for Multicore Architectures**. *Proceedings of the* 14^{th} *Euromicro Conference on Digital System Design, DSD - IEEE*, pp. 257-261, Oulu, Finland, Aug. 2011.

2. S. Isaza, F. Sanchez, F. Cabarcas, A. Ramirez, G. Gaydadjiev. **Parametrizing Multicore Architectures for Multiple Sequence Alignment**, *Proceedings of the* 8^{th} *ACM International Conference on Computing Frontiers*, pp. 1-10, Ischia, Italy, May 2011.

3. S. Isaza, E. Houtgast, F. Sanchez, A. Ramirez, G. Gaydadjiev. **Scaling HMMER Performance on Multicore Architectures**. *Proceedings of the* 4^{th} *International Workshop on Multicore Computing Systems, MuCoCoS - IEEE*, pp. 618-623, Seoul, South Korea, Jun. 2011.

4. S. Isaza, F. Sanchez, G. Gaydadjiev, A. Ramirez, M. Valero. **Scalability Analysis of Progressive Alignment on a Multicore**. *Proceedings of the* 3^{th} *International Workshop on Multicore Computing Systems, MuCoCoS - IEEE*, pp.889-894, Krakow, Poland, Feb. 2010.

5. S. Isaza, F. Sanchez, G. Gaydadjiev, A.Ramirez, M. Valero. **Preliminary Analysis of Cell BE Processor Limitations for Sequence Alignment Applications**. *Proceedings of the* 8^{th} *International Symposium on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS - Springer LNCS*, pp. 53-64, Samos, Greece, Jul. 2008.

Other Workshops

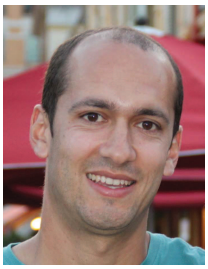
1. S. Isaza, G. Gaydadjiev. **Bioinformatics Specific Cell BE ISA Extensions**. *Proceedings of the Workshop on Circuits, Systems and Signal Processing, ProRISC*, Veldhoven, The Netherlands, Nov. 2008.

2. S. Isaza, G. Gaydadjiev. **Limitations of Current Multicore Architectures for Bioinformatic Applications - A Case Study on the Cell BE**. *Proceedings of the* 4^{th} *ACACES Workshop*, L'Aquila, Italy, Jul. 2008.

Samenvatting

In dit proefschrift richten we ons op de uitdagingen in schaalbaarheid van bio-informatica toepassingen op multicore-architecturen. In het bijzonder richten we ons op sequence alignment, een van de fundamentele taken in de bioinformatica. Als gevolg van de exponentile groei van biologische databases en als gevolg van de computationele complexiteit van de gebruikte algoritmes, is gebruik van high performance computing-systemen vereist. Recentelijk heeft er in de computerarchitectuur een verschuiving plaatsgevonden naar het multicore paradigma, in een poging om de verwachte prestatiegroei van processoren te kunnen doorzetten, iets waarvoor single-core architecturen niet meer kunnen zorgen. Hoewel multicore-architecturen goed toegerust zijn om parallellisme op taak- en data-niveau zoals deze aanwezig zijn in bioinformatica workloads te exploiteren, vergt efficint gebruik van systemen met honderden kernen een diepe kennis van de applicaties en de bijzonderheden in architectuur. Dit proefschrift presenteert een studie van twee sequence alignment toepassingen die worden gemodelleerd, gekarakteriseerd, gemapped en geoptimaliseerd op twee multicore architecturen. We maken gebruik van de Cell BE processor en de SARC architectuur, de laatste is ontwikkeld binnen het project waar ook dit proefschrift deel van uit maakt. De twee geanalyseerde toepassingen, te weten HMMER en ClustalW, worden gebruikt voor pairwise alignment en multiple sequence alignment. Allereerst stellen we een analytisch model voor, om de prestaties van de met behulp van het master-worker paradigma geparalleliseerde applicaties te kunnen voorspellen. Voor onze case study en voor de validatie van het model maken we gebruik van HMMER en de Cell BE-processor. De resultaten tonen de hoge mate van nauwkeurigheid van het model en brengen ook het schalingsgedrag van de verschillende fasen van de applicatie in kaart. Vervolgens onderzoeken we de optimale mapping van ClustalW op de Cell BE, identificeren we een aantal beperkingen in deze architectuur en stellen we een aantal instructie-set extensies voor om de ClustalW kernel te versnellen. Tenslotte bestuderen we de schaalbaarheid van ClustalW en HMMER op de SARC architectuur, waarbij gebruik wordt gemaakt van maximaal duizend cores. Hierbij wordt ook de impact van de verschillende input types op de prestaties van ClustalW onderzocht.

About the Author



Sebastián Isaza Ramírez was born on November 10, 1979 in Medellin, Colombia. He studied Electronic Engineering (5-year program) at the University of Antioquia, Medellin, Colombia, and graduated in 2004. His undergraduate thesis was on the implementation of internet protocols for the public pay-phones designed by Celsa S.A, where he also worked part-time, for a year, after concluding his thesis project. While at the University of Antioquia, he was teaching and research assistant from 2002 to 2004 in the Digital Circuits Labs and the Microelectronics Group. In 2004 he received a scholarship from the ALaRI Institute in Switzerland in order to conduct his master studies. In 2006 he graduated from the ALaRI Master of Science program in Embedded Systems Design at the Faculty of Informatics, University of Lugano, Switzerland. His MSc. thesis was supervised by Dr. Laura Pozzi and dealt with instruction-set extensions for multimedia processors.

In 2006 Sebastian met Prof. Stamatis Vassiliadis, late chair of the Computer Engineering Laboratory, and was accepted to pursue his PhD studies within his group at Delft University of Technology, the Netherlands. After the decease of Stamatis in 2007, Sebastian continued working under the supervision of Dr. Georgi Gaydadjiev in the context of the SARC project and with a focus on scalable multicore architectures for bioinformatics applications. Within the SARC project, with the participation of 16 European partners, most of Sebastian's work was done in close collaboration with the Barcelona Supercomputing Center and the Technical University of Catalonia, where he paid two short-term visits in 2007 and 2008. Sebastian has been student member of the HiPEAC Network of Excellence and participated in two collaboration grants from this network. He has been reviewer in more than twenty international conferences in the computer architecture domain and was part of the organization of FPL 2007, SAMOS 2007, 2008, 2009, 2010 and ICCD 2010. Sebastian is interested in computer architecture in general, multicore processors, microarchitecture, parallel computing and programming, simulation techniques, reconfigurable computing, bioinformatics and bioengineering. He enjoys and appreciates teaching very much.