Olaf René Birkeland

Searching large data volumes with MISD processing

Thesis for the degree doctor philosophiae

Trondheim, September 2008

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU

NTNU
Norwegian University of Science and Technology

Thesis for the degree doctor philosophiae

Faculty of Information Technology, Mathematics and
Electrical Engineering
Department of Computer and Information Science

ISBN 978-82-471-1099-7 (printed version)
ISBN 978-82-471-1100-0 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2008:207

Printed by NTNU-trykk

Abstract

Historically, supercomputing has focused on number crunching. Non-numeric applications, such as information retrieval and analysis, have to a lesser extent been able to exploit the inherent resources of supercomputers. This thesis presents the results from the development of a novel multiple instruction, single data (MISD) architecture, targeting evaluation of complex queries in large data volumes. For such applications, this architecture provides a better price versus performance ratio, better use of the available memory bandwidth, lower power consumption, as well as linear scalability.

The core element of this technology is the Pattern Matching Chip (PMC). Each chip provides 1024 processing elements, with an accumulated performance of 10^{11} operations per second. Multiple chips can be run in parallel with linear scalability, either within one computer, or in larger clusters. Up to half a million processing elements have been used in parallel in this project, providing $5 \cdot 10^{13}$ operations per second at 48 GB per second data throughput, in a unit smaller than one cubic meter. Even larger systems can be constructed, still with linear scalability.

Through the novelty of this hardware architecture, the performance gained has enabled information processing in a way that would have been cost prohibitive with traditional computers. Such processing has demonstrated the capability of finding nuggets of valuable data in large and complex data volumes. The main effort — thus also the most important practical results — has been in bioinformatics. However, the technology has applicability in numerous other data mining applications.

Preface

THIS DISSERTATION is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree of *Doctor philosophiae*. The dissertation contains seven papers, which will be referenced as Paper I through Paper VII, and one patent (see List of Papers on page xiii and List of Patents on page xv).

Acknowledgement

Fortunately, my solitude position as a hardware developer within the Interagon team has been complemented by extremely skillful software developers. Without them, the result would only be a large numbers of transistors etched into silicon. Within Interagon, I would especially like to thank Magnar Nedland for his excellent analytical capabilities, his depth of hardware competence, and his ability to produce flawless software; Pål Sætrom as the mastermind in using genetic programming to tap the resources of the PMC; and Ola Snøve for moving to Oslo. Without the inspiration and guidance from Ola, I would not have started writing this thesis.

The Interagon team started as a project group within Fast Search & Transfer (FAST) in 1998, backed by professor Arne Halaas at NTNU. Arne has a long experience in data filtering that substantially accelerated our development. He has provided valuable suggestions and guidance throughout the entire project. FAST and Interagon should both be honored for having stamina in funding this research over such a long period of time. Interagon has been backed by NTNU's bioinformatics platform in the national functional genomics programme (FUGE). Furthermore, professor Lasse Natvig at NTNU deserves a large credit for reading through my manuscript and providing numerous valuable comments on the content.

I would also thank my co-authors for their contributions, including Ståle H. Fjeldstad, Thomas Grünfeld, Arne Halaas, Håkon Humberset, Magnar Nedland, Ola Snøve Jr., Børge Svingen, and Pål Sætrom.

Finally, I would like give my greatest gratitude to my entire family, especially my wife Anne Ma and our children René and Ingrid, for their love and support. You demonstrate that working as a team is a lot more productive — not just when developing computer technology.

Olaf René Birkeland
Oslo, April 25th 2008

Contents

Preface **iii**

Contents **v**

List of Figures **ix**

List of Tables **xi**

List of Papers **xiii**

List of Patents **xv**

1 Introduction **1**
1.1 Aim of Study 1
1.2 Thesis structure 2
1.3 Research questions 3
1.4 Research methods 4
1.5 Contributions 4
1.6 Paper Abstracts 5
1.7 On joint authorship 10
1.8 Supplementary material 11

2 Hardware aware optimizations in data search applications **13**
2.1 Background 14
2.1.1 The evolution of the x86 processor 14
2.1.2 The evolution of memory technology 18
2.2 Sorting large arrays 19
2.2.1 Characteristics of search application at hand 20
2.2.2 Rearranging memory references 20
2.2.3 In-place radix sort 21
2.2.4 Algorithm complexity 24
2.2.5 Measuring the sorting throughput 26

2.2.6 Remarks on practical aspects . . . 28
2.3 Counting the number of 1-bits . . . 28
2.3.1 The naive approach . . . 29
2.3.2 Using a lookup table . . . 29
2.3.3 Using the full register width . . . 30
2.3.4 Implementation . . . 32
2.4 Comments . . . 35

3 Sequential data processing — in parallel 37
3.1 When indices fall short . . . 37
3.2 Pattern matching without indices . . . 39
3.2.1 Dynamic programming . . . 39
3.2.2 Finite state machines . . . 40
3.2.3 Limitations with dynamic programming and automata 41
3.3 Sequential data processing . . . 42
3.4 Improving speed by parallel execution . . . 42
3.5 Sequential access is aligned with technology trends . . . 43

4 The Pattern Matching Chip 45
4.1 Introduction . . . 45
4.2 Existing technologies . . . 46
4.3 Design goals . . . 48
4.3.1 Linearly scalable architecture . . . 48
4.3.2 No preprocessing before analyzing data . . . 48
4.4 Previous architectures . . . 49
4.4.1 The F and H-matrices . . . 49
4.4.2 The MS160 . . . 50
4.5 Implementation choices for the PMC . . . 52
4.5.1 Increasing parallelism . . . 53
4.5.2 Maintaining high data throughput . . . 54
4.5.3 Balancing memory volume with processing power . 56
4.5.4 Memory interface . . . 56
4.5.5 Data distribution . . . 56
4.5.6 Processing elements . . . 59
4.5.7 Result processing . . . 60
4.5.8 System interface . . . 60
4.5.9 Query configuration . . . 61
4.5.10 Scalability . . . 61
4.6 Designing the PMC . . . 62
4.6.1 Extensive acceptance testing . . . 62
4.6.2 Hardware and software co-development . . . 63

4.6.3 ASIC and hardware system co-development 65
4.6.4 Final ASIC and full scale system tests 66
4.7 Programming model 67
4.7.1 Interagon Query Language (IQL) 68
4.7.2 Code compilation and mapping 69
4.7.3 Multi-threaded execution 69
4.8 Distance metrics 71
4.8.1 One dimensional data 71
4.8.2 Multidimensional data 71
4.8.3 PMC distance metrics avoid expensive arithmetics . . 74
4.9 Results 76

5 Applications for the Pattern Matching Chip 79
5.1 Introduction 79
5.2 Characteristics of suitable applications 80
5.3 String data mining 82
5.3.1 Bioinformatics 83
5.3.2 Digital network communication 85
5.3.3 Written text 85
5.4 Vector data processing 86
5.5 Genetic programming 87
5.5.1 Methodology and issues with genetic programming . 87
5.5.2 Molecular biology 88
5.5.3 Financial fraud 88
5.5.4 Seismic processing 89
5.6 Commonalities across application areas 90

6 Evaluation of results 93
6.1 Summary of findings 93
6.1.1 Better usage of available memory bandwidth 93
6.1.2 Scalable parallel architecture 94
6.1.3 Low cost — high performance 94
6.2 Propositions to the research questions 95
6.3 Evaluation of the contributions 97
6.3.1 Summary of contributions 97
6.3.2 Comparison with state-of-the-art 97
6.3.3 Discussion 97
6.4 Further work 98
6.4.1 Context save and restore 98
6.4.2 Spatial awareness beyond 1D 99

Glossary **101**

Bibliography **107**

Papers **121**

List of Figures

1.1 Relations between papers in this thesis 6

2.1 Block diagram of the Intel 8086 processor 15
2.2 Block diagram of the Intel Pentium4 (Prescott) processor . . 16
2.3 Historic development of the x86-family of CPUs 17
2.4 Radix sort algorithm 21
2.5 In-place radix sort example 23
2.6 In-place radix sort algorithm 25
2.7 In-place radix sort scalability 27
2.8 Half-adder 30
2.9 Adding four bits with half-adders 31
2.10 Implementation of parallel bit-serial bit counting 33

3.1 NFA state machine accepting the pattern *ab?c+*. 40
3.2 DFA state machine accepting the pattern *ab?c+*. 40

4.1 MS160 block diagram 50
4.2 High level PMC block diagram 53
4.3 Data distribution for a small query 57
4.4 Detail level PMC block diagram 58
4.5 PMC search core data distribution 58
4.6 PMC search core result processing 59
4.7 PMC FPGA test jig 64
4.8 PMC ASIC test jig 67
4.9 Mapping of a siRNA query 70
4.10 Multithreading of query evaluation 70
4.11 2D distance decomposition 72
4.12 Comparison of Minkowski metrics in 2D 73

5.1 PMC search card 80
5.2 Cluster of PMC-accelerated machines 81
5.3 Deriving data from music for PMC-enabled searching 86

5.4 Test wells for lithology training 90
5.5 Predicted properties of seismic cube 91

List of Tables

1.1 Relations between papers and contributions 5

2.1 Performance development for DRAM devices 18
2.2 Throughput for different bit counting methods 32

4.1 Technical data for different string search ASICs. 76

5.1 Comparison of siRNA screening capabilities 83

List of Papers

Paper I Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve Jr., and Olaf René Birkeland. A recursive MISD architecture for pattern matching. Published in *IEEE Trans. on VLSI Syst.*, 12(7):727–734, 2004.

Paper II Olaf René Birkeland, Ola Snøve Jr., Arne Halaas, Magnar Nedland, and Pål Sætrom. The petacomp machine — A MIMD cluster for parallel pattern-mining. *2006 IEEE International Conference on Cluster Computing*. In proceedings.

Paper III Ola Snøve Jr., Håkon Humberset, Olaf René Birkeland, and Pål Sætrom. Sequence Explorer: interactive exploration of genomic sequence data, 2005. Manuscript.

Paper IV Ola Snøve Jr., Magnar Nedland, Ståle H. Fjeldstad, Håkon Humberset, Olaf René Birkeland, Thomas Grünfeld, and Pål Sætrom. Designing effective siRNAs with off-target control. Published in *Biochem. Biophys. Res. Commun.*, 325(3):769–773, 2004.

Paper V Olaf René Birkeland, Magnar Nedland, Ola Snøve Jr. Massively parallel MIMD system achieves high performance in a spam filter. Presented at *Parallel Computing* 2005. In proceedings.

Paper VI Pål Sætrom, Olaf René Birkeland, Ola Snøve Jr. Boosting improves stability and accuracy of genetic programming in biological sequence classification Presented at *Genetic Programming Theory and Practice* 2006. Published as book chapter in *Genetic Programming Theory and Practice IV* by T. Soule and R. Riolo and W.P. Worzel (editors), Springer, 2007.

Paper VII Olaf René Birkeland, Ola Snøve Jr. The Pattern Matching Chip Technical whitepaper for Interagon AS, 2002

List of Patents

Patent I Børge Svingen, Arne Halaas and Olaf René Birkeland. A processing circuit and a search processor circuit. International patent PCT/NO99/00344, 1999

This patent is not enclosed in the printed version of this thesis. It can be obtained at http://v3.espacenet.com/origdoc?RPN=WO0029981

Chapter 1

Introduction

THIS doctoral thesis discusses searching through large data volumes with multiple instruction, single data (MISD) processing. In order to test the applicability of this approach, a custom processor was designed and implemented, called the Pattern Matching Chip (PMC). Throughout this thesis I will describe the reasoning behind the PMC's construction, and how this architecture was able to efficiently solve demanding non-numeric applications.

1.1 Aim of Study

The overall aim of this research was to investigate and design methods for searching though large volumes of data. The initial work focused on an index based search system, with construction of new algorithms for efficient processing of the related data structures. As index based systems have limitations in the type of queries that can be handled, the focus was switched to processing unstructured data natively. Although no common definition of *unstructured* data exists, the term is used in this thesis to cover any kind of data where no efficient indexing scheme is known.

The main part of this research has been to identify, design and test a novel MISD architecture for processing unstructured data. The hypothesis of this being an efficient way for processing came from the observed unsuitability of SIMD architectures to parallelize such processing. It is important to note that this work does not claim to replace index based systems in general. This thesis will discuss a subset of search applications where indexing has several disadvantages, and where the proposed architecture is shown to be more appropriate.

In a MISD architecture, multiple parallel searches are executed on a

smaller number of data streams. In the extreme case, there is one shared data stream, originating from a high bandwidth source. The speed of sequential data transfer bandwidth has historically scaled better than other storage technology performance parameters (Patterson 2004). Architectures based on sequential rather than random memory access can thus more readily exploit memory technology improvements.

Further, MISD processing presents a different way of algorithm construction. Evaluating this approach is only possible by formulating new algorithms specifically for the PMC. These algorithms can be benchmarked versus exiting methods solving the same tasks. Our research has demonstrated the applicability of the PMC across multiple application areas.

Throughout all the papers and underlying work, my main contributions on the hardware side have been within designing the processor and system architecture, and identifying feasible implementations. The latter includes everything from ASIC development, to handling the production of the final systems. A lot of effort has been put into making the system as general as possible, enabling multiple application usage cases. On the software side, I have worked on overall algorithm design, and finding effective mappings from the problem domains onto the PMC feature set.

1.2 Thesis structure

This thesis includes seven papers and one patent as the main contributions. The papers will be referenced as Paper I through Paper VII, as defined in the List of Papers on page xiii. Reprints of the original papers are attached at the end of this thesis. The relations between the papers are shown in figure 1.1.

The introduction to this thesis provides the overall structure of the document. Chapter 2 describes work done in bridging the architectural gap between software algorithms, the CPU and system memory in index systems. From this work it became evident that the random access pattern exhibited by many algorithms was not optimal for the most common semiconductor memory technology, dynamic RAM (DRAM). Chapter 3 discusses a somewhat unconventional approach for searching large volumes of data through parallel searches across multiple sequential fragments. This approach brought up a number of data processing applications, for which a traditional CPU would not be a good solution. This led to the creation of the PMC, as described in chapter 4. Chapter 5 describes how this technology was used in different applications. The last chapter summarizes the findings of the research, providing some ideas for further work.

1.3 Research questions

The following is a list of the most important research questions behind this thesis. The propositions to these questions will be summarized in section 6.2.

RQ1 *What are the system requirements for doing unstructured data processing efficiently?*

The main focus for computer performance is on calculation throughput, FLOPS, exemplified by the constant competition for inclusion in the Top500 ranking (http://www.top500.org). Computers are to an increasing extent being used for handling "information", rather than crunching numbers. This might need more attention to other aspects than calculation performance, for example data bandwidth.

RQ2 *What are the constraints for processing unstructured data with standard* CPU *technologies?*

Off-the-shelf computer hardware has very good availability and affordability. The generic computer is although a tradeoff in functionality to enable a fit with as many application categories as possible. Is the *central* processing regime also suitable for unstructured data processing? Could such processing be served better with a more distributed system?

RQ3 *Can other processing architectures than* SIMD *provide better performance for unstructured data processing?*

SIMD (single instruction, multiple data) is currently the most common CPU architecture in personal computers and servers. This is a good fit for many computational intensive applications, where the same operation is applied to more than one data element. For example, all the components of a vector might be scaled with the same factor, regardless of their value. Otherwise, if the flow of operations is dependent on the data values — an important feature for unstructured data processing — , it would be hard to take advantage of the parallelism in a SIMD architecture.

RQ4 *What are the obstacles for massively parallel systems in unstructured data processing?*

Massively parallel systems have scalability limited by Amdahl's law. Furthermore, large numbers of processors are likely to have high power consumption and space requirements, not only for the system

itself, but also the associated cooling systems. What aspects of unstructured data processing can be exploited to overcome — or lessen the impact of — these aspects?

RQ5 *What applications could benefit from unstructured data processing?*

Although researching new technology has a value by itself, improvements in solving real problems are needed for long term sustainability. What existing application classes could benefit from such processing? Does a novel technology approach enable new opportunities?

1.4 Research methods

The research process leading to this thesis has gone through several phases, requiring different methodologies. The initial work for processing unstructured data on regular CPUs consisted mainly of research, development and testing of small prototypes for critical code sections. These served as tools for better understanding of the bottlenecks in such systems.

This first phase suggested that the SIMD architecture was not a good fit for the applications under investigation. Thus the next phase included a study of existing literature on alternative technologies. A MISD architecture was in theory a good fit, but lagged behind the other alternatives in terms of existing research and deployment. Furthermore, we wanted a degree of parallelism not previously tested in existing studies. Thus most of the proposed ideas for a MISD architecture could only be simulated on existing hardware.

To identify if this architecture and parallelism imposed limitations overseen by theoretical estimations, the third phase was to identify, design and construct such an MISD processing device, the PMC, and the surrounding hardware and software systems. This phase is further described in chapter 4.

Armed with this technology, the fourth and final phase consisted of extensive benchmarking of the novel architecture in several applications; both synthetic scenarios — and in my opinion more importantly — real world applications.

1.5 Contributions

The main contributions of this thesis are:

Table 1.1: Relations between papers (see section 1.6) and contributions (see section 1.5).

	Contributions					
Paper	C1	C2	C3	C4	C5	C6
Paper I	•	•	•	•		
Paper II		•	•	•		•
Paper III		•				•
Paper IV		•				•
Paper V		•	•	•		•
Paper VI						•
Paper VII	•					
Patent I	•				•	

C1 Development of the Pattern matching Chip (PMC), an MISD architecture parallel processor implemented as an ASIC

C2 A scalable system architecture, with proven operation of 500,000 parallel processing elements at near 100 % utilization.

C3 A parallel processing architecture with low infrastructure requirements.

C4 A power efficient design, with power consumption at par with or better than alternative technologies.

C5 Patenting of the PMC architecture

C6 Research, development and benchmarking of selected applications for the PMC architecture.

The merits of the research are presented in the enclosed papers, while chapter 2 through 6 rather provides the context of the overall work. Given the diversity of topics covered in the papers, all details will not be reiterated in these chapters. Table 1.1 summarizes the relations between the papers and the respective contributions.

1.6 Paper Abstracts

Paper I and Paper IV have appeared in peer-reviewed scientific journals. Paper V and Paper VI have been presented at scientific conferences. Pa-

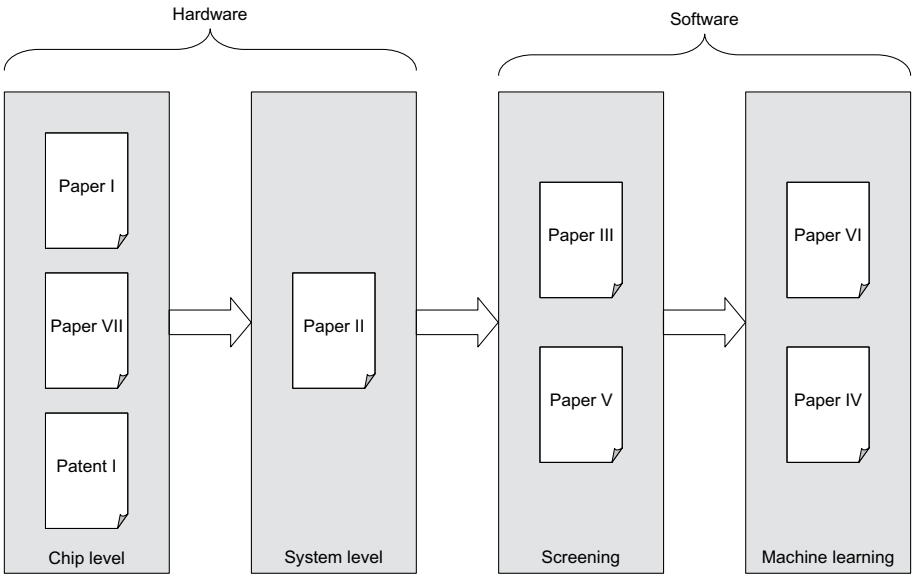

Figure 1.1: The papers in this thesis present both hardware and software aspects of our work. Paper I, Paper VII and Patent I describe research and design of the chip level technology. The cluster level system is covered by Paper II. Paper III and Paper V present the usage of this cluster when screening for occurrences of predefined patterns. Finally, Paper IV and Paper VI explain how the same cluster can be used for accelerating genetic programming.

per VI will also be published as a book chapter with Rick Riolo as editor. Paper II has been accepted for conference presentation in September 2006. Paper III has been submitted for publication, but not yet accepted. Paper VII is a technical note. More detailed information on each paper can be found in the List of Papers on page xiii.

The patent has been filed in the European EP system and 13 additional countries. The patent has so far been granted by 6 of these 14 patent bodies, with 8 countries still pending. Due to the volume of the patent (107 pages), it is not reprinted in this thesis. A version can be found for downloading at the address given in the List of Patents at page xv. The patent is a formalization of the material in Paper I and Paper VII, thus reading the entire patent is not required for an understanding of the work behind this thesis.

The following list of abstracts is taken from the papers and patent related to the research in this thesis.

Paper I *A recursive* MISD *architecture for pattern matching.* Many applications require searching for multiple patterns in large data streams for which there is no preprocessed index to rely on for efficient lookups. A multiple instruction stream-single data stream (MISD) VLSI architecture that is based on a recursive divide and conquer approach to pattern matching is proposed. This architecture allows searching for multiple patterns simultaneously. The patterns can be constructed much like regular expressions, and add features such as requiring sub-patterns to match in a specific order with some fuzzy distance between them, and the ability to allow errors according to prescribed thresholds, or ranges of such. The current implementation permits up to 127 simultaneous patterns at a clock frequency of 100 MHz, and does 1.024×10^{11} character comparisons per second.

Paper II *The petacomp machine — A* MIMD *cluster for parallel pattern-mining.* Multiple instruction stream-single data stream (MISD) architectures have not found many practical applications in supercomputing. We present a multiple instruction stream-multiple data stream (MIMD) cluster implementation that uses MISD search processors with extreme pattern mining performance as a building block.

We use PCI cards that hold sixteen search processors with local memory to build a relatively small cluster of five PC nodes with six PCI cards each. This cluster can handle anything between 64 independent queries at 48 GB per second, to 30,720 independent queries at 100 MB per second.

The cluster's performance characteristics are such that we can easily scale the system to 10^{15} operations per second with containable overhead using just 100 nodes. Further, the solution has lower power consumption and memory bandwidth requirements than comparable technologies.

Paper III *Sequence Explorer: interactive exploration of genomic sequence data.* Current solutions for complex motif searching in DNA and protein sequences are not interactive as users usually wait tens of seconds before the results can be viewed. We propose a hardware-accelerated client-server solution that is fast enough to retain the interactive feeling even when screening whole genomes. We structured our framework for interactive sequence analysis around query, dataset, filter, and result presentation modules. The query and dataset specification enable simultaneous, interactive screening of multiple complex queries against several datasets. The filters impose restrictions such as only allowing hits to be reported if they occur in coding regions, and the different result presentations include histograms and hit lists. Our results show that interactive searching is possible even though response times vary significantly depending on filter, network bandwidth and hit frequencies. With a relatively small server, we obtain response times of about one and a half second on gigabytes of data when queries are sufficiently complex to avoid network bottlenecks due to high hit frequencies.

Paper IV *Designing effective si*RNA*s with off-target control.* Successful gene silencing by RNA interference requires a potent and specific depletion of the target mRNA. Target candidates must be chosen so that their corresponding short interfering RNAs are likely to be effective against that target and unlikely to accidentally silence other transcripts due to sequence similarity. We show that both effective and unique targets exist in mouse, fruit fly, and worm, and present a new design tool that enables users to make the trade-off between efficacy and uniqueness. The tool lists all targets with partial sequence similarity to the primary target to highlight candidates for negative controls.

Paper V *Massively parallel MIMD system achieves high performance in a spam filter.* Supercomputer manufacturing is usually a race for floating point operations, and must therefore opt for a design that allows for the highest possible clock frequency. Many modern applications are, however, limited not only by the number of operations that can be performed on the data, but by the available memory bandwidth. We

review the main features of a MISD architecture that we have introduced earlier, and show how a system based on these chips is able to scale with respect to query and data volume in an email spam filtering application. We compare the results of a minimal solution using our technology with the performance of two popular implementations of deterministic and nondeterministic automatons, and show how both fail to scale well with neither query nor data volumes.

Paper VI *Boosting improves stability and accuracy of genetic programming in biological sequence classification.* Biological sequence analysis presents interesting challenges for machine learning. Using one of the most important current problems — the recognition of functional target sites for microRNA molecules — as an example, we show how joining multiple genetic programming classifiers improves accuracy and stability tremendously. When moving from single classifiers to bagging and boosting with cross-validation and parameter optimization, you require more computing power. We use a special-purpose search processor for fitness evaluation, which renders boosted genetic programming practical for our purposes.

Paper VII *The Pattern Matching Chip.* The Pattern Matching Chip (PMC) is an Application Specific Integrated Circuit (ASIC), capable of searching for advanced patterns in arbitrary data at a constant high speed. The PMC is based on breakthroughs made by researchers at the Norwegian University of Science and Technology (NTNU), who have devoted more than 15 years into developing ASICs for approximate searching. With a clock frequency of 100 MHz, the PMC is able to search with up to 64 distinct queries on 100 MB of data per second.

Patent I *A processing circuit and a search processor circuit.* A processing circuit P_1 for recognition and comparison of complex patterns in high speed data streams can form a node in a network of circuits of this kind and comprises an interface for inputting of parameters for the circuit, at least one kernel processor P_0 in the form of a comparator unit (COM) for comparing two data words, a logic unit (E) connected with the comparator unit and comprising a multiplexer (MUX1), a first D flip flop (2), a latency unit (LAT) for delaying a positive binary value with a given number of time units, a second D flip flop (4), a sequence control unit (SC) which monitors and controls a comparison operation in the comparator unit (COM), and a result selector (RS) which combines two result values from other processing circuits or other result selectors.

A search processor circuit (PMC) for performing search and comparison operations on complex patterns comprises a multiprocessor unit P_n with processing circuits P_1 in a tree structure and forms a binary or superbinary tree with $n+1$ levels S and degree $k = 2m$, m being a positive integer ≥ 1. An underlying level S_{n-q} generally comprises 2^{mq} circuits P_{n-q} provided nested in the $2^{m(q-1)}$ circuits P_{n-q+1} on the level S_{n-q+1}. A 0th level S_0 defined for $q = n$ in the unit P_n comprises $2^{m(n-1)}$) to 2^{mn} kernel processors P_0 which form comparator units (COM) in the circuits P_1. All circuits $P_1, P_2 \ldots P_n$ have identical interfaces (I) and a logic unit (E) with a result selector (RS) for collecting the results of a search operation or a comparison operation. Use in search engines for search and retrieval of data stored in data bases.

1.7 On joint authorship

All papers and patents building the base for this dissertation origin from collaborations with several other people, bringing together a multidisciplinary team, with each member contributing with unique competence in our respective fields.

The papers and patents are listed on page xiii and xv respectively. My specific contributions beyond co-writing all of the manuscripts are as follows:

Paper I Contributed to the architecture research for the PMC, identified suitable solutions for implementation, headed the research project.

Paper II Researched the cluster architecture, evaluated the resulting performance, identified potential system enhancements.

Paper III Defined requirement specifications, researched implementation optimizations, and wrote the user tutorial.

Paper IV Contributed to the research on methodology, especially on ensuring efficient usage of the PMC.

Paper V Identified workload distribution scheme, performed performance comparisons.

Paper VI Contributed to methods.

Paper VII Selected topics to be presented. Researched selection of processing element functions to provide both any Boolean function as well as higher order functionality.

Patent I Contributed to the architecture research for the PMC

1.8 Supplementary material

I have written a tutorial on usage of the screening application described in Paper III. This tutorial is available upon request.

Chapter 2

Hardware aware optimizations in data search applications

Grove giveth and Gates taketh away

Robert Metcalfe (1946 –)

THERE is a large gap between the increments in raw compute speed and the resulting application level benefits. While the semiconductor industry has been able to keep on track with Gordon Moore's predictions, often referred to as "Moore's law", providing a doubling in the number of transistors on an integrated circuit every 18 months, all of this added compute power has not been harnessed.

This chapter will describe some of the most important issues that prevent applications from getting full advantage of hardware improvements. The discussion will focus on two main aspects. First, the widening gap in performance between processing and memory (Wilkes 2001); secondly — and perhaps even more important — the resulting effects when this memory gap and other feats of a modern processor are not top priorities within the general programmer community. For most applications, advanced compilers can help bridging the technology gap (Tian et al. 2005). For data intensive algorithms, the programmer can use hardware awareness to gain large improvements (Eichenberger et al. 2005). These aspects will be illustrated by examples taken from the implementation of an index based web search engine, described in more detail in Risvik (2004).

The examples presented in this chapter are taken from work done in the construction of commercial program code in 2000, and are not previously published. The proposed solutions are more an illustration of achievable improvements through CPU-aware programming, than claiming to be the

state-of-the-art algorithms in their respective fields. The main purpose of this chapter is to describe the difficulties in getting decent performance from a SIMD architecture applied to algorithms with a data dependent instruction flow. These findings provided motivation and inspiration for developing an application targeted processor architecture, which will be described in the following chapters.

2.1 Background

Modern approaches to programming agree that code optimization should wait until a profiler reports a performance bottleneck in the application. The 20-80 rule applies to all stages of software development, meaning that large portions of a programmer's time and a program's runtime are spent on relatively small parts of the total volume of code (see for instance Beck 2001, for a discussion of these topics). Programmers should therefore write comprehensible code that implements simple designs rather than opting for complex solutions that often turn out to have only a marginal effect on the application's performance.

Add to the aforementioned arguments that today's compilers often outsmart the average programmer when it comes to local optimizations, and it may seem like it is all a matter of selecting the best algorithm with respect to theoretical complexity and memory requirements. This is not the case. Critical parts of the code, as identified by performance profiling, may benefit significantly from architecture-dependent code optimizations.

2.1.1 The evolution of the x86 processor

One of the most widespread processor architectures is based on the Intel 8086 processor from 1978, also referred to as the x86 architecture. This architecture will be used as an example throughout this chapter. Although several other architectures exist, modern x86 implementations incorporate much of the same concepts found in other processors.

During its years of existence, the x86 CPU architecture has changed considerably. The original 8086, as shown in figure 2.1, was executing one instruction at a time[1]. All but the very simplest instructions required multiple clock cycles, being executed as a small program within the CPU itself

[1]The 8086 did also have a separate bus interface unit that speculatively fetched the next instructions. The main purpose of this unit was to remove the memory latency, that is the function of a cache, rather than parallel execution

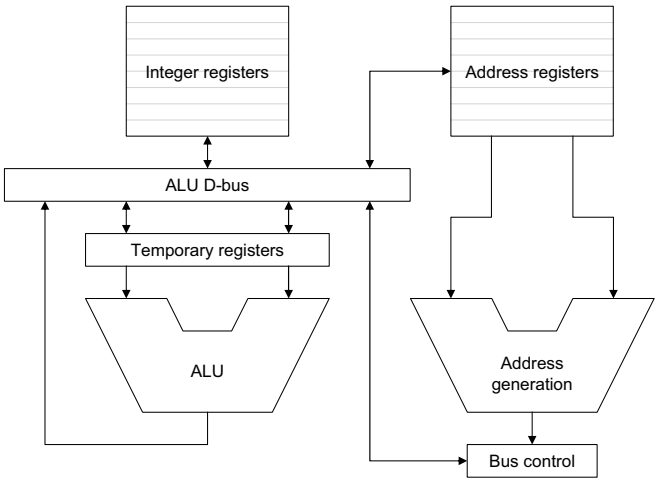

Figure 2.1: Block diagram of the Intel 8086 processor. All operations are executed by a single ALU.

as microcode. A typical computer had a very limited amount of memory, typically in the tens of kilobytes.

This posed an understandable model for the programmer when designing algorithms. The determining factor for performance was the number of operations required, and how this depended on the dominating size n of the problem at hand. The performance was analyzed using the O-notation for characterizing algorithms, where a linear algorithm would be described as $O(n)$, and a cubic algorithm as $O(n^3)$. The focus among programmers where thus to get to the fastest algorithm based on the O-notation. (More information on O-notation can be found in Aho et al. 1982, page 16–27).

Throughout the generations following the original 8086, deeper pipelining and higher frequencies were used. Released in 2005, the Prescott design was a very different architecture, as shown in figure 2.2 (Boggs et al. 2004). The instructions are decoded into RISC-style micro-operations, which are fed to the execution engine. Instead of a single ALU, there were seven execution units, of which two could issue two operations per clock cycle. Thus up to nine instructions can be started in a single clock cycle[2]. The processor has a deep pipeline with 31 stages. There could potentially be more than a hundred operations in some state of execution at any moment. The parallelism implied high sensitivity to serial dependencies in the in-

[2]Only three operations can complete simultaneously

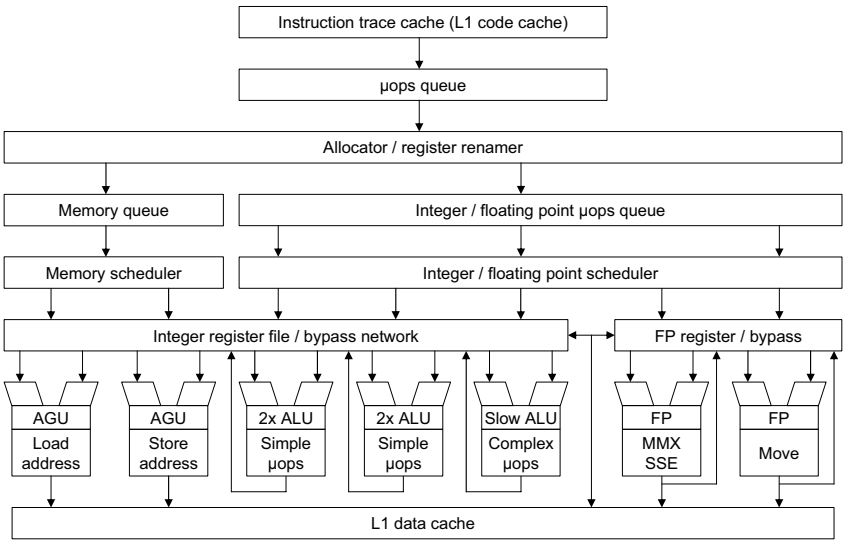

Figure 2.2: Block diagram of the Intel Pentium4 (Prescott) processor. Execution is run in parallel across two floating point and five integer ALUs (Boggs et al. 2004, Adapted from).

struction stream, as warned by Amdahl's law (Amdahl 1967). A previous study suggests that this level of instruction level parallelism is not readily achievable (Wall 1991).

From the initial 8086 and up to the Prescott design, each generation gave dramatic increases in clock frequency, but the instruction processing latency saw small reductions. New architectures even increased this latency at their time of introduction. This latency can to a large extent be handled by compilers in combination with an out-of-order execution engine. There are although limits to what can be achieved with these methodologies (Gerber et al. 2006). If the algorithms are constructed with a 25 year old architecture in mind, reaching high utilization of the available processing power is unlikely. Consequently, handbooks guiding programmers to suitable practices are needed (Intel Corporation 2004).

Figure 2.3 shows an abrupt change between the Prescott and the more recent Core design in terms of clock frequency. The Prescott architecture was intended to scale to 10 GHz, but never got faster than 3.8 GHz. The main obstacle preventing increased clock frequency was not the speed of the transistors, but rather the power consumption. When reducing geometries below 90 nm, the transistor leakage became larger than expected (Krishnarnurthy et al. 2002; Mistry et al. 2007).

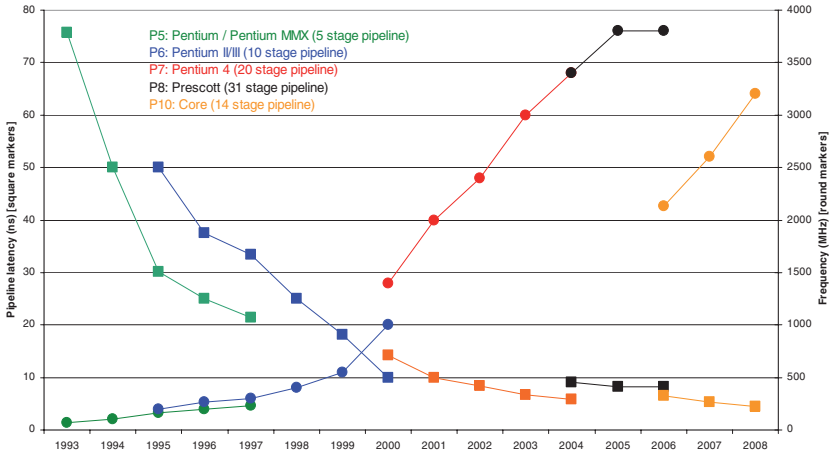

Figure 2.3: Development of clock frequency and instruction processing latency of the x86-family of CPUs over time. Clock frequency is plotted with round line markers against the right hand axis. Latency is plotted with square line markers against the left hand axis.

The Core design reduced the clock frequency, and at the same time almost halved the number of pipeline stages. As a result, the execution latency did not increase despite the reduced frequency. Even though the clock frequency of the Core architecture has increased gradually with the maturity of the design, the road maps from Intel; the largest x86 manufacturer; does not suggest major future cycle time reductions.

Further advances in processing throughput are anticipated to stem from using multiple processor cores, either identical copies, or heterogeneous designs (Kumar et al. 2003). Parallel cores can also be implemented by hyper-threading, which replicates the part of a processor holding the state, but not the execution units (Marr et al. 2002). The replicated states and their respective threads run alternatingly, increasing the time between instruction issue within the same thread. Since each thread runs slower, memory access latency is reduced when measured in execution cycles. If the threads can run independently, the overall processing throughput is larger than for a higher frequency, single threaded execution.

Future processor designs, not only in the x86-family, it thus likely to increase the number of parallel execution cores, but not provide significant speed-up for single threaded applications. Any algorithm with a built in serial dependency must be redesigned to increase its performance.

Table 2.1: Performance for DRAM devices over several generations. Random access time is given as the time from issuing a row select command from idle mode, until data appears. Per-pin bandwidth is the available bandwidth per data pin on the component. Data adapted from Hennessy and Patterson (2007, figure 5.14).

Year	Density	Random access	Per-pin bandwidth
1980	64 Kbit	150 ns	13 Mbit/s
1986	1 Mbit	100 ns	40 Mbit/s
1992	16 Mbit	60 ns	66 Mbit/s
2000	256 Mbit	45 ns	133 Mbit/s
2004	1 Gbit	35 ns	533 Mbit/s
2006	2 Gbit	40 ns	1600 Mbit/s

2.1.2 The evolution of memory technology

Apart from the changes in processing architecture, there have also been large changes in the memory subsystem. The most commonly used memory technology is still dynamic RAM (DRAM) due to its low price and high storage capacity. As shown in table 2.1, the storage volume of DRAM devices has increased dramatically, with a factor of 30000 over 25 years. The random access time has had less than a four-fold reduction, and has seen no improvement for the latest generation. Notably, the per-pin transfer bandwidth has seen better improvements (Patterson 2004). Combined with wider memory interfaces, this substantial increase in bandwidth, at a level above the reduction of random access time, can be exploited in algorithm development, as will be shown in section 2.2 and 2.3. Burger et al. (1996) have predicted that off-chip access at one point will become so expensive that all memory will reside on the CPU itself. So far, the integration of processing and memory is not widespread, but some implementations exist (Gebis et al. 2004; Kirsch 2003; Patterson et al. 1997).

In addition to hyper-threading mentioned in the previous section, caching is used at one or more hierarchical levels to hide the increasing difference in processing speed and memory random access time (Hennessy and Patterson 2007, chapter 5). This do indeed provide a large performance benefit in the average case, but one can no longer consider all memory accesses to be equal. We can rank different memory accesses in order of increasing execution time.

Cache access Data residing in one of the cache levels is accessible with low latency. For the uppermost cache level, this is usually within one clock cycle (e.g. 0.3 ns).

Sequential access Due to the high bandwidth of memory, sequential access has a low cycle time even from DRAM (less than 1 ns). Cache fills and flushes are done through cache lines; reading address n in memory will in most cases cause the entire cache line to be retrieved. If n is th last element of a cache line, most processors will do speculative prefetching of location $n + 1$, giving sequential reads a further advantage over sequential writes.

Random access Random access to DRAM is relatively slow. Table 2.1 gives the access time for the DRAM itself as 40 ns even for the most recent devices. The CPU to memory bridge adds an additional delay of the same order of magnitude.

Paged memory Paged memory is residing on a disk drive, and will in the context of a CPU take forever to access (e.g. 10 ms). For the remaining discussions, all memory structures will be assumed to fit within the system memory, and not require paging. The principles explained, will still increase the performance if paging should occur, as paging also performs better with sequential than random accesses.

As a guideline, an algorithm should try to work with a subset of the data that can fit within the cache (memory locality). As bringing data in and out of the cache is costly, the data should preferably be processed to its final results with a single access. With the expectation of large memory subsystems, algorithms frequently use large data structures residing in memory. When accessing such structures, maintaining a predictable, preferably also sequential, address sequence is beneficial. Random accesses will have low cache hit ratios, and a large performance impact.

In the following sections some of these principles will be applied to real examples, demonstrating the performance potential that can be harnessed when keeping the processor architecture in mind during algorithm development.

2.2 Sorting large arrays

Sorting is a common subproblem within many applications (Knuth 1997, chapter 5). For example, it can be used to maintain ordering between elements within a data structure for more efficient retrieval, or for ranking ele-

ments before providing this as output to a user or other applications. Sorting has been investigated for decades, but still no golden solution solves all kinds of sorting problems. Quicksort is considered to work well for most common application and data volumes (Knuth 1997, section 5.5), but is not suited for parallel machines (Knuth 1997, section 5.2.2). Other important sorting algorithms are insertion sort, bin sort and radix sort.

2.2.1 Characteristics of search application at hand

The sorting application that will be discussed, originated from ranking web pages based on a user query. With a myriad of web pages, combined with the lack of specificity in typical web user queries, most queries result in a list of millions of potential web pages. Risvik (2004) describes an architecture where each web page in the listing has a relevance score. In order to present the user with a ranked list of results, not only for the ten most relevant web pages, but also anywhere within the results, the list must be sorted.

A sorting algorithm had already been implemented based on recursive bin sort. This algorithm was taking much of the CPU processing time (30 % in typical scenarios), which was considered to be unacceptably high. The algorithm was theoretically very efficient, with $O(n \log n)$ complexity. Further analysis with CPU profiling tools[3] confirmed the suspicion that the task was not computationally demanding. But more alarming, the analysis showed that most of the time was spent waiting for memory references. There were obvious margins for improvement if the memory access pattern could be optimized.

2.2.2 Rearranging memory references

The main limitations causing the memory bottleneck were an unpredictable access pattern, and multiple intermediate moves of each data record before reaching its final position. The unpredictability of the memory write operations lies within the problem itself: With unsorted data as input, the final location is data dependent. The clue to a solution would lie in minimizing the number of such moves to a minimum.

This reduction was achieved by investigating the data more elaborately before moving any records. This must be very efficient to not become a bottleneck by itself. Reading sequentially through the data to calculate their

[3] Intel VTune

```
procedure radixsort;
{ Sorts list A of n records with keys consisting of fields f_1, ..., f_k
  of types t_1, ..., t_k respectively. The procedure uses arrays
  B_i of type array[t_i] of listtype for 1 ≤ i ≤ k, where listtype
  is a linked list of records. }
begin
   for i := k downto 1 do begin
      for each value v of type t_i do {clear bins}
         make B_i[v] empty;
      for each record r on list A do
         move r from A onto the end of bin B_i[v],
         where v is the value of field f_i of the key of r;
      for each value v of type t_i, from lowest to highest do
         concatenate B_i[v] onto the end of A
   end
end {radixsort}.
```

Figure 2.4: Radix sort algorithm (as defined in Aho et al. 1982, page 281).

value distribution was considered. Such an approach could be integrated with a variant of radix sort.

2.2.3 In-place radix sort

Radix sort (Hildebrandt and Isbitz 1959) is a sorting algorithm that divides the sorting problem into sorting on a single bit of the sorting key at a time, which also can be extended to work on multi-bit symbols (see Knuth 1997, section 5.2.3 for details). A variant of this principle is being used even outside computer algorithms, such as the postal service. Here, each shipment is initially sorted into bins by the most significant digit in the zip code, thereafter each bin is resorted with regards to the next digit(s). A formal description of the radix sort algorithm is given in figure 2.4. Note that this radix sort starts by sorting on the least significant digit first, for later concatenation of the individual digit bins and resorting on the next digit of increasing significance.

Due to the splitting in figure 2.4 of the sorting key into fields f_i, the resulting complexity for fixed size integer keys (where k is constant) is given

by Aho et al. (1982, page 281) as

$$O(n + \sum_{i=1}^{k} s_i) \tag{2.1}$$

where s_i is the number of different values of type t_i. However, since s_i has a constant upper bound for fixed size integers, the complexity is reduced to $O(n)$.

Variable sized keys would also affect the complexity of e.g. Quicksort. The analysis of Quicksort resulting in $O(n \log n)$, assumes that each comparison is $O(1)$. In practice, all digital computers require $O(m)$ for a comparison, with m being the length of the variables.

While radix sort as described in figure 2.4 is $O(n)$ for integer keys, it has some serious disadvantages for sorting extremely large arrays. Most obvious is the doubled memory usage for storing the sorting bins, and secondly the handling of these when implemented as lists. More subtle, but even more important, is the fact that radix sort starts out with the least significant digit. Thus the records brought together in the same bin are not more likely to end up in each others vicinity in the final sorted order. This implies no gain in locality when processing the data, prohibiting the cache system from hiding the memory latency of the DRAM system memory.

To improve the locality, advice was taken from the postal service sorting method mentioned above: Start sorting on the most significant digit instead. This would result in much better data locality after the initial sorting step.

There was also a need to remove the duplicated memory footprint. It was observed that if one could put a record directly into the correct position range for each sorting iteration, there would be only one swap element at any time requiring extra storage, i.e. in-place sorting.

In-place MSD radix sort implementations have been compared by Al-Darwish (2005), who argues that infrequent usage of such algorithms is due to the misconception of complex implementations and tedious bookkeeping. This was also stated by McIlroy et al. (1993) as *"The troubles with radix sort are in implementation, not in conception"*. Variants of radix sort have been used for sorting strings (Bentley and Sedgewick 1997) or using adaptive digit size (Maus 2002). Adaptive sorting algorithms seek to avoid worst case behavior due to input data values (Estivill-Castro and Wood 1992)

Given the application at hand, several optimizations could be applied. The sorting keys were fixed size integers, simplifying management of the in-place swapping of records. Similarly, by avoiding program flow de-

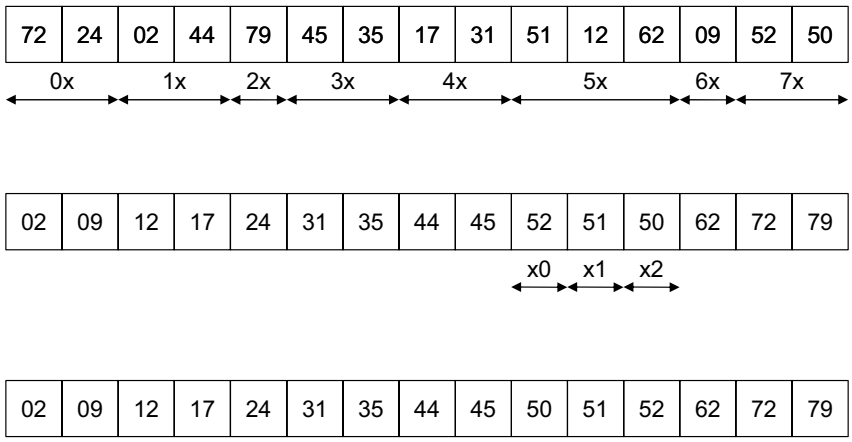

Figure 2.5: Example of in-place radix sort using octal numbers.

pendent on the value of the sorting keys, there was no need to make an adaptive algorithm.

These ideas were implemented into a variant of most significant digit (MSD) in-place radix sort, with figure 2.5 showing the steps in a simple case. For simplicity, the example uses two-digit octal numbers. The topmost line consists of $n = 15$ such unsorted numbers. Starting with the most significant digit, the number of occurrences of each symbol for that digit is found by reading linearly through all the data. This finds two numbers starting with 0, another two starting with 1, and so on. After finding the value distribution, pointers can be placed to the sections of the array that should be used for each value of this digit.

The next step will swap all records into the right range with respect to this digit. The algorithm starts by finding the lowest labeled range containing any elements, as some ranges might be empty. The first record in this range, in this case 72, becomes the swap element. This number is put into the first position of the correct range, taking the place of 52 which now becomes the new swap element. The pointer for the first available position in the range 7x is updated accordingly. This process continues (through 51 and 12) until we reach 02 which is taking the vacant position previously held by 72. Once such a swap cycle completes, the first element that has not yet been moved is found, starting another swap cycle. This process is terminated after n swaps. Since no element will be swapped more than once due to the design of the algorithm, all elements must now have been swapped, and consequently be placed in the correct range.

The process is repeated recursively within each range on the next digit,

as shown in the middle line of figure 2.5. In this case, only the range *5x* needs to be reordered, ending with a sorted array as shown in line 3. A listing of this algorithm implemented in the C programming language is shown in figure 2.6, using radix 256 on 32-bit integers.

Some of the implementation details in figure 2.6 justify an explanation. The function takes the data array as an argument, in addition to the size *n* and the number of right-hand shifts (*shift*) that is needed to bring the current sorting digit into the least significant byte of the key. Invoking the function is thus done with *shift* set to 24.

The switch statement on lines 7–12 counting occurrences could have been replaced by the following statement:

```
for(i=0; i<n; i++) cnt[(a[i]._rankValue>>shift) & 0xFF]++;
```

Even though this is more general, expanding the capability of the function beyond 32-bit integers, it infers a non-constant shift as well as a redundant AND-operation for the most significant bit. Constant shifts can be aggressively optimized by the compiler, speeding up the execution. The swapping is also started at the highest labeled value range, as the search for value ranges that need processing is more efficient with a decrement of the loop variable *i*.

Insertion sort is used when the size of each value range becomes relatively small. The threshold for shifting to insertion sort is architecture dependent. Insertion sort proved useful after reaching a level when the code and all data in the current range could be contained in the uppermost cache levels of the CPU. Furthermore, the arrays *last*, *ptr* and *cnt* have some redundancy. Two tables would be sufficient, at the expense of code readability.

2.2.4 Algorithm complexity

In the analysis of the run time of the algorithm proposed in figure 2.6, n is the number of records to sort, each having a key of m binary digits (bits). In each iteration, a radix of 2^d is used for the partial sorts, with d being the number of bits in that radix. c_i will be used for denoting constant factors. The initial counting of occurrences has a run time of

$$t_1(n) = c_1 \cdot n \tag{2.2}$$

Building the pointer tables has a run time of

$$t_2(n) = c_2 \cdot 2^d \tag{2.3}$$

```
1  void inplace_radixsort(ArrayRep a[], unsigned int n, unsigned int shift) {
2    unsigned int last[256], ptr[256], cnt[256];
3    unsigned int i, j, k, sorted, remain;
4    ArrayRep temp, swap;
5
6    memset(cnt, 0, 256*sizeof(unsigned int)); // Zero counters
7    switch (shift) { // Count occurrences
8      case 0:  for(i=0; i<n; i++) cnt[a[i]._rankValue&0xFF]++; break;
9      case 8:  for(i=0; i<n; i++) cnt[(a[i]._rankValue>>8)&0xFF]++; break;
10     case 16: for(i=0; i<n; i++) cnt[(a[i]._rankValue>>16)&0xFF]++; break;
11     case 24: for(i=0; i<n; i++) cnt[a[i]._rankValue>>24]++; break;
12   }
13   sorted = (cnt[0]==n);  // Accumulate counters into pointers
14   ptr[0] = n-cnt[0];
15   last[0] = n;
16   for(i=1; i<256; i++) {
17     ptr[i] = (last[i]=ptr[i-1]) - cnt[i];
18     sorted |= (cnt[i]==n);
19   }
20   if (!sorted) { // Go through all swaps
21     i = 255;
22     remain = n;
23     while(remain>0) {
24       while(ptr[i]==last[i]) i--; // Find uncompleted value range
25       j = ptr[i]; // Grab first element in cycle
26       swap = a[j];
27       k = (swap._rankValue >> shift) & 0xFF;
28       if (i!=k) { // Swap into correct range until cycle completed
29         do {
30           temp = a[ptr[k]];
31           a[ptr[k]++] = swap;
32           k = ((swap=temp)._rankValue >> shift) & 0xFF;
33           remain--;
34         } while (i!=k)
35         a[j] = swap; // Place last element in cycle
36       }
37       ptr[k]++;
38       remain--;
39     }
40   }
41   if (shift>0) { // Sort on next digit
42     shift -= 8;
43     for(i=0; i<256; i++)
44       if (cnt[i]>INSERT_SORT_LEVEL) inplace_radixsort(&a[last[i]-cnt[i]], cnt[i], shift);
45       else if (cnt[i]>1) insertion_sort(&a[last[i]-cnt[i]], cnt[i]);
46   }
47 }
```

Figure 2.6: Modified radix sort for in-place sorting of 32-bit integer keys using an initial occurrence count.

The swapping stage will move one of the n elements in each iteration, yielding

$$t_3(n) = c_3 \cdot n \tag{2.4}$$

In the recursive sorting on the next digit, insertion sort is only used for arrays below a constant size, and can thus in the context of this analysis be considered to run in constant time. Thus the recursion step takes

$$t_4(n) = c_4 \sum_{i=0}^{2^d} T(\frac{n}{r_i}) \tag{2.5}$$

where r_i is the number of elements in range i for each of the 2^d symbols in the partial sort key, and $T(n)$ is the runtime of the overall algorithm. Furthermore, $\sum_{i=0}^{2^d} r_i = n$. The run time is thus recursively defined as

$$T(n) = (c_1 + c_3) \cdot n + c_2 \cdot 2^d + c_4 \sum_{i=0}^{2^d} T(\frac{n}{r_i}) \tag{2.6}$$

It is obvious that a high number d of bits to be used in the radix will degrade the performance. In the implementation shown, $d = 8$. The run time is then given as

$$T_{d=8}(n) = c_5 + c_6 \cdot n + c_4 \sum_{i=0}^{256} T_{d=8}(\frac{n}{r_i}) \tag{2.7}$$

with

$$\sum_{i=0}^{256} r_i = n \tag{2.8}$$

Thus $T_{d=8}(n)$ has the complexity $O(n)$.

2.2.5 Measuring the sorting throughput

The new algorithm was benchmarked on different CPUs, spanning from 450 MHz Pentium III to 1.4 GHz Pentium4, resulting in three times faster execution compared to the original recursive binsort. As expected, the memory accesses were still the bottleneck, but were now being used more efficiently. For example, in test runs with 100 million records with random keys, each record was in average only moved 1.7 times (out of the theoretical maximum of 4) before reaching a location where the remaining sorting would be done by insertion sort, occurring in only in the uppermost cache level.

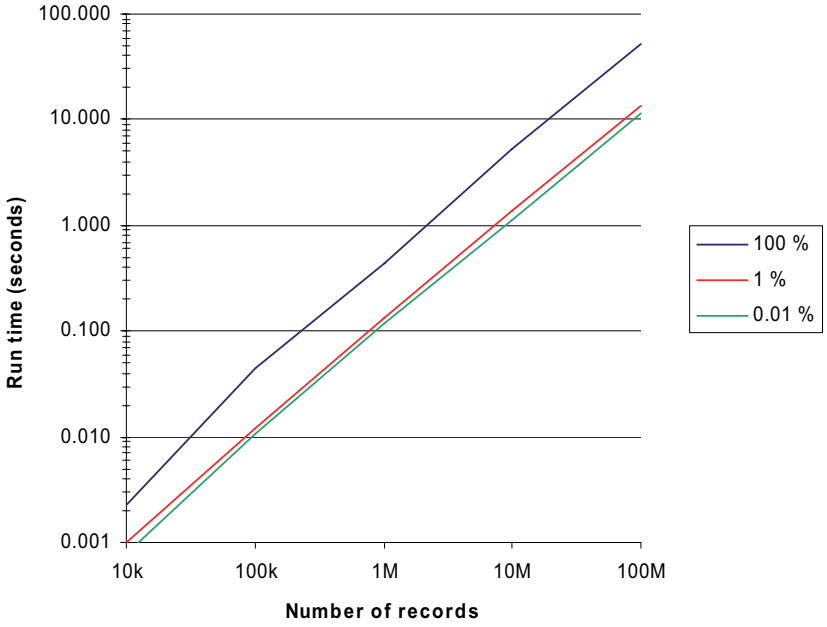

Figure 2.7: Scalability of in-place radix sort for complete (100%) and partial (1% and 0.01%) sorts. The fractional sorts can establish correct sorting for any range of keys, or any region within the fully sorted list. Note that both axes are plotted with logarithmic scale. Each record consists of 8 byte, including the 32-bit key. These tests were run on a 733 MHz Pentium III.

As shown in figure 2.7, the run time is indeed linear with respect to the number of records, in correspondence with the complexity analysis in section 2.2.4. This figure also shows the run time when sorting only a specified fraction of the array as described in section 2.2.6.

2.2.6 Remarks on practical aspects

Although the above algorithm sorts the entire array, this was not needed in all application cases. For example, a user of a web search engine might only request results ranked from position 1000 to 1500. A variant of the algorithm that takes this into account was developed. After sorting on the most significant digit, only the relevant subsets of ranges were sorted on the lesser digits.

For the even more common web search, where the user only wants the top n records, a separate version of insertion sort holding only the n highest ranked records in sorted order was developed. This provided a further seven fold performance increase, since the memory accesses in practice were reduced to a linear read of all records. Using this algorithm, processing speed was able to run at the peak bandwidth of the memory subsystem.

A final note should be made on swapping records in memory as opposed to just pointers to the same records. As pointers usually would be smaller than the entire records, this might be considered advantageous. In reality, the records needed to be more than 20 times the size of a pointer for this to hold true on the test machines mentioned in section 2.2.5. Storing pointers requires additional memory, stealing both storage space and bandwidth. Additionally, accessing a small pointer still requires a full cache line access between the cache and system memory. Sorting through pointers should thus only be used in applications with large records.

2.3 Counting the number of 1-bits

Another problematic area within the web search engine was the seemingly simple task of counting bits, that is finding the number of bits set to 1 in a long vector. There are multiple algorithms for solving this problem, where the optimal solution might not be the most intuitive one.

2.3.1 The naive approach

The simplest way for counting the bits would be to use two nested loops. Since modern CPUs do not access bits individually, but rather longer words, working directly on the bit level becomes impractical. The outer loop would iterate through all the words containing the bit vector, while the inner loop would accumulate the number of bits within each such word.

For each bit, three operations would be needed: Shifting the bit to the least significant position, masking off any higher order bits, and accumulating the final result. This tedious algorithm takes in the range of 200 instructions just for processing one 64-bit word. Furthermore, a direct implementation would have large serial dependencies between the instructions, prohibiting efficient use of all the available parallelism in the CPU, shown in figure 2.2

2.3.2 Using a lookup table

The typical way to get around the problems described in the previous section, is to replace the inner loop with looking up precomputed values in a table. For a word size of $n \cdot m$ bits, the number of bits would be found through n lookups in a table of 2^m potential values. While m should be large to reduce n, and thus the number of operations, the size of the lookup table grows exponentially with m. Selecting m in an implementation is thus a balancing act, most commonly ending up with $m = 8$ as a good compromise.

An often overseen implication of this approach is the memory accesses inferred by the table lookups. Even though this table is relatively small, each read of a word from the bit vector, is followed by n read operations from the lookup table. Solving performance issues by using more memory accesses is not attractive in a long term perspective, given the widening gap between compute speed and memory bandwidth.

Furthermore, the functional units of the CPU are not really kept busy. Each addition would never add a value larger than m to the accumulator. With a 64-bit processor and $m = 8$, only the least significant byte of the registers are effectively used, corresponding to 12.5%.

To increase the performance, two separate issues must be considered: The full register width of the CPU should be used for processing. At the same time, this should be achieved without accessing temporary data structures like the lookup table.

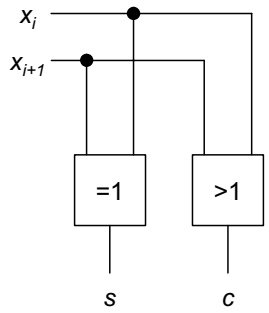

Figure 2.8: Adding bit x_i and x_{i+1} resulting in sum s and carry c.

2.3.3 Using the full register width

In order to find a better implementation for this problem, the basic operations in binary additions had to be investigated. This disclosed a processing flow where every bit in the involved registers has the possibility of doing some actual work on every operation.

Basics of binary addition

The addition of two single digits is shown in figure 2.8, using a half-adder. The sum s is the XOR of the inputs, while the carry c is the AND of the inputs. The concatenation of c and s are also a two-bit binary number containing the sum, that is $\sum_{j=i}^{i+1} x_j$.

By chaining half-adders, the sum of more than two bits can be produced. Without a potential carry bit at every stage, the circuitry can be simpler than if chaining complete full-adders, as shown in figure 2.9, for calculation of $\sum_{j=i}^{i+3} x_j$

Parallelizing bit serial addition

At first glance, this approach might look like a step in the wrong direction. In order to exploit CPU parallelism, bit serial adding has been introduced. An important observation is although that all operations on the data are now two input, single output binary operations. Such AND/XOR functions can be run effectively in registers, where the result of each bit position will not affect the other bits in the same register. Thus parallel operations can be run in one register. For a n-bit register, n summations are carried out in parallel. The register bank is so to speak used orthogonally, with n se-

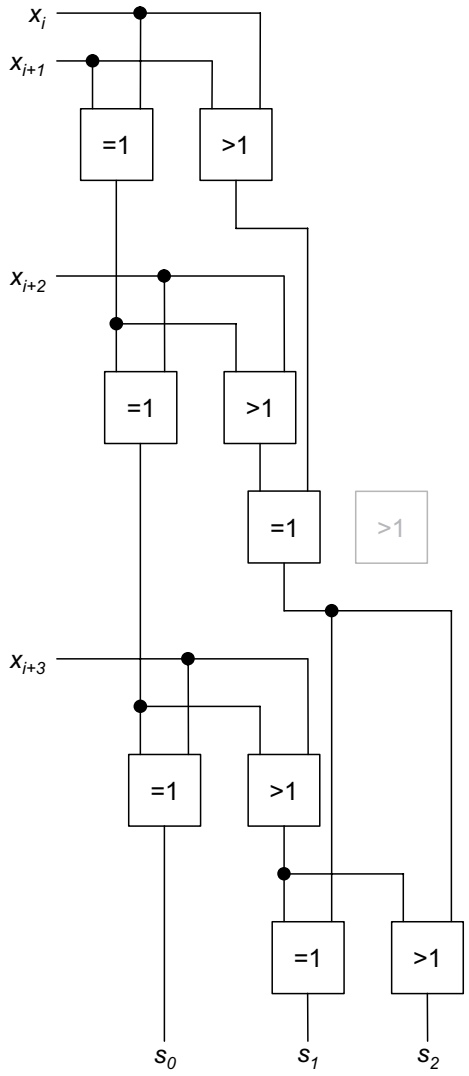

Figure 2.9: Calculating $x_i + \ldots + x_{i+3}$ into the sum $s_2 s_1 s_0$. Unused part of half-adder is shown in gray.

Table 2.2: Throughput for different bit counting methods. These numbers were measured on a 733 MHz PentiumIII CPU. All implementations but the MMX version were limited by CPU capacity, while the latter was limited by memory bandwidth.

Method	Bits per iteration	Clocks per iteration	Bits processed per clock
Mask/shift/add	64	200	0.3
Lookup-table	32	6	5
Orthogonal registers (C)	480	40	12
Orthogonal registers (MMX)	1920	100	20

rial adders in parallel. For summing $m \cdot n$ bits, $\lfloor \log_2(m) + 1 \rfloor$ registers are needed for storing the n counters. The limiting factors for parallelization are thus the width of each register, limiting the number n of parallel counters; and the depth of the register bank, limiting the width m of each bit serial counter. This is quite intuitive, as the intention was to keep all the register bits busy.

2.3.4 Implementation

This method was initially implemented in the C language, with $m = 15$ and n determined by the longest available integer value in that environment (shown in figure 2.10). Details for handling bit vectors of a size not dividable with 15 integers are omitted for readability.

Lines 4–24 implement the half adders as depicted in figure 2.9. Line 26–41 accumulates the 32 counters stored across four registers into an ordinary binary number. The first step in these conversions (line 27, 31, 35 and 39) deviates from the others. The obvious alternative to line 30 would be

```
tmpa = (s0 & 0x55555555) + ((s0>>1) & 0x55555555);
```

This line takes multiple pairs of one-bit counters stored as $x_{j+1}x_j$ and calculates $x_{j+1} + x_j$ stored as two-bit values. Even though x_{j+1} and x_j are individual single bit counters, they are stored as a two-bit binary number interpreted as $2x_{j+1} + x_j$. By using this interpretation, and subtracting x_{j+1} we get

$$2x_{j+1} + x_j - x_{j+1} = x_{j+1} + x_j \tag{2.9}$$

```
1   sum = 0;
2   for(i=0; i<bitVectorSize; i+=15)
3     {
4       s0 = bitVector[i];
5       tmpa = bitVector[i+1];
6       s1 = s0 & tmpa;
7       s0 = s0 ^ tmpa;
8       tmpa = bitVector[i+2];
9       s1 = s1 ^ (s0 & tmpa);
10      s0 = s0 ^ tmpa;
11      tmpa = bitVector[i+3];
12      tmpb = s0 & tmpa;
13      s0 = s0 ^ tmpa;
14      s2 = s1 & tmpb;
15      s1 = s1 ^ tmpb;
                ⋮
16
17      tmpa = bitVector[i+14];
18      tmpb = s0 & tmpa;
19      s0 = s0 ^ tmpa;
20      tmpa = s1 & tmpb;
21      s1 = s1 ^ tmpb;
22      tmpb = s2 & tmpa;
23      s2 = s2 ^ tmpa;
24      s3 = s3 ^ tmpb;
25
26      // Join counters
27      tmpa = s0 - ((s0>>1) & 0x55555555);
28      tmpb = (tmpa & 0x33333333) + ((tmpa>>2) & 0x33333333);
29      sum += (((tmpb + (tmpb>>4)) & 0x0f0f0f0f) * 0x01010101) >> 24;
30
31      tmpa = s1 - ((s1>>1) & 0x55555555);
32      tmpb = (tmpa & 0x33333333) + ((tmpa>>2) & 0x33333333);
33      sum += (((tmpb + (tmpb>>4)) & 0x0f0f0f0f) * 0x01010101) >> 23;
34
35      tmpa = s2 - ((s2>>1) & 0x55555555);
36      tmpb = (tmpa & 0x33333333) + ((tmpa>>2) & 0x33333333);
37      sum += (((tmpb + (tmpb>>4)) & 0x0f0f0f0f) * 0x01010101) >> 22;
38
39      tmpa = s3 - ((s3>>1) & 0x55555555);
40      tmpb = (tmpa & 0x33333333) + ((tmpa>>2) & 0x33333333);
41      sum += (((tmpb + (tmpb>>4)) & 0x0f0f0f0f) * 0x01010101) >> 21;
42    }
```

Figure 2.10: Parallel bit-serial bit counting executed orthogonally across multiple registers implemented in C. Note that the counting of the 4th to the 14th bit is omitted from this listing.

as desired. Thus the line

$$\text{tmpa} = \text{s0} - ((\text{s0}>>1) \;\&\; 0\text{x}55555555);$$

is equivalent, and saves one AND masking operation.

An x86 specific version was written in assembly, using SIMD registers and operations. Due to the register width in MMX registers[4], n could be increased to 128. With 8 MMX registers available, the summation was done for $m = 15$ bits in each iteration with 128 parallel counters. This required 4 registers for the accumulator itself, leaving 4 registers for temporary variables. The summation loop consisted of reading data linearly from memory, followed by pure register-to-register operations. One such digest accumulated the number of 1's across $n \cdot m = 1920$ bits.

After this summation, the accumulator is stored in 4 registers, and need to be combined and stored as a standard integer before reusing the registers for the next section of the bit vector. Fortunately, special MMX instructions are available for such purposes. This functionality is not available when coding in C, making the assembly version even more efficient.

One assembly loop, including converting the accumulators, consisted of approximately 200 instructions, most of which could be issued in pairs. The run time was four times faster than the lookup table method described in section 2.3.2. The results for the different methods are summarized in table 2.2. The implementation in C performed 2.5 times faster than the lookup table alternative. In practice, the limiting factor for the assembly version was the read speed from memory. As this was a linear read, it ran at peak bandwidth. Additionally, since the access pattern was highly predictable, prefetch instructions were included into the program, hinting the cache subsystem well in advance to bring in new data.

Previous implementations of bit counting in the web search engine had special algorithms for handling bit vectors with very scarce 1's (Risvik 2004). These could take advantage of checking whether all bits in a word were zero, in which case no calculation needed to be done. With the new algorithm, the processing itself was no longer a limiting factor, and such customization was not needed. This resulted in a predictable run time of the bit counting, independent of the input data.

[4] 256-bit MMX2 registers are available, but operations on these take twice as long than MMX operations, and are thus no benefit in processing speed.

2.4 Comments

This chapter has discussed efficient ways to handle searches in large data volumes with an index based system. Such systems rely on organizing the raw data into a data structure, usually an inverted index file, which by itself becomes voluminous. These data structures must be used optimally in order to achieve high performance. Due to its construction, any series of random keyword lookups in an inverted index, will lead to a series of random reads per query; although a very limited quantity. Performance bottlenecks in accessing such data structures have also been observed in other search engines (Barroso et al. 2003; Brin and Page 1998). Note that the discussion in this chapter is limited to finding and sorting documents satisfying a set of criteria, not the underlying processing step of calculating the relevancy of each document by algorithms like PageRank (Brin and Page 1998) or Hypertext Induced Topic Selection (HITS) (Kleinberg 1999).

As shown by the examples, algorithms can be designed to use the maximum capacity of the available resources, although with more hardware dependent design, and loss of comprehensibility of the code. Hardware aware software optimizations have been successfully implemented in several bioinformatics applications (see for example Farrar 2007; Rognes 2001; Rognes and Seeberg 2000). In most cases, the speed limiting factor turns out to be the available memory bandwidth.

Indices have obvious advantages for searching by keywords in relatively static collections of documents, for example a web search engine. There are although some limitations to this approach, especially with regards to the required time taken to build an index, and the limited query flexibility. In the next chapter a different approach will be discussed, entirely omitting the need for any data structures built on top of the raw data. Chapter 4 follows with the details of one such implementation, before chapter 5 discuss applications where this approach might be more attractive than indexing.

Chapter 3

Sequential data processing — in parallel

The great tragedy of science — the slaying of a beautiful hypothesis by an ugly fact

Thomas Henry Huxley (1825–1895)

UP to now, this thesis has discussed searching large volumes of data by the means of an indexing system. This serves as an efficient implementation for keyword based information retrieval. However, in some application cases the use of an index is less efficient, or even impossible.

This chapter will argue that complex pattern matching could be executed by scanning sequentially through the data, rather than accessing them through a data structure. Although sequential, such processing can be run in parallel on a large number of processors, resulting in a short run time despite the linear dependency on the data volume.

3.1 When indices fall short

All indexing systems are based on extracting keyword tokens from the raw data, placing these into a data structure called an index. Later searches in the same data are done by looking up one such keyword in the index to find pointers to the occurrences in the original data. The efficacy of such lookups is dependent on the data structure chosen for storing the index.

Building the index is usually a lengthy process. This is of less importance in relatively static document collections. The return on the computational investment spent on building an index should be considered

when the underlying data changes so rapidly that frequent re-indexing is required.

Indexing relies on all of the following conditions to hold true:

- Later retrieval will be done with a predetermined set of keywords.
- Keyword tokens can be placed in a data structure suitable for the application at hand. Some of the important differentiating characteristics are the number and size of the tokens, multiple occurrences of each token, and the specificity of each token.
- There is available time to run the indexing step ahead of the data processing itself.

Applications conflicting with any of these requirements, will not lend itself easily to an index based solution. Several examples of such applications will be discussed in chapter 5. Most of the work related to this thesis has been targeting applications within bioinformatics. One such application, siRNA design described in Paper IV, is breaking *all* of the above requirements.

To illustrate the shortcomings of an indexing system, this application can be simplistically described as follows: A 19-nucleotide (19-mer) RNA sequence should be designed with respect to efficacy as a gene silencing agent, as well as having a large edit distance (Sankoff and Kruskal 1983) from other known sequences (Snøve Jr. et al. 2004). Since gene silencing can occur even with multiple sequence differences (Amarzguioui et al. 2003; Dykxhoorn et al. 2003; Hannon 2002; McManus and Sharp 2002; Zamore 2001), it is not sufficient to look for identical sequences. Between the 19 nucleotides, at least four differences can occur while the molecule still maintains some of its silencing potency. Thus the longest consecutive conserved fragment between two similar sequences could be as short as three nucleotides[1].

The uniqueness of the sequence, quantified by the edit distance, would need a check for any 3-mer subsequence in the reference data as a seed point. With four nucleotides, only 4^3 unique subsequences exist. Any index lookup would thus match $\sim$ 2% of the reference data. Checking all seventeen 3-mers within an 19-mer, with a typical data base size of 50 MB, would result in 17 groups of around one million fragments that need to be patched together in an optimal way. Not only is the size of the problem only marginally reduced in this initial step. The combinatorial explosion

[1]Some studies indicate that even this is too optimistic. Tolerance for more than four mismatches could reduce the longest conserved fragment size to two nucleotides. See Paper IV for details and references

when combining the potential seeds is unbearable, rendering indexing as an unsuitable approach.

For the efficacy calculations, one might expect that longer tokens could be used. In reality, the biological mechanism is not yet fully understood, and any restrictions imposed by such tokens could affect the results. When using genetic programming (Koza et al. 1999) to find the most effective molecules, it is beneficial to have all degrees of freedom in hypothesis generation (Sætrom 2005). Genetic programming will by definition build motifs of random characters and operations. Using an index for finding motif occurrences could limit what patterns that could be constructed.

3.2 Pattern matching without indices

Navarro (2001a) has reviewed different non-indexing algorithms for approximate string matching, demonstrating that no algorithm is superior for all types of data and degrees of approximation. The fastest algorithms discard negatives through filtering before checking for matches in the remaining data.

When indexing fails, another alternative approach would be to process the raw data directly. As discussed in chapter 2, the most optimal memory access method in terms of transfer bandwidth is a linear read. This approach is taken by several algorithms, among them dynamic programming and finite state machines (automata). Belkin and Croft (1992) argues that such data filtering is a variant of information retrieval. Filtering is easier to run in parallel (Mak et al. 1991) than data structure manipulations like sorting (Bitton et al. 1984).

3.2.1 Dynamic programming

Dynamic programming is a bottom-up approach for solving problems (see for example Aho et al. 1982). The opposite would be a top-down approach, dividing each problem recursively into smaller sub-problems. A top-down approach has the risk of solving the same sub-problem multiple times, resulting in an exponential complexity. If there is only a polynomial number of subproblems, these could be solved first, and later combined into higher level solutions. This is the approach taken by dynamic programming. It starts by storing the sub-problem results, typically in a multidimensional matrix, reusing these to compute the remaining matrix values. Memoization is a recursive approach storing the results of subproblems when encountered, in addition to maintaining a *memorandum* of the subproblems

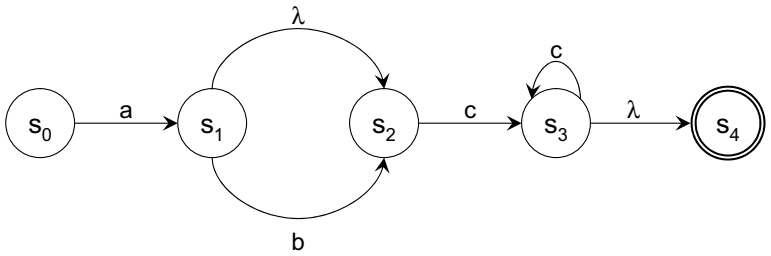

Figure 3.1: NFA state machine accepting the pattern *ab?c+*.

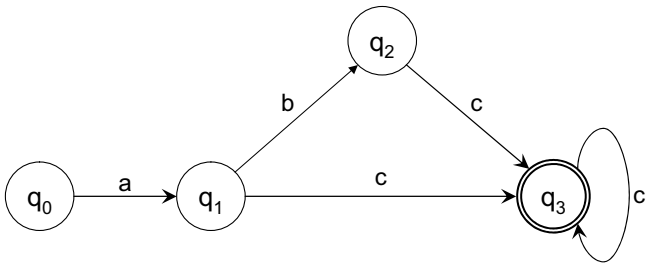

Figure 3.2: DFA state machine accepting the pattern *ab?c+*.

that have been solved (Michie 1968). Later referrals to the same subproblem do thus not need recomputation.

The Smith-Waterman dynamic programming algorithm (Smith and Waterman 1981) is frequently used in bioinformatics for sequence alignment. It starts by calculating the alignment scores for the first nucleotide in the query versus the entire reference sequence. The remaining query nucleotides are added one by one, recalculating the accumulated alignment scores in each step.

3.2.2 Finite state machines

Finite state machines (FSM), also referred to as automata, are models composed of states, transitions and actions. Finite state machines are divided into two classes: Deterministic (DFA) and nondeterministic finite automata (NFA). Both types can be used for accepting patterns. The patterns are encoded into the FSM, which consumes the input data as a string. A match is found if the state machine ends up in an accepting state.

Figure 3.1 and 3.2 show two finite state machines that accept the same input string. Accepting states are marked with double rings. Transitions

are marked with the accepted symbol. Transitions marked with λ in the NFA version can be taken without consuming an input symbol. In the NFA-version, more than one traversal of the state machine might be valid at any moment, leading to the classification as *nondeterministic*. In contrast, a DFA can only be in one state. A NFA can be converted to a DFA, however with a potential exponential increase in the number of states and transitions. More information on finite state machines can be found for example in Sipser (2005).

3.2.3 Limitations with dynamic programming and automata

Dynamic programming and finite state machines are valuable solutions to a large range of applications, but have limitations preventing their usability in certain scenarios. NFA-implementations need to track a large number of simultaneous active states, making the processing speed unpredictable. They might also need more memory than available for storing the intermediate possibilities.

Dynamic programming and DFA implementations do in contrast have predictable memory requirements as well as bandwidth. The tradeoff is in the vast usage of memory structures; either for storing intermediate results in dynamic programming, or for capturing the state transition tables needed with a large number of states and transitions.

As discussed in chapter 2, using large data structures for speeding up problems does not scale well with the current directions of SIMD CPU design, which favors computational throughput rather than memory bandwidth. Additionally, the common CPU architectures do not lend themselves to processing multiple queries in the same data stream.

Automata implemented in software have a rather large overhead even for parsing well defined patterns, in the order of 10 – 100 of clocks per symbol consumed (van Lunteren et al. 2004). When parsing even more unspecific patterns, this penalty will become larger.

A SIMD architecture can process multiple data streams, but only with the same instructions. Parallel pattern matching requires a program flow dependent on the input data in each of the data streams. While SIMD provides large benefits in applications with a predetermined instruction flow across different data — for example in image filtering — it falls short when used for evaluating multiple patterns across a single data stream.

3.3 Sequential data processing

The performance of running parallel queries during a linear search through raw data is dependent on the bandwidth of the data stream, as well as the throughput of the processing stage. Within the scope of this dissertation, the focus has been on implementing the processing stage, not on improvements of the data bandwidth already available in commercial memory products. With a given, fixed data bandwidth, the problem reduces to creating an architecture that can keep up with that speed.

To evaluate multiple queries versus a single data stream, a MISD architecture emerged as an alternative. With separate instruction streams for processing individual queries, a flexible processing unit can be built. This approach will be discussed in the remaining chapters of this dissertation. After introducing the PMC in chapter 4, chapter 5 will discuss several applications where linear search is applicable. Several of these practical problems could not have been efficiently solved by any of the methods described in section 3.2.

3.4 Improving speed by parallel execution

Sequential processing has a run time of $T(n) = c \cdot n$, resulting in $O(n)$ complexity. This is theoretically inferior to a large number of sub-linear algorithms. In practice, it might still be attractive given the constant c can be made sufficiently small. Given abundant processing capabilities, c is only determined by the data bandwidth. Having such a dependency on memory bandwidth might immediately seem counterintuitive after the discussion in chapter 2.

The key point is that such sequential scans can be easily parallelized, at least for the types of applications that will be discussed in chapter 5. This is due to the fact that the patterns of interest have a limited scope. For example, searching for a gene promoter motif is only relevant for matches within a small window of the total data volume.

The reference data can thus be split — potentially with some overlap dependent on the scope size — and distributed for processing across multiple nodes without loosing any matches. With reference data several orders of magnitude larger than the relevant patterns, a large number of nodes can be applied. The aggregated bandwidth of hundreds, or even thousands of such modest data channels could be sufficient to make sequential processing attractive.

3.5 Sequential access is aligned with technology trends

With the performance of sequential access being bound by the data bandwidth, its usability should preferably not be limited exclusively to today's dominating memory technology, DRAM. Fortunately, linear access is the most efficient access method in all major current data storage technology. Storage technologies organized in matrixes (for example DRAM, SRAM and NOR-flash) have roughly half the access time for accesses within the currently active memory page.

Other technologies are serial by nature (for example NAND-flash and disk drives), with huge penalties resulting from non-linear access. Even beyond the storage segment, serial data access is a common practice, such as network traffic. An efficient implementation for processing of linear streams of data could thus be used with numerous data sources.

Chapter 4

The Pattern Matching Chip

If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?

Seymour Cray (1925–1996)

THE previous chapter introduced the concept of processing large volumes of data through linear searches. This chapter will focus on the concept and design of the Pattern Matching Chip (PMC), implementing this idea. For readers who are unfamiliar with the PMC, it is recommended to read Paper VII before starting on this chapter.

The research behind designing the PMC has been published in Paper I and Patent I. This chapter complements that information by adding background material on the architecture and development process of the chip. The implementation has taken a considerable amount of the allocated time, and deserves some consideration. Further, the ability of the PMC to compare multidimensional numeric data — although with a slightly unconventional distance metric — is presented. This chapter will focus on the technical aspects of the PMC itself, leaving more detailed application descriptions for the following chapter.

4.1 Introduction

As described in the examples in chapter 2, the largest gains when building data search algorithms for superscalar CPUs are achieved by optimizing the memory access patterns, as well as taking advantage of the SIMD instruction set of modern CPUs. SIMD vector processing reduces the required instruction bandwidth (Kozyrakis and Patterson 2003), but require that all

data are treated equally in lockstep operation. This increased my awareness of that a general CPU is not optimized for processing unstructured data. Hillis and Guy L. Steele (1986) present a fine grained processing architecture, but with a data parallel approach.

A fundamental architectural mismatch exists in applications where multiple users are processing independent queries versus the same collection of data. Natively, such searches require multiple paths of query processing on the same data, which should call out for a MISD architecture. Algorithms for fuzzy pattern matching exists (see for example Isenman and Shasha 1990; Navarro 2001b), but most of them have severe speed penalties with increasing degree of non-perfect matches (Navarro and Raffinot 2002).

MISD architecture was first classified by Flynn (1972), but is still very rare in real implementations besides systolic arrays (Chalmers and Tidmus 1996). MISD was considered impractical by Hwang and Briggs (1985). Lima et al. (2001) does although describe a general purpose MISD architecture with better scalability than traditional multiprocessor systems.

In a scenario with myriads of independent queries, massive parallelism might be feasible without being bounded in performance gains by Amdahl's law (Amdahl 1967). This gave the inspiration to construct a MISD architecture tailored for data searches, and test out the practical implications in real applications.

As suggested from the findings described in section 2.3, it is easier to distribute workload with maximum throughput with a large number of small tasks, than the other way around. We thus chose to make an architecture building on the "atoms" of a query, that is the individual symbols in the query string.

The technical details of the PMC are described in Paper I and Paper VII. This chapter will thus not repeat the implementation details, but rather focus on the reasoning and considerations behind the chosen implementation.

4.2 Existing technologies

Several architectures for approximate patterns and string matching have been proposed over the years (Blüthgen and Noll 2000; Cheng and Fu 1987; Dahle et al. 1997; Foster and Kung 1980; Hirschberg et al. 1998; Park and George 1999; Sastry et al. 1995; Yu et al. 1984), with some based on content addressable memory (Hirata et al. 1988; Robinson 1992). A fair comparison of performance is difficult due to significant discrepancies be-

tween the algorithms (Paper I). Other proposed architectures exist for finding the longest common subsequence (Lin and Yeh 2002; Michailidis and Margaritis 2005; Mukherjee 1989), while the connection in our research to bioinformatics have favored approximate searches. Several groups have developed special purpose hardware for sequence analysis in computational biology (see for instance Hirschberg et al. (1998); Lindelien (2002); Schmidt et al. (2001)), and Hughey (1996) have published a comparison of parallel hardware for sequence comparison and alignment.

Our architecture implementation is based on previous work by Halaas (1983), and does almost 1,000 times more character comparisons per chip than the closest competitor (Paper I), and the outlined cluster nodes in Paper II each contain 480 search processors with a corresponding linear increase in performance. The queries are typically specified as regular expressions (Friedl 2002; Nedland et al. 2002b). Regular expression matching in strings have been reviewed by Navarro (2001a) and Michailidis and Margaritis (2002).

Commercial alternatives for regular expression matching in general are available from for example Integrated Device Technologies (Santa Clara, CA), Safenet (Belcamp, MD), TippingPoint (Austin, TX), and Tarari (San Diego, CA). However, benchmarking is difficult as their designs have never been published in peer-reviewed journals. Most hardware regular expression accelerators convert the problem into a DFA state machine, for example as shown by van Lunteren et al. (2004). This provides high throughput, with the drawback of not being able to map all relevant regular expressions as we have shown in Paper V.

While other architectures have been proposed for solving an increasing discrepancy between CPU and memory bandwidth in SIMD/MIMD processing (Patterson et al. 1997), our MISD construction has reduced that problem on the conceptual level. With this processing scheme, data is fed only once — and in sequential order — to the processing stage, reducing the memory subsystem requirements.

Among other groups working on MISD processing, Lima et al. (2001) have proposed an architecture for general processing called SHIFT[1]. They demonstrate better scalability with increasing chip transistor count due to the reduced global wiring. This architecture has demonstrated general applicability, for example in graphics processing (Nakamura et al. 2003), and better scalability than ordinary multiprocessing (Lima and Nakamura 2002). Schneider and Rossignac (1995) have also proposed a MISD archi-

[1]Michael J. Flynn is one of the coauthors for this paper, helping to fill out the almost empty MISD-quadrant of his taxonomy

tecture for graphics processing.

4.3 Design goals

The overall design goal for the PMC was to build a system with predictable performance at low cost. This requires the architecture to be linearly scalable to maintain a constant ratio between problem sizes and the computational resources required. All operations on the data should be $O(n)$, including data preprocessing. Relieved from building complex data structures, all processing could be done in real time, entirely eliminating preprocessing in most cases.

4.3.1 Linearly scalable architecture

The resources required by a PMC based system should be linearly dependent on the volume n of data to be analyzed, and the total query volume. Query volume is given as the accumulated lengths of all queries q_i. Informally, the capacity p of a system can be stated as

$$p = n \cdot \sum q_i \tag{4.1}$$

Obviously, p as measured in comparisons between query symbols and data, will easily become a very high number. A vast number of processing elements must be able to operate autonomously to maintain scalability. To keep cost down, high chip scale integration is required. This also reduces the power consumption as seen in other multiprocessor designs (Kahle et al. 2005; Kongetira et al. 2005)

The system should be scalable even in the situation of dynamic data (as demonstrated in Paper V) or dynamic queries (Snøve Jr. et al. 2005). Scenarios where both data and queries are constantly changing were not considered, as the proposed architecture would not have the required bandwidth to receive all of this information from the host system.

4.3.2 No preprocessing before analyzing data

Without an index to speed up access to the desired section of the data at hand, *all* data analysis should be completed with one linear scan. The runtime $T(n) = c \cdot n$ of this approach is bounded by $O(n)$, which is not appealing for large volumes of data. However, through massive parallelization, c could be reduced sufficiently to still make this a viable approach. The

PMC should serve as an pattern matching accelerator within a standard PC. Such a PC could serve as a building block for larger clusters, with near linear performance gains when adding more machines.

Different types of preprocessing would have undesirable properties. For example, indexing can be a lengthy process (Risvik 2004), and even loose accuracy (Snøve Jr. and Holen 2004), while mapping the problem into different domains could have nondeterministic resource requirements (Paper V).

4.4 Previous architectures

The PMC is the fourth proposed architecture with origins in the Department of Computer and Information Science at the Norwegian University of Science and Technology. The previous implementations have provided both a concept validation for MISD processing, and generated ideas for more advanced architectures. All designs have shared the concept of using multiple processing elements to monitor a sequential stream of data. The organization of the processing elements; feeding of data; aggregation of results and the processing of these have been implemented very differently.

The processing concept is founded on orthogonal range queries, that is finding all points within a n-dimensional box. A query box Q can formally be described as

$$Q = [a_1, b_1] \times \ldots \times [a_n, b_n] \qquad a_i, b_i \in \mathbb{R} \tag{4.2}$$

For a set of points $X \subseteq \mathbb{R}^n$, an orthogonal range query will find the subset of points within the query box, that is $X \bigcap Q$. Orthogonal range queries have been extensively studied, with several proposed software solutions. Agarwal and Erickson (1996) and Willard (1996) provide good summaries of the subject literature. Charikar et al. (2002) has a more recent data structure proposal. Algorithms can achieve sublinear search time, at the expense of superlinear data structure size or build time; or both.

4.4.1 The F and H-matrices

The F and H matrices were conceptual studies for evaluating orthogonal range queries in hardware. The F-matrix also had extensions for handling non-orthogonal general range queries. The H matrix had a 128 stage data pipeline, where the range membership for the current data value was evaluated. The output was a Boolean function, indicating if all data values

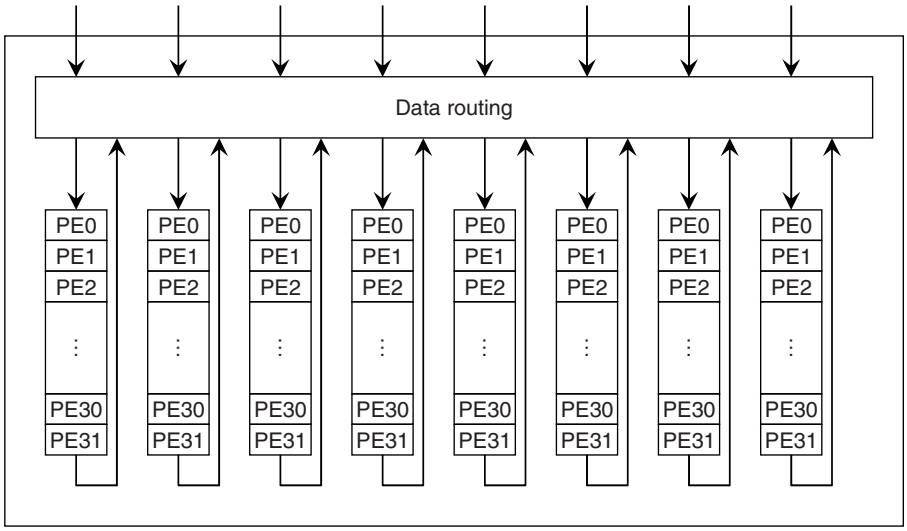

Figure 4.1: Block diagram of the MS160 data pipeline. The chip has 256 processing elements (PEs), arranged into eight groups of 32 PEs

were within their respective range, that is

$$\prod_{i=1}^{128} x_i \in [a_i, b_i] \tag{4.3}$$

where a_i and b_i are the lower and upper bounds respectively for each of the stages, being compared to the current data value x_i of that stage. The H-chip was incorporated into a demonstration system running on a PC, providing rapid keyword searches in text collections such as Shakespeares complete works, the Bible and the Norwegian phone directory. Halaas (1983) describes the principles of these matrices, while Heggebø (1989) use the H-matrix as an usage example for a VLSI compaction tool.

4.4.2 The MS160

The MS160 development was started in 1991, with the first prototype available in late 1993. The chip has eight *windows*, each consisting of 32 processing elements (PEs) as shown in figure 4.1. The chip had eight byte-wide data stream inputs, that could either be routed to their respective windows, or replicated and sent to two, four or all eight windows. The data routing could also take the data stream output from one window

and use that as the input to the next window, creating a longer window for larger search patters. All data streams ran at 20 MB/s, for a total of 160 MB/s across all eight input streams. Any query required at least one window, allowing a maximum of eight simultaneous queries at 20 MB/s throughput, or one query duplicated across all windows for 160 MB/s throughput.

Each PE consisted of two comparators, checking if the current data value was within a range specified by upper and lower limits. A window would report a hit if all PEs had a match. Unused PEs would be configured with upper and lower values set to accept any value. Any other function than AND would require more than one window. Results from the eight windows where used as a look-up table key, determining which combinations that qualified as a hit.

Although the MS160 had a high theoretical throughput, there were a number of limitations that led to performance and scalability issues in practical use. The most important of these were:

Complex queries required multiple windows. Any other pattern than a sequence of acceptable data value ranges required more than one window in the MS160. Although the design allowed for variations in the value of each element in the sequence, this proved to be an infrequently used feature besides handling upper and lower case letters. Rather than variations in the value range of each element, complex queries demanded more flexibility in the positioning of individual values within the sequence. Without this capability, the MS160 was limited to keyword searches, and soon surpassed by index based systems as these are especially good at prefix and keyword queries.

Low PE utilization As a result if the previous issue, even a simple query such as *a or b followed by c* would require three windows if *a* and *b* where not encoded as consecutive values. Due to limitations in the data distribution, also a the fourth window would be unavailable for other queries in this example. Thus half the chip, or 128 PEs was consumed by a three symbol query.

Low bandwidth utilization The design could either run at the maximum data bandwidth (160 MB/s), or the maximum number of simultaneous queries (8), but not both at the same time if all queries should be evaluated for all the input data. This approach served single user applications well, where query latency could be reduced under light system load. With multiple or complex queries, half or less of the available memory bandwidth was utilized.

Interrupt driven result processing When finding a hit, MS160 reported this to the host system through an interrupt. The host system needed to respond to this interrupt before further hits could be processed. Slow interrupt handling in most operating systems implied a limit to the rate of hits that could be reported. Unspecific queries returning numerous hits would easily swamp the whole system.

Glue logic requirement The design was more a processing core than a system component. It had a generic FIFO style interface for the data streams, but did not include any memory controller for the data storage. Correspondingly, the system interface had an address and data interface similar to a SRAM requiring additional logic; at that time an ISA interface. The glue logic both increased the cost and circuit board area for building a working solution. More importantly, the lack of integration implied that both the memory and system interface could not use the connected devices at their full potential

Unbalanced processing resources compared to storage volume The MS160 system design had one chip supported by up to 320 MB of RAM, which at the time took 160 memory chips to implement. The above issues with low effective memory bandwidth as well as low PE utilization made most of the RAM operating in idle mode. It was obviously undesirable to have low utilization of the largest and most expensive part of the subsystem, at that time the memory.

In summary, the MS160 was good at keyword searches, but did not support sufficiently fuzzy queries to enable applications where index based systems are inadequate. It was intended as a single chip accelerator, and was difficult to use in larger configurations with parallelism outside the chip itself. MS160 served as the query engine for the web service FTPSearch during 1992 – 1996, when it became outperformed by an software application. In 1996, FTPSearch served 650000 queries per day, and was one of the most visited web sites in Norway.

4.5 Implementation choices for the PMC

This section will discuss the implementation options considered for the PMC, and the choices made for the final design. Figure 4.2 shows the overall block diagram. The search core is the largest part of the design, but still requires a number of support functions to achieve the design goals. The design choices for these supporting functions was considered just as important as for the core itself. Figure 4.4 has a more detailed view of these

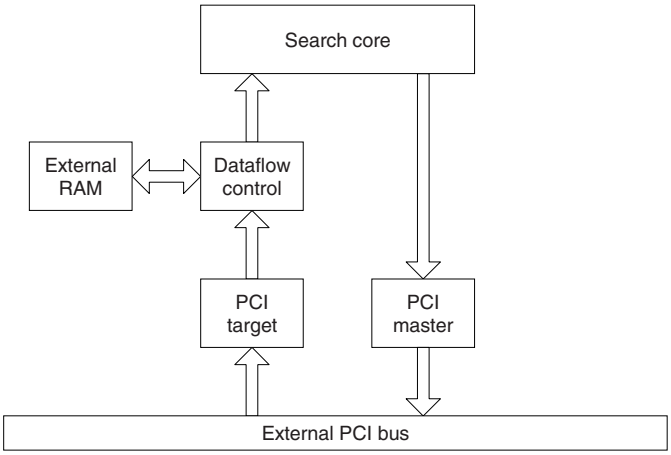

Figure 4.2: High level block diagram of the PMC. The search core is supported by a local RAM, and connected to the host system and other PMCs through PCI.

support functions. Paper I and Paper VII provides a functional description of the PMC. This section will thus concentrate on arguing why the different design choices were made.

4.5.1 Increasing parallelism

The design start of the PMC coincided with more widespread network connectivity. Rather than building a system for each user, the aim was to provide a platform that could support multiple simultaneous applications and users. The focus was shifted from reducing single query latencies, to getting the highest query throughput achievable while still maintaining a usable response time.

The multiple user scenario also demanded more parallelism, both within each chip and across the whole system. On the application level, sub-second response time while serving more than 100 queries per second was desirable. Each user query might also map to more than one query during execution. For example, a DNA nucleotide search might automatically be expanded into variants with different specificity levels in addition to the original exact sequence search. Having thousand or more simultaneous queries was thus expected even for entry level applications, with each query containing 10 – 100 symbols.

When evaluating queries, two different process scheduling approaches were considered. One would be to have one processing element for each

query symbol, just like the MS160 architecture. This approach is easily modeled, given the predictability of what each PE is doing at any time. In some applications, this approach would waste a lot of resources. For example, if monitoring headers in network traffic, it would only be meaningful to look for a specific symbol in a single position of the header. For a n-byte header, each PE would only do necessary work for every n'th data cycle.

The other approach considered had each PE controlled by an instruction stream representing a part of, or even a whole query. Each PE could be implemented as a state machine as described in section 3.2.2. This would increase the processing utilization, at the expense of a more complex system and programming model. With PEs executing different programs, they might also have different data rate demands. Running multiple queries on the same data stream might thus imply that some queries was still not utilizing the processing capabilities to their limits.

As discussed in chapter 3, restructuring computation for parallel execution is likely to be performance limited by the available data bandwidth. Further, the available transistor count on a single ASIC implied that "wasting" PEs was not expensive. The limiting factor for the integration level would be the power consumption, not the transistor count as such. Idle PEs would require much less power than active ones.

Our choice was thus to stay with the one PE per symbol architecture inherited from the MS160 design. This was also the only legacy we kept from the previous research. With each PE dedicated to a specific symbol of the query, it would not change roles during a search. There is thus no need for changing instructions, reducing instruction storage and scheduling down to a set of configuration registers being static during the execution of a query. As long as the PE does not find a match, it has no activity. The only toggling signals inside the chip are then the distribution of data, reducing power consumption.

4.5.2 Maintaining high data throughput

Processing throughput in a data streaming MISD architecture is necessarily tied to the data throughput. In the MS160 architecture, each hit generated by any query resulted in pipeline stalls. This effect would have been larger if the same principle were carried over to the PMC design, as the latter was intended to have more parallel queries.

It was evident that maintaining throughput, implied each query to be more autonomous than in the MS160. Under most conditions, finding a

hit should not stop the data stream, neither to this PE or to the others sharing the same data. Instead of awaiting CPU intervention on each hit, we chose to implement *hit managers* for each query. These hit managers are DMA engines, capable of taking the result of a hit and storing it anywhere in the addressable host system memory. Each hit manager included a small buffer to increase bus utilization through larger block transfers, and to allow processing to go on even when there was a system bus contention.

If the hit managers report hits at a higher rate than could be handled by the host system, these local buffers will fill up, and the processing must be stalled. This is obviously undesirable, but anyhow unavoidable if the host system is not able to sustain a high rate of reported hits. To reduce such effects, we chose to add bandwidth management features to the hit mangers. The most important of these were:

Compact reporting format Upon finding a hit, the PMC will report the location of the hit only, and not the data that matched the query. Limiting the local memory address space for each PMC to 32 bit, reduced the required data to one 4-byte word per hit. If the data surrounding the hit position is required, they can be read out by the CPU from the PMCs local memory with a speed comparable to what the PMC could have provided itself. In some setups, these data might already be present in the main memory of the host system, making a copy sent along with the hit position redundant.

Variable result buffer size There is often a limit for the number of hits that is relevant from an application perspective. If a query is too unspecific, returning numerous results, the usability of the results might be reduced. The PMC was thus given the capability of letting the application decide the number of desired results per query by allocation variable length result buffer sizes. When more hits than what fit the allocated buffer are found, the PMC will inform the host system, which can either extend the buffer, or terminate further processing of that specific query.

Hit decimation Not all applications needs to know the exact position of each hit. For example, one might use a motif describing protein encoding DNA regions as a query across the genome of an organism to find the chromosome regions that are more "gene-rich" that others. Presented as a histogram along the sequence, there is no need for single nucleotide resolution. A hit decimation mode was thus added, with a programmable decimation of the number of hits.

4.5.3 Balancing memory volume with processing power

Using linear access to search through the local data of each PMC, the amount of memory would also affect the processing time. As discussed in section 4.4.2, previous designs was not utilizing the memory chips to their full potential. The capacity per memory chip had also increased dramatically. A low memory-to-PMC chip ratio was thus used. The upper limit was chosen to 8 RAM chips per PMC, as this still allowed for 256 MB per PMC initially, and with an upgrade option up to 4 GB as denser memory should become available. Typically, the number of memory chips would not even need to be taken to the maximum limit.

4.5.4 Memory interface

With a relatively low number of RAM chips per PMC, it did not make sense to have a separate memory controller. Integrating the memory controller into the PMC implied better bandwidth management. The controller was designed to guarantee a sustained data rate to the search core exceeding the processing rate of 100 MB/s. This was achieved by having a 16-bit memory interface operating at the same frequency as the PMC itself, with a burst speed of 200 MB/s. Some of this capacity is lost due to memory refresh and protocol overhead, but there would still be more than 50 MB/s available bandwidth from the host system to the local RAM of each PMC even during search, which could be used for reading or altering the memory content.

In some applications, for example monitoring of real time data like network traffic, storing data into the local memory of each PMC before doing the actual processing would add to the query latency. To minimize this, it was decided to add a bypass mode in the memory controller, where data could be streamed directly from the system interface and into the search core.

4.5.5 Data distribution

Data distribution within the PMC is sourced from a shared stream. Two functionally different distribution aspects needed to be served: Distribution to parallel independent queries, and between the individual PEs that would cooperate on evaluating one query. The latter is the most difficult, as restrictions in which PEs that can communicate directly affects the allocation granularity of the PEs themselves. Two distribution alternatives was considered.

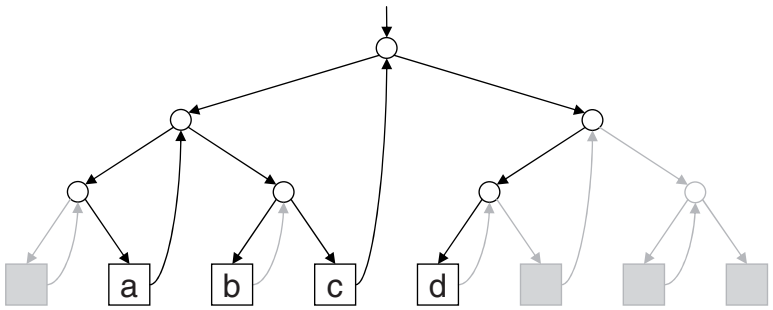

Figure 4.3: Schematic view of the data distribution for the query $a(b+c)d$ through a binary tree. The structure ensures sequential order of evaluation, at the expense of not utilizing all PEs. Unused resources are shown in gray.

The first implied a synchronized broadcast of the data stream to all the PEs. Such a distribution scheme would be easy to implement, and each PE could be allocated independently from a data distribution perspective. The main drawback was the lack of a native evaluation order between the PEs. A query like $a(b+c)d$ would require the PEs looking for b through d to know if a match occurred one respectively two clocks after an a had a match. In this scheme, the PEs needed some sort of history storage.

As the result processing was chosen to be best served by a binary tree (see section 4.5.7 below), a similar scheme was also considered for the data distribution. By pipelining data through the PEs looking for a sequence of symbols, the evaluation order is inherently guaranteed. The binary tree structure does although limit which PEs that can be chained. Figure 4.3 shows the data distribution for the query $a(b+c)d$, requiring eight PEs for a four symbol query.

Such worst case doubling of resource usage was initially considered prohibitively expensive. By modeling the transistor count impact of a broadcast distribution combined with a history buffer; alternatively adding further paths to the binary tree structure; we found that the added complexity increased the processing elements size more than what could be reclaimed in higher PE utilization. To get the same processing capacity, it took less power and transistors to have an abundance of PEs that was not fully utilized, than having more complex but fully utilized units.

The binary tree distribution scheme was thus selected, with Figure 4.5 showing the data distribution for the whole search core. In the physical implementation, the number of multiplexers was reduced to minimize combinatorial delays. Instead of a large number of two-input multiplexers,

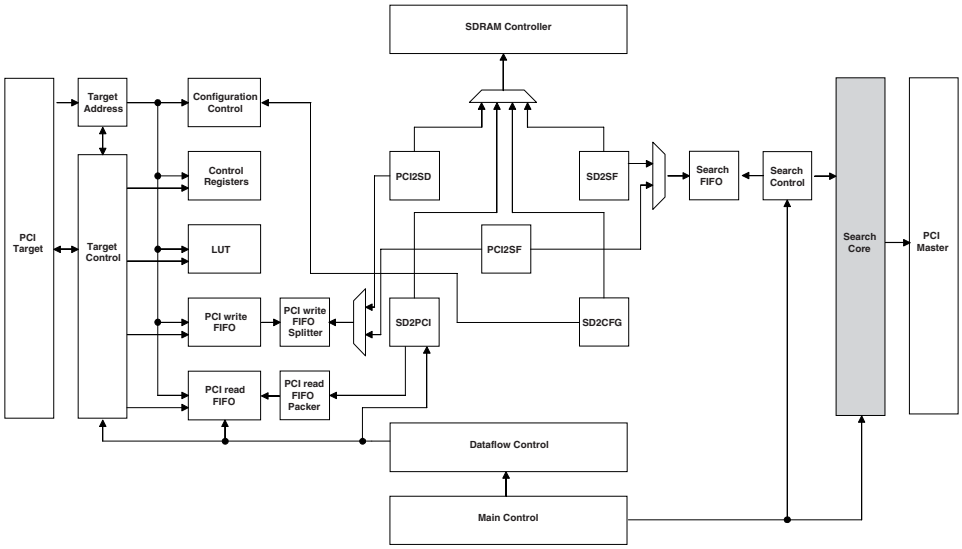

Figure 4.4: Block diagram of the PMC. The search core (shaded block) is expanded into more detail in figures 4.5 and 4.6.

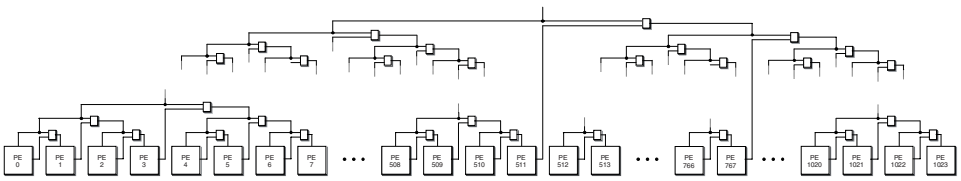

Figure 4.5: Block diagram of the PMC search core data distribution. For readability, not all levels of the tree are shown.

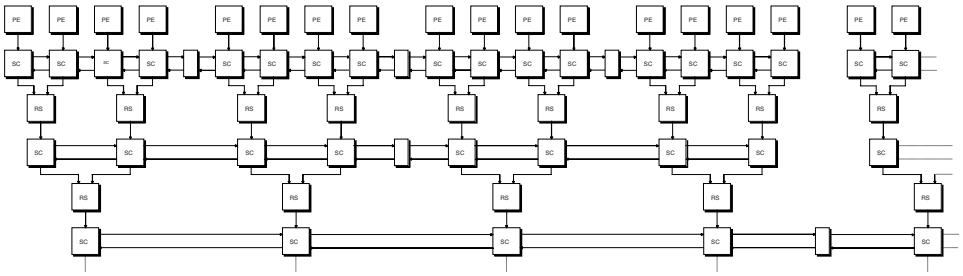

Figure 4.6: Block diagram of the PMC search core result processing. The result from individual processing elements are fed through a tree of sequence control units (SC) and result selectors (RS). Pipeline registers are inserted between every fourth sequence controller on the same level to avoid long combinatorial delays. Only the first three out of eleven levels in the tree structure is shown.

fewer levels of wider input multiplexers were used. The collapsed structure is although functionally equivalent to this figure.

4.5.6 Processing elements

Partly due to the data distribution scheme chosen, an abundance of simple PEs were required. The previous architecture used upper and lower range comparisons in each PE, checking if the current data value $x \in [a, b]$. As ranges had been infrequently used, and could be evaluated by combining two single range comparisons, the PMC PE design was simplified to only have one reference value a for comparison. The comparison mode could be any of the following Boolean functions:

$$f_1(x) = (x \leq a) \tag{4.4}$$
$$f_2(x) = (x \geq a) \tag{4.5}$$
$$f_3(x) = (x \neq a) \tag{4.6}$$
$$f_4(x) = (x = a) \tag{4.7}$$

The output of the $f_4(x)$ function was decided to be always available in parallel with one of the three others, as this allowed magnitude comparisons for values spanning more than one byte.

4.5.7 Result processing

While the PEs execute the value range comparisons, it is the combination of results from multiple PEs that allows for complex queries. The dominating functions carried out in the result processing stage is sequence control between individual PEs; also supporting nonadjacent matches; and combining partial results through some function, for example a Boolean AND of the PE outputs.

For every PE to be fully utilized, an arbitrary number of PEs should be able to cooperate in a query. As for data distribution, full flexibility in the result processing would be very resource intensive. Measured by the number of transistors, the result processing would in any scheme envisioned be the largest part of the search core. Keeping the result processing tree busy was thus more important than utilizing all PEs for overall throughput. Reducing the number of potential points where results processing could take place was thus mandatory.

Thus a binary tree structure as shown in Figure 4.6 was evaluated and found suitable. This implied 2^n PEs allocated to any query, with $n \in \mathbb{N}^*$. Due to the complexity of the result processing, pipelining was needed to achieve the desired clock rate. Pipeline registers was inserted to break up combinatorial delays sufficiently. Higher clock frequencies could have been achieved by further pipelining.

4.5.8 System interface

The most important system interface parameters investigated was bandwidth and latency. At the same time, having system compatibility with a wide range of host systems left only one real alternative, PCI. Vendor specific interconnect systems existed that had higher performance than PCI, at the expense of being unique for a specific server architecture. PCI was used not only at the add-in board level, but all the way to each individual PMC chip.

PCI also allows more than just passive interconnect. If required, the path between a group of PMCs and the host system can be split by a processor that can distribute data and aggregate results, offloading the main CPU.

With an order of hundred PMCs in a system, each with local memory, 64-bit PCI addressing was chosen to simplify the mapping of PMC resources into the system memory space. Ideally, all of the memory on the PMCs should be available as a linear range to the host. Initial testing showed that most systems had BIOSes that did not allow for multiple large

memory regions on PCI, even though this technically should be possible. To overcome this compatibility issue, we instead chose to only expose a 64 kB memory page from each PMC at any time. The overhead of switching the page pointer was considered negligible. The main mode of memory access is bulk sequential uploading data to the PMC, and not random access.

4.5.9 Query configuration

The programming model for the PMC will be discussed in section 4.7. From a hardware perspective, a distributed program storage, or rather configuration settings, was chosen. The configuration would be static during a query, but could be loaded from the local memory. Multiple compiled PMC configurations could be stored in this memory, and activated by a load command. Alternatively, all of the individual configuration registers could be manipulated directly through the host interface. These registers hold the settings that otherwise would have been decoded from the instruction stream in a conventional architecture.

4.5.10 Scalability

The PMC was intended to be a building block for larger systems, not a one chip solution. Packing the maximum possible processing power into each individual chip was thus not top priority. The focus was rather put on getting the maximum processing density and power efficiency in an assembled system.

There is no architectural limits to the number of PEs that each PMC can hold. The overall binary structure makes scaling of the design easy for any size of 2^n for $n \in \mathbb{N}^*$. The final sizing to 1024 PEs per PMC was dominantly determined by the resulting die size and power consumption. Fitting into relatively small ball grid package (14 by 14 mm, 1.4 mm thick), and using less than 1 W of power, resulted in a chip that could be mounted in any industry standard enclosures without added cooling systems.

The PEs within each PMC exhibit fine grain parallelism, which also could be said for the PMCs themselves. It is not the performance of each chip that is important, rather that thousands of PMCs can be assembled into a handful of regular host server enclosures and used as a coherent system.

The tight coupling of the PMC and RAM is close to the concepts of intelligent RAM (Patterson et al. 1997), although not fully integrated on the same die. The main reason for not choosing a hybrid solution, was that

processing and memory on the same die implies both reduced processing performance and storage density. For the PMC project, having external RAM was rather an advantage, as advances in memory chips could be utilized immediately.

Choosing the number of PEs per PMC implies a hard limit to the maximum query size that can be evaluated natively by the hardware. A single query can not span across multiple chips in the current implementation. Queries needing more than 1024 PEs was considered unlikely, and can also to some extent be modeled by splitting the query into two or more fragments, followed by merging the partial results. Further, the off-chip communication between different PMC cooperating on a large query would have had significantly different timing compared to the internal pipeline. The small potential application usage for such intra-chip communication, and the large risk of this becoming a bottleneck, led us to not include such a feature.

4.6 Designing the PMC

Starting on a project to develop a custom processor, using an unproven architecture, with no system software in place, implied a large risk for failure, especially with only a handful people in the team. To manage this risk, we sat up a series of acceptance tests in the development plan.

4.6.1 Extensive acceptance testing

Without any real silicon implementation availably until late in the project, these tests would to a large extent be emulating the final system. Software emulation would never be able to reach the same processing power as the final system, implying restrictions to how extensive testing that could be carried out. The acceptance criteria were divided into the following categories:

Physical constraints These criteria were the easiest to formulate, including aspects like die size, power dissipation and pad drive strength for signals. Even if noncompliance could affect all the other tests, the compliance checks themselves would require dedicated measurements on the more analog level, as described in section 4.6.4.

System compatibility The PMC is just one part in a large system, where we would prefer to have flexibility in choosing the other components such as host system platform, operating system, or even what brand

of SDRAM to use. Although most if these aspects are governed by industry standards, a lot of effort was put into making sure that we adhered to those standards down to the smallest details. During this process several non-compliant issues were discovered in other vendor's products, for which we needed to make a workaround without affecting the general compatibility.

Functional tests These tests were built to test individual functional blocks of the PMC. As not all blocks would be directly observable in the final product, preference was given for tests where the pass or fail criteria could be judged by the output of the chip, in most cases the result listing of a search.

Robustness Since the PMC contain several different types of functional blocks, and most block classes have multiple instances, making functional tests for all possible configurations was not feasible. Instead we made a *random* test generator, building configurations that were tested on at least two of the test benches described in section 4.6.2. Even though the result of such a configuration could have no practical usage, running it though different test benches should produce the same results. This also served as a way to make sure that the system models we used were accurate.

Such grouping of the tests made it more containable to find the cause of an unexpected behavior during the debugging phase. For example, we would not try to hunt down failing functional tests before being confident that no system compatibility issues were affecting the situation. It also eased the communication with the subcontractors for the PMC layout and production, as we had a defined set of criteria to use.

4.6.2 Hardware and software co-development

To avoid a long hardware development phase, followed by a software development cycle, it was decided to run both development phases in parallel. This implied software development would need to start without having any hardware to test on. Since this was a novel architecture, we had to build any software related specifically to the PMC from scratch, such as system drivers, query language compilers and resource management. Some of this did not need real hardware to be tested, but for the majority of the components there was a dire need to get a PMC model in place.

The following three models were produced to satisfy the demands in the different development phases.

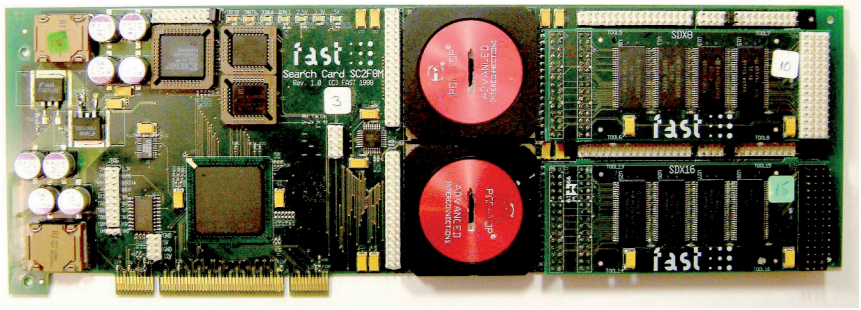

Figure 4.7: Test jig for the FPGA version of the PMC. The card holds two FPGAs, each running a miniaturized version of the final ASIC. Memory is mounted on pluggable modules, to enable testing of different SDRAM vendors. The FPGAs are configured from a flash memory chip, which can be reprogrammed in-system.

Software simulator A functional software model of the PMC was built first. This provided a way to test the different functional blocks, and verify that it was possible to map all the query language constructs into the hardware.

Verilog model The functional Verilog model served as the reference description of the PMC architecture. It also provided a well defined system when starting to develop the actual PMC design, written in VHDL.

FPGA implementation While the software and Verilog models could be built rapidly, and provided testability for the functionality, they both lacked the performance to run any real application tests, much less the robustness tests outlined in section 4.6. Thus a FPGA model of the final ASIC was built, including a PCI board to place into systems for testing (see figure 4.7). This board also served as the test bench for system compatibility testing. The FPGA was built with the same VHDL as the ASIC, but with only 64 processing elements. Furthermore, it could only run at 33 MHz, and with two FPGAs on a full size PCI card. With 8 times more chips per card, 3 times the clock speed, and 16 times more internal resources in the ASIC version, the FPGA was not an accurate representation of the final product. However, it still provided sufficient processing power to become a valuable debugging tool in the ASIC development.

Based on these model systems, we were able to build a complete ver-

sion of the required software solutions in parallel with the development of the hardware. The experience gained from these tests enabled optimization of parameters such as buffer sizes inside the ASIC, getting maximum throughput.

4.6.3 ASIC and hardware system co-development

Although the ASIC was the main development task, there was also a need to build the supporting hardware platform. As with the software development, this also took place in parallel. With trial layouts for the production board, the design of the ASIC could be optimized from a systems perspective, and not as a component by itself. Some of the most important issues covered were:

Power pad placement was not done to reduce the inductance of the packaged ASIC alone, but rather the effective inductance when mounted into the board. The low inductance package pins was often not within reach of a bypass capacitor. Choosing a higher impedance package pin, for which it was possible to place a bypass capacitor nearby, provided a better overall solution. Special attention was given to the analog power supply of the built in phase locked clock multiplier in the PMC.

Signal pad placement enabled routing the high speed PMC to SDRAM interface in one layer without any vias. The lower speed PCI signals were heavily interleaved with power and ground pads, as the PCI bus could have high capacitance, and thus cause noise with simultaneous switching outputs.

Pad drive strength was selected to be just strong enough to drive the signals, but not so strong that power-hungry line termination would be needed. The only deviance for this was the clock line fed to the SDRAM chips. As this was a multi-tap signal, line termination was required for reliable clocking. Alternatively, multiple clock drivers could have been chosen, but this would both use more power as well as prohibit single layer routing.

Power dissipation would potentially have very high peaks on every clock edge, as all PEs operated in parallel. To smooth this out (on a picosecond time scale), the clock lines feeding the different functional blocks of the PMC were deliberately skewed. By ensuring that the skew was in the opposite direction of the data flow, no race conditions could

occur. A more even current draw also reduced the electromagnetic radiation of the card.

Power distribution was laid out as solid copper layers in the board, with dual ground planes. The power for input and output pads was kept separately from the core supply. Extensive power gridding was also put into the ASIC itself, providing a low inductance grid, as well as a bypass capacitance at very high frequencies. Power supply bypass capacitance was thus distributed as power grids within the ASIC covering the 10 GHz range; board power planes in the 1 GHz range; multiple low inductance ceramic capacitors directly under each component covering the 100 MHz range; larger ceramic capacitors across the board for the 10 MHz range; and finally large bulk electrolyte capacitors for everything below 1 MHz and power supply stabilization.

Most of these effects were extensively modeled for their analog properties. This effort proved valuable, as no signal integrity issues arose. In fact, radiation was so low that the testing agency for electromagnetic compliance wondered if the equipment was even turned on! Low radiation is desirable for compliance, but is also a good indicator that the system interconnect — responsible for a large fraction of the power budget in a multiple processor system — is wasting little energy.

4.6.4 Final ASIC and full scale system tests

After several years of development, it all boils down to a crucial moment: The power up of the first ASIC prototypes. Using a test card as shown in figure 4.8, specifically built for this purpose, we were able to put the first chips into sockets without soldering them to a board. After verifying that no fatal faults were present — like a complete short of the power supply — the first chip was powered up. After passing the basic register and memory access tests, complete data query tests were run successfully. It took less than 15 minutes from receiving the prototypes until we had a working system.

The full design verification and acceptance testing did obviously stretch over several weeks, but without any major issues. The time spent preparing for these tests paid back. With a test portfolio and a testable platform in place, we were also able to verify that there were wide margins for tolerating the production variability that would occur in full scale production.

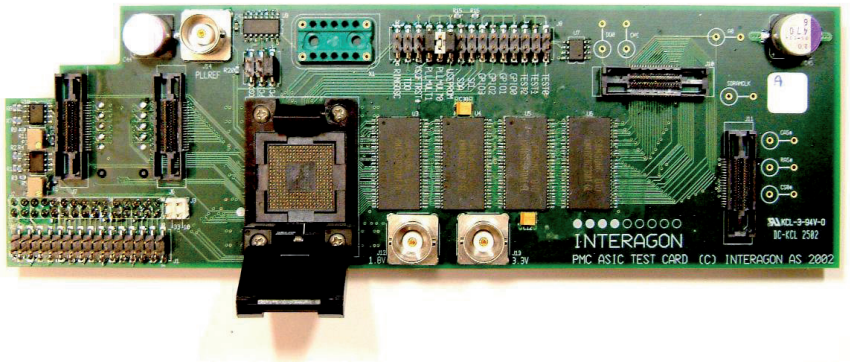

Figure 4.8: Test jig for the ASIC version of the PMC. This cards plugs into the production version card shown in figure 5.1, sharing the same infrastructure for power supply and PCI bridging. The PMC is mounted in a clamshell socket. To achieve testability, this card contains probe points to all signals, most of which would be inaccessible on the production card. The PMC can be clocked from other sources than the PCI clock. The card also enables measurement of the real time current consumption.

4.7 Programming model

Each PMC consists of 1024 PE, an equal number of result selectors, as well double this number of sequence controllers. Multiplied by the number of PMCs per host system, there are potentially several hundred thousand instructions active at any time. This is orchestrated by the host CPU. To illustrate the the flow and life span of such instructions, one of the applications solved by the PMC is used as an example: Screening siRNA sequences. The molecular biology foundation for this is explained in more detail in Paper IV. This section discusses the program flow.

The user input to such an application can either be a DNA or RNA sequence, or even an accession number that would be used for looking up the actual sequence in a data base. A typical sequence length is some thousand nucleotides long. As an example, the following description will use a sequence that start with `actcggcttcctcctcct.....`

While this example example will use a deterministic way to program the PMCs, queries have also been generated through genetic programming, further elaborated in Snøve Jr. (2005) and Sætrom (2005).

4.7.1 Interagon Query Language

Based on the above sequence, siRNA candidates can be chosen as any contiguous subsequence. The number of siRNA candidates are thus in the same order as the sequence length. While real siRNA molecules are around twenty nucleotides long, a length of seven will be used in this description for readability.

The uniqueness of each candidate can be established by generating a number of queries based on the sequence, and check if these have any matches in a larger data base like the transcriptome. All of these queries must be formulated in a query language specifically crafted for the PMC, the Interagon Query Language (IQL), a variant of regular expressions. For more details on IQL, please refer to Paper VII and Nedland et al. (2002a).

Starting with the first siRNA candidate, that is the first seven nucleotides of the sequence, a query for an exact match is easily formulated in IQL as

```
actcggc                                    (4.8)
```

Further queries can allow for one, two or three mismatches in the sequence, formulated in IQL by the pattern modifier *p*, as

```
{actcggc:p>=6}                             (4.9)
{actcggc:p>=5}                             (4.10)
{actcggc:p>=4}                             (4.11)
```

RNA hybridization is also affected by a phenomenon called GU-*wobble*. Uracil (U) is the RNA equivalent for thymine (T) in DNA. While guanine (G) has the largest affinity for binding with cytosine (C), it will also form weaker bindings with uracil, creating a GU-wobble. Accounting for these bindings, thymine might take the place of a cytosine nucleotide during hybridization. Equivalently, adenine (A) might be substituted by guanine.

Requiring that all seven nucleotides in the siRNA has a binding, but where one through three respectively could be a weaker GU-wobble, are specified by the IQL queries 4.12 through 4.14. The first half require (at least) six through four perfect matches respectively, while the latter half require any position to have a perfect match or a GU-wobble. The total query will thus achieve the desired effect.

```
{actcggc:p>=6} & (a|g)(c|t)t(c|t)gg(c|t)    (4.12)
{actcggc:p>=5} & (a|g)(c|t)t(c|t)gg(c|t)    (4.13)
{actcggc:p>=4} & (a|g)(c|t)t(c|t)gg(c|t)    (4.14)
```

Further variants could allow for a combination of complete mismatches and GU-wobbles, exemplified by 4.15 allowing one miss combined with two GU-wobbles.

```
{actcggc:p>=4} & {(a|g)(c|t)t(c|t)gg(c|t):p>=6}
```
(4.15)

There are further variants that could be of interest. Additionally the reverse complementary queries must also be formed, as DNA have two complementary strands where either could be affected. For each siRNA candidate there are thus in the order of 10–20 required IQL queries. A siRNA screening for a single gene; with a typical sequence length of 1000–5000 nucleotides; would generate in the order of 10000–100000 queries.

One could argue that only the most unspecific of these queries should be used in an initial screening for each of the subsequences. If this has no alignments, then neither of the more specific queries will generate any matches. Due to the small number of nucleotides, and the large number of reference data to screen against, it is although more likely that all, even the most specific query, has numerous hits. It is rather the statistical hit distribution of the cohort of queries that are of interest, than individual results. Thus it is sensible to run all queries simultaneously.

4.7.2 Code compilation and mapping

While the above step included application specific logic for mapping a DNA sequence into IQL queries, the remaining steps are generic for all PMC applications. The assembly of queries are fed to a compiler that generates the configuration for each query, and the mapping of this on to a subtree of the search core, exemplified in Figure 4.9. Such subtree configurations are combined to utilize as many PEs as possible within each chip.

Subtrees using the same data for input, in this case all of them, will be clustered into the same set of PMCs, potentially even a single chip. The management software will distribute the required reference data to the relevant PMCs, and reuse information that is already loaded in the local RAM when possible. In the case of siRNA screening, this is rather trivial. All queries should see the same reference data. When iterating through the required query sets, no reference data needs to be altered.

4.7.3 Multi-threaded execution

The host system generate the queries, the cohort of PMCs evaluate them, and the host system aggregates the results, for example in presentation to

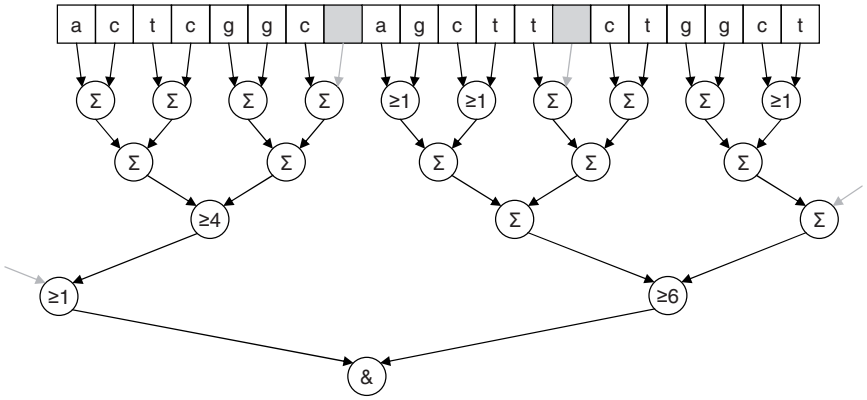

Figure 4.9: One possible mapping of the query given by equation 4.15 on to a subtree of the search core. The data distribution tree is not shown, but would consist of a combination of parallel and serial distribution, with the data flowing right-to-left. To align with the binary tree structure, some parts will be unusable. These are shown in gray, or pruned away. This query would need to allocate all the 32 PEs being leaf nodes of the final logical AND operation.

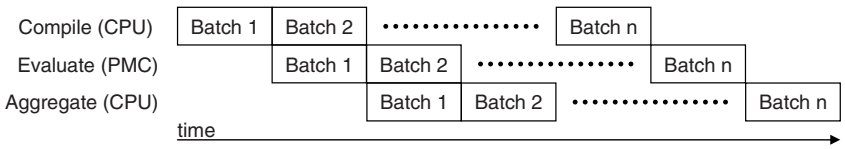

Figure 4.10: Query evaluation involves query compilation, evaluation, and result aggregation. These steps can be pipelined to achieve maximum throughput. The evaluation is executed by the PMCs in the system, with the other tasks served by the host CPUs

a user. These steps can be pipelined for overlapping execution in multiple threads as long as the result of one query does not affect the query that will run immediately afterward. When the number of queries exceed the instantaneous capacity of all PMCs in the system, a query queue is unavoidable, but can be executed with near 100% PMC resource utilization as illustrated by Figure 4.10.

4.8 Distance metrics

From the architectural description in Halaas et al. (2004), it might look like the PMC is only suited for string related search applications. This is not the case. The PMC can also work with vector or multiple attribute data, whereof text search is a sub-class. This section will describe the distance metrics used for specifying how a multidimensional data value will be matched with a non-exact query. Multidimensional values could be the exclusive content of a query, but also interpreted in combination with any other fields.

4.8.1 One dimensional data

Each PE in the PMC can be individually programmed to compare data with a specified value for that query. In addition to checking for equality, *larger than* and *less than* operators can be used. By using two PEs, a range of accepted values for each symbol can be specified. The lower and upper bounds does not have to be symmetrical around a reference value. In fact, only the bounds need to be specified.

For the following discussion it is assumed that the reference value is centered between these bounds to simplify the equations. Since the PMC does not relate to this virtual reference value at all, the results would be the same even if the real reference is off-center.

In the one-dimensional case, the deviation δ between a query value u, and the observed value of v, would be calculated as

$$\delta = |v - u| \tag{4.16}$$

4.8.2 Multidimensional data

Expanding into two dimensions infers multiple potential metrics for calculating distances. The Euclidean distance is commonly used, defined as

$$\delta_2 = \sqrt{x^2 + y^2} \tag{4.17}$$

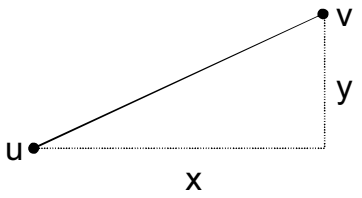

Figure 4.11: Decomposition of Euclidian distance into x and y components in a two-dimensional coordinate system.

where x and y are the differences along each of the two axis as shown in figure 4.11.

Another common metric is the Manhattan distance, defined as the shortest distance between two points when only moving in parallel to any axis, that is

$$\delta_1 = |x| + |y| \tag{4.18}$$

Both of these are special cases of the Minkowski p-metric (Royden 1988; Tzionas et al. 1994), defined in two dimensions as

$$\delta_p = [|x^p| + |y^p|]^{\frac{1}{p}} \tag{4.19}$$

where $p = 1$ for the Manhattan metric, and $p = 2$ for Euclidian distances. The Minkowski p-metric is defined more generally for n dimensions by

$$L_p(u,v) = \delta_p(u,v) = \left(\sum_{i=1}^{n} |u_i - v_i|^p \right)^{\frac{1}{p}} \quad \text{where} \quad p \in [1,\infty) \tag{4.20}$$

where u_i and v_i are the coordinates in each of the n dimensions for u and v respectively.

When $p \to \infty$, the Minkowski metric is dominated by the component with the largest difference. In the extreme case it is simplified to *only* be dependent on the maximum difference found in any of the components, as given by

$$L_\infty(u,v) = lim_{p\to\infty} \left(\delta_p(u,v)\right) = \max_{i=1}^{n} |u_i - v_i| \tag{4.21}$$

The relationship between L_1, L_2 and L_∞ in two dimensions is shown in figure 4.12 for distances of two units. Since

$$L_1(u,v) \geq L_2(u,v) \geq L_\infty(u,v) \tag{4.22}$$

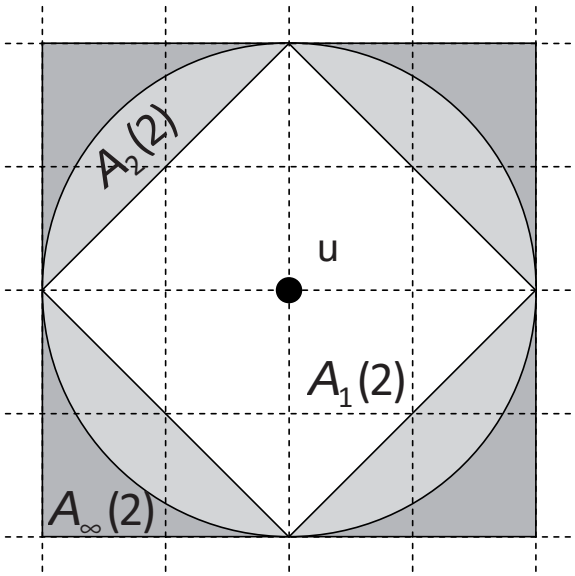

Figure 4.12: Comparison of the L_1, L_2 and L_∞ metrics in a two-dimensional coordinate system. All points within the circular area of $A_2(2)$ are within an Euclidian distance (L_2) of two units from u. The inscribed rectangle defining $A_1(2)$ contains all points within a Manhattan distance (L_1) of two units. The outermost rectangle defines $A_\infty(2)$, the set of points that all are within two unit distance from u according to the max-metric L_∞.

and thus

$$A_1(d) \subseteq A_2(d) \subseteq A_\infty(d) \tag{4.23}$$

where

$$A_p(d) = \left\{ v \in \mathbb{R}^n \mid L_p(u,v) \leq d \right\} \tag{4.24}$$

the L_∞ metric can be used as a prerequisite for finding data within a given distance by the other metrics.

4.8.3 PMC distance metrics avoid expensive arithmetics

Applications can have different requirements for computing the distance between two values. When comparing for magnitude, n'th roots can be omitted as long as the true distance is not required. Multiplication and addition are still required to find the inner sum in equation 4.20. Single clock cycle multipliers require many transistors, even for low word widths. Supporting all variants of L_p in the PMC would require in the order of 1000 transistors for each result selector on the lowest level just for the multiplications. Higher level result selectors would be even larger due to wider word widths.

Such an arrangement would easily use several million transistors in the PMC, and was thus considered unfeasible. There was a need for a simpler distance metric, that still would be a required conditions for all L_p metrics.

Since

$$\begin{array}{rcl} L_p(u,v) & \leq & C \\ & \Downarrow & \\ L_{p+1}(u,v) & \leq & C \end{array} \tag{4.25}$$

as given by 4.22, L_∞ can by induction be used as a required condition for all Minkowski p-metrics. The PMC uses an extended version of the L_∞ metric when matching queries and data, given by

$$\begin{array}{rcl} L'_\infty(u,v) & = & \max_{i=1}^{n} \left[w_i \alpha_i \left| u_i - v_i \right| \right] \\ & \text{where} & \\ w_i \geq 0, \alpha_i \in \{0,1\} & , & \sum_{i=1}^{n} \alpha_i \geq m \end{array} \tag{4.26}$$

There are two differences between equation 4.21 and 4.26. First is a weighting factor w_i, allowing for different dynamic range in each component. Implementation-wise, this is the upper and lower bounds used for

checking each component. The valid range is inversely proportional to w_i, and defined during query compile time.

Secondly, only m out of the n components contributes to the metric, formalized by $\alpha_i = 0$ for disregarded components, and $\alpha_i = 1$ for those contributing to the L'_∞ metric. All possible sets of α_i where at least m members are 1 are evaluated during execution.

The L'_∞ metric was chosen for the PMC as it requires few transistors evaluate if a value is inside or outside the n-dimensional cube defined by a distance C from the reference value. As

$$\begin{aligned} L'_\infty(u,v) &\leq C \\ &\Updownarrow \\ \alpha_i \left| u_i - v_i \right| &\leq \frac{C}{w_i} \;\; \forall \;\; i \in [1,n] \end{aligned} \tag{4.27}$$

the calculation is reduced to a Boolean AND (masked by α_i) of the value comparison for each component. Supporting any other variant of the L_p metric natively would have required considerable more resources. This design avoids the expensive multiplications, and replaces them with much simpler value comparisons. Even L_1, which could be implemented by adders only, is more complex. L'_∞ also fulfills the conditions for a metric, that is non-negativity, identity, symmetry, and the triangle inequality.

$L'_\infty(u,v) \leq C$ can be used as an easily calculated *necessary* precondition for a *sufficient* criteria $L_p(u,v) \leq C$ for all values of p. There is also a well defined bound $C' < C$ for which

$$\begin{aligned} L'_\infty(u,v) &\leq C' \\ &\Updownarrow \\ L_p(u,v) &\leq C \end{aligned} \tag{4.28}$$

Exact calculation of $L_p(u,v)$ to determine if this distance is within a given boundary C, can thus be reduced to only involve the points (u,v) where

$$C' \leq L'_\infty(u,v) \leq C \tag{4.29}$$

As an example based in Figure 4.12, finding the set $A_2(2)$ of all points v within a Manhattan distance of 2 from u implies $C' = \sqrt{2}$ and $C = 2$, and classifies any point v by principally using L'_∞ as a metric:

$$L'_\infty(u,v) \leq \sqrt{2} \;\; \Rightarrow \;\; v \in A_2(2) \tag{4.30}$$

$$\sqrt{2} < L'_\infty(u,v) \leq 2 \;\; \Rightarrow \;\; v \in A_2(2) \;\; \text{if} \;\; L_2(u,v) \leq 2 \tag{4.31}$$

$$2 < L'_\infty(u,v) \;\; \Rightarrow \;\; v \notin A_2(2) \tag{4.32}$$

Table 4.1: Technical data for different string search ASICs (adapted from Blüthgen and Noll (2000) and Paper I).

System	Number of PEs	Number of transistors	Silicon area	Maximum clock rate	Character comparisons
PMC	1024	1.17×10^7	114 mm^2	100 MHz	1.024×10^{11} ch/s
Blütgen[1]	64	3.4×10^5	53 mm^2	132 MHz	8.448×10^9 ch/s
Kestrel[2,3]	64	1.4×10^6	60 mm^2	33 MHz	7.04×10^8 ch/s
CASM[4]	7	2.047×10^4	30.7 mm^2	40 MHz	2.8×10^8 ch/s
SSE[5,6]	512	2.176×10^5	110 mm^2	10 MHz	5.12×10^9 ch/s

[1]Blüthgen and Noll (2000) [2]Dahle et al. (1997) [3]Hirschberg et al. (1998)
[4]Sastry et al. (1995) [5]Yamada et al. (1987) [6]Hirata et al. (1988)

Only 4.31 require the computationally more demanding $L_2(u, v)$ metric, which can not be calculated by the PMC. Thus the L'_∞ metric will be used to find which of the three cases each point v belongs to. For those that fall within the range given by equation 4.31, the CPU must calculate the $L_2(u, v)$ metric. This is manageable as long as the fraction of points within this category is limited.

With the L'_∞ metric, the PMC can handle multidimensional data with different dynamic range in the components. The "at least m out of n" feature has proved valuable for handling noisy data, as will be exemplified in chapter 5.

4.9 Results

Table 4.1 summarizes the performance of the PMC compared to other string search ASICs. Further details of this comparison are given in Paper I. The resulting performance of the PMC chip as a system component has been published in Paper II and Paper V. Further application examples are described in chapter 5. The PMC has also been the enabling technology for numerous publications (Hetland and Sætrom 2002, 2003, 2004; Sætrom et al. 2005a; Sætrom 2004, 2005; Sætrom and Hetland 2003a,b; Sætrom and Snøve Jr. 2004; Sætrom et al. 2005b; Sandve et al. 2006; Snøve Jr. 2005; Snøve Jr. and Holen 2004; Snøve Jr. et al. 2004). The PMC technology has been granted two patents; one for the general architecture (Fast Search & Transfer ASA 2000b); the other for the current implementation (Fast Search & Transfer ASA 2000a).

Based on these results, the PMC accelerated machine could be argued to have found important practical use. In the next chapter, some of the applicable problem domains for the PMC will be discussed.

Chapter 5

Applications for the Pattern Matching Chip

Some day, on the corporate balance sheet, there will be an entry which reads, 'Information'; for in most cases, the information is more valuable than the hardware which processes it.

Grace Murray Hopper (1906–1992)

MISD architectures have traditionally not found widespread use in supercomputing applications (van der Steen and Dongarra 2004). However, for a new breed of search problems, building upon the growing volumes of unstructured data, novel MISD architectures may still have a role to play. For example, bioinformatics (Feng 2003) and internet search engines (Risvik and Michelsen 2002) require matching such unstructured collections with numerous queries. Consequently, parallelism can be exploited through both query and database segmentation.

While one could justify developing a MISD architecture solely for exploring the "dark corner" of Flynn's taxonomy, this technology has also enabled applications to get valuable information from already existing data.

5.1 Introduction

This chapter presents different applications enabled by regular expression-like searching accelerated by parallelization across multiple chips, cards, and computers to achieve supercomputing performance (Paper II; Paper I). Some of these applications have been previously published in detail in papers related to this thesis (Paper V; Paper VI; Paper IV), as well as by others (see section 4.9. Further application examples are given in this chapter.

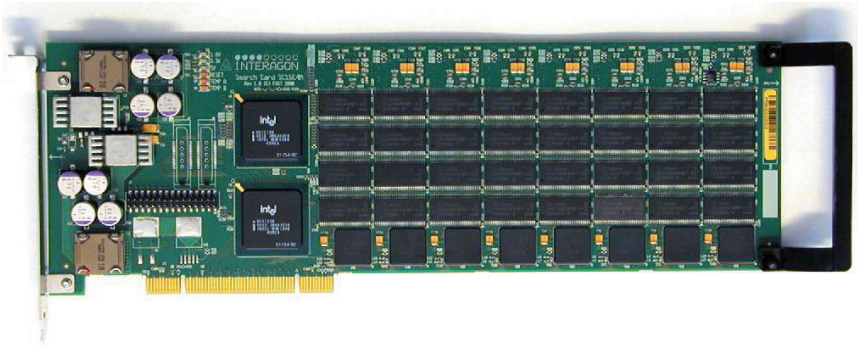

Figure 5.1: PMC search card with 16 chips and associated memory. Eight PMC chips are mounted along the bottom edge of the card, while another eight are mounted along the top edge on the reverse side.

All application examples are running on a shared hardware platform. Sixteen PMC chips are mounted on PCI search cards, and these are combined in a cluster of up to five machines — dependent of the problem at hand — containing between one and six cards each, resulting in a system with up to 480 search processors. The system is further linearly scalable, with no fundamental cluster size limit.

The five PC cluster was designed to solve search problems that can be divided into sub-problems that can be solved in parallel with minimal overhead. Typical applications consist of a large number of queries, and works with (relatively) static and unstructured datasets. Further, data mining based on genetic programming can be accelerated by our system (Koza et al. 1999; Sætrom 2004). Example applications include data mining applications in genomics, network surveillance, and transaction monitoring.

5.2 Characteristics of suitable applications

Throughout our work we have applied the PMC technology to numerous applications. Despite working in very different problem domains, a shared set of commonalities has made the PMC applicable:

No known index key The data processed could be classified as *unstructured* data, where no (known) properties of the data could be used as an index key. For example, much of our work in bioinformatics has focused on predicting sequence properties. These predictions are guided by biological laboratory experiments, rather than know-

Figure 5.2: Rack of five nodes, containing 480 PMC chips operating in parallel.

ledge of the contributions of the different sequence elements themselves. If this was done through an index-based system, this index would have become a filtered version of the data, capturing only established knowledge about that sequence. Being relieved from the restrictions of an index, we have been able to predict and later verify novel structures.

No time for indexing Even when an index can be built, there could be real time processing requirements prohibiting the (potentially lengthy) creation of an index, such as in network communications. Processing data on-the-fly reduces the decision latency, even though an index would have been more effective regarding the amount of processing required.

Large data volumes While the above two items both are functional requirements, performance implications are not fully seen before the data volumes are cranked up. Low volume or low bandwidth data can be processed by software rather than the PMC. With large data volumes, typically in the order of gigabytes, the PMC excels not only due to the single device speed, but also the scalability when used in larger systems.

Large number of queries In some applications it is the number of queries that gives the PMC an advantage. The data volumes might fit within main memory, or even the cache, of a regular CPU but still be outperformed by a PMC solution. While a MISD architecture can apply the same set of operations across different data, it can not run multiple instruction streams within the same thread. Running simultaneous queries thus requires a combination of multiple cores, hyper-threading or context switching; alternatively an approach as discussed in section 3.2. While a 1 W PMC can run 64 simultaneous queries, an eight-core CPU will only have progress in eight evaluation threads at any time, despite using typically more than 100 W.

5.3 String data mining

The most obvious use of the PMC is within data mining in string data, both written language and non-textual data. These searches can be distributed across several cluster nodes with little overhead (Michailidis and Margaritis 2003). The PMC provides access to large sets data with great flexibility in query construction. The processing speed enables even interactive user

Table 5.1: Comparison of estimated siRNA screening capabilities of PMC systems and software algorithms. The data set is 65MB. All potential siRNA subsequences in these data are screened versus the complete dataset

	25-mers/sec	Run time	Sensitivity
BLAST	1.2	$\sim$ 3 years	$<$ 50%
Smith-Waterman	0.92	$\sim$ 5 years	100%
Melko and Mushegian (2004)	0.04	$>$ 100 years	100%
Yamada and Morishita (2005)	140	300 hours	$<$ 100%
FPGA supercomputer	470	3 days	$<$ 50%
Single PMC chip	44	35 days	100%
One PMC accelerated node	4180	10 hours	100%
Five node PMC cluster	20900	2 hours	100%

interfaces, where the results of a search can be illustrated immediately as each symbol of the query is typed in. Paper III describes one such application for genomic sequence data.

5.3.1 Bioinformatics

The focus for our PMC applications has been on bioinformatics. This field is attractive since it has a large volume of relatively static sequence data, where exact matches are a rare requirement. Sequence searching and alignment are important problems, illustrated by the fact that seven out of nine frequently used bioinformatic benchmarks belong to this category (Albayraktaroglu et al. 2005).

The Smith-Waterman algorithm is preferred for DNA, RNA and protein searching with high sensitivity, and is based on dynamic programming (Smith and Waterman 1981). As this algorithm is very time consuming, less sensitive heuristics based on indexing sequence fragments have found wide usage, like BLAST (Altschul et al. 1990).

The PMC provides both sensitivity similar to that of the dynamic programming solution, as well as higher throughput than the heuristic algorithm. This enabled us to test numerous hypotheses without too much concern for the required computation time. As an example, we completed a screening of all potential gene silencing siRNA molecules within the human genome in 10 hours on one node.

Table 5.1 compares the PMC performance in this application with results

from other groups. The common way of measuring performance for such workloads is in the number of symbol-to-symbol comparisons, also known as cell updates, executed per second. Heuristic methods attempt to avoid most of these comparisons, but their performance are still evaluated in terms of the number of comparisons that would have been needed by a brute force approach. For the problem described in this table, there are 65 million patterns of 25 symbols each, all to be compared with the same 65 million symbols as reference data in both the original and the reverses order. For each 25-mer, 3.3 billion symbol-to-symbol comparisons are thus needed, for a total of $2 \cdot 10^{17}$ symbol-to-symbol comparisons for all the 65 million possible 25-mers.

Smith-Waterman, Melko et al., and the PMC alternatives are the only methods that will evaluate all possibilities, and thus guarantee a 100% sensitivity. Numerous research groups optimize Smith-Waterman for performance. Farrar (2007) is one of the most recent studies, claiming more than 3 billion cell updates per second using a single thread on a 2.0 GHz Xeon, corresponding to 0.92 25-mers per second in this case. The same article have also tested BLAST on similar hardware; which is the BLAST performance used in Table 5.1; finding their Smith-Waterman implementation to be 30 % slower. In Paper IV we demonstrated BLAST to have less than 100 % sensitivity for such applications. The numbers for Melko and Mushegian (2004) and Yamada and Morishita (2005) are taken from their respective papers, also referenced in Table 2 of Paper II.

Mitrionics and Silicon Graphics Inc released an "FPGA supercomputer" in November 2007, using Xilinx programmable gate arrays as a hardware accelerator for BLAST[1]. Using 70 FPGAs, they ran 600.000 25-mer queries against a data base of 100 MB in 33 minutes, resulting in $1.5 \cdot 10^{12}$ cell updates per second, assuming their numbers also included the reverse sequence queries. The setup is an acceleration of BLAST, and have thus inherited the same lack of sensitivity. According to the product data sheet[2], every two FPGAs require 150 W of power, for a total of 5 kW for the overall system.

In summary, this job would have taken very long using a software implementation of BLAST, and still missing a large fraction of the relevant results. The more sensitive Smith-Waterman algorithm would have found all relevant results, but with an estimated run time around 5 years. As discussed in Section 2.1.1, major advances in CPU sequential processing speed is not expected. Speeding up Smith-Waterman could be done by us-

[1] http://www.sgi.com/company_info/newsroom/3rd_party/downloads/07_mitrionics_900x.pdf

[2] http://www.sgi.com/pdfs/3920.pdf

ing more parallel threads. Around 50 CPU cores would be required to get the same performance as a single PMC, with obvious differences in power consumption. The usage of the PMC in bioinformatics is further discussed in Snøve Jr. (2005) and Sætrom (2005).

Rapid hypothesis testing is important in an emerging field like molecular biology. The PMC enables a researcher to have more than one good idea verified during her lifetime, without investing in a large computer cluster, nor being hampered by unsuitable heuristics that can loose a large fraction of the relevant results.

5.3.2 Digital network communication

The previous section described applications with large, static data sets, and a less voluminous, but dynamic, set of queries. Paper V discusses use of PMC technology as an alternative to automata (see section 3.2.2) applied to email spam filtering.

This setup involves a moderate number of queries — rules to characterize the emails — to be evaluated against a high throughout data stream. A further complication comes from the efforts of spam providers to circumvent such rules. A rule must be flexible to catch engineered variants. Spam must anyhow be interpretable by a human to have the desired — but to the receiver still undesirable — effect. The flexibility in rule construction for the PMC eases the process of encapsulating such vague patterns into a query.

Instead of preloading the data, this setup required distribution of precompiled rules across several chips. The data stream was divided into smaller packets, each sent to one chip, which iterated all the pre-stored rules over these data. This processing would be completed before a new package was given to the same chip. Meanwhile, other chips took care of the in-between data packets. With no replication of data across chips, increasing bandwidths could be processed by adding linearly more PMC's, within the limitations of the host systems.

5.3.3 Written text

Data mining in written text with fuzzy queries is a more general variant of the bioinformatics applications discussed above. Even though more structured than DNA, it could be limited by an index. For example, the common Scandinavian name "Johansen", have numerous variations (for example "Johanson", "Johannesen", "Johannson", "Johnson", "Jonsen", all mean-

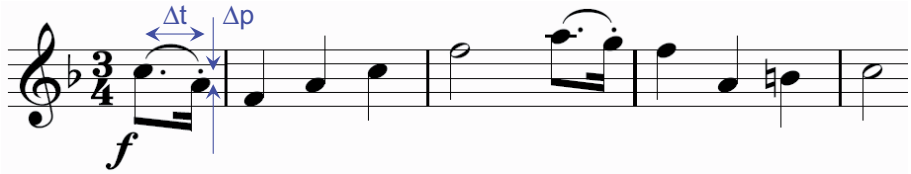

Figure 5.3: Music can be represented as a tuple series of pitch changes and durations $(\Delta p, \Delta t)$. In this representation, the above score could be written as $\left\{\left(-3, \frac{3}{16}\right), \left(-4, \frac{1}{16}\right), \left(4, \frac{1}{4}\right), \left(3, \frac{1}{4}\right), \left(7, \frac{1}{4}\right), \ldots\right\}$.

ing "son of John") that would not be linked to each other in an index without applying domain knowledge. A phone directory lookup for such an ambiguous name would actually have to be executed as multiple lookups from a synonym word list.

Optical character recognition (OCR) — reading text from an image — is another suitable example. The OCR process is not accurate, with commercial products reaching 95–99% conversion accuracy on the character level. Text archives generated from OCR — currently a large ongoing effort in building digital libraries with material from the pre-digital era — will not be exact. Retrieving documents with the PMC — for example using the n out of m feature to overcome incorrect conversions — reduces the need for manually proofreading all the text.

5.4 Vector data processing

String data as discussed in the previous section is a subclass of vector data or attribute data. As shown in section 4.8, the processing tree of the PMC can also be used to find neighboring data in a multidimensional space.

For the PMC to be efficient for such data, it is dependent on having potential usability of all the data that is streamed through the chip. Any data reduction that could be done in advance — for example processing only a fraction of the volume, or a small subset of all available attributes — would increase the possibility for *not* needing the PMC. Data reformatting could be done to build a compacted representation that resides in a linear range of memory.

Searching for music is one example of vector processing that could take advantage of the PMC. Instead of searching in the time or frequency domain, preprocessing could be applied to the data as shown in figure 5.3.

By representing music as a series of pitch changes in half-tones and durations relative to the beat frequency, it will not matter what key or tempo it is played in (see for example Arentz et al. 2005; Pickens 2001, for further details).

5.5 Genetic programming

In the applications presented so far, the queries to be analyzed are known, or specified by an operator of the system. The rapid evaluation of hypothesis provided by the PMC enabled other approaches based on trial and error. It became obvious that an automated iterative search for finding the optimal queries could take advantage of PMC acceleration.

5.5.1 Methodology and issues with genetic programming

Genetic programming is one such supervised machine learning system for solving multivariate problems. The essence of solvable problems is as follows: Given a set of data known to share common features (like being implicated in the cause of a cancer), find an expression or model that characterizes these data. This can also be stated as a problem of discrimination: One wants to find an expression that distinguishes the sequences sharing the common feature — the positive set — from sequences known not to have this feature — the negative set.

Researchers have developed many different algorithms for solving the discrimination problem. These algorithms vary both in ease of use, output quality, and output interpretability, ranging from simple expressions to complex equation sets and sets of real valued numbers.

Genetic programming — a relatively recent addition to the machine learning toolbox — has been shown to be both simple to use and able to create high quality solutions. In addition, the genetic programming solutions are symbolic expressions like programs in a programming language or queries in a query language. They are therefore easier to interpret than for instance the collection of numbers forming an artificial neural network.

Genetic programming was introduced and popularized by John Koza in 1992. It uses the Darwinian idea of evolution by means of natural selection to solve problems. It operates on a population of randomly generated candidate solutions. Each candidate solution is evaluated and assigned a fitness value that represents how good the candidate is at solving the problem at hand. Then, new candidate solutions are created by randomly

selecting candidates from the population and either combining two candidates (crossover) or introducing random change in single candidates (mutation). The selection is performed so that better candidates are more likely to be selected. This process of selection, combination and fitness evaluation typically continues until a satisfactory solution has been found or a predetermined number of iterations have been performed.

The main problem with genetic programming is the amount of computer power needed to generate solutions. Genetic programming has been successfully used, but only with small data sets using clusters containing tens to hundreds of PCs.

In addition to helping genetic programming through acceleration, the PMC provides an orthogonal query architecture, where the machine generated queries have few restrictions on composition, length and operators. The tree-based query-representation makes mutation and crossover operations simple, as these can occur on any subtree or -node.

5.5.2 Molecular biology

Most of the work done by our group with machine learning in molecular biology is discussed in detail in the PhD theses by Pål Sætrom (Sætrom 2005) and Ola Snøve Jr. (Snøve Jr. 2005). The focus has been on design and prediction of small nucleotides, such as RNA design and gene prediction.

5.5.3 Financial fraud

Financial services are moving rapidly into an electronic mode of operation. With most transactions happening without any human intervention, the expert knowledge of identifying dubious transactions is no longer in the loop. Computer systems must be trained to find common patterns in fraud, as well as having the processing power to execute such rules in a real time environment.

There are many facets of financial fraud, with credit card fraud being problematic due to the sheer volume of transactions. Credit card fraud starts with someone obtaining illegitimate card information, and afterward using this to their own benefit. Several exploitation strategies exists among the fraudsters, ranging from taking a small amount from a large volume of cards, to charging large amounts as soon as possible before the card gets blocked.

The fraud is also dynamic. As soon as fraudsters find out that their current scheme will not pass an audit, they invent something new. Analyzing

such data with conventional statistical tools has limitations. First order linear regressions will have difficulties picking up patterns in multi-factorial data with heterogeneous fraud. More advanced statistical modeling will require an expert to build a model based on a hypothesis, and later testing this out on real data. Building a model can be difficult especially in the face of evolving fraud patterns. Such models can also reach a complexity that prohibits rapid evaluation and refinement of the model as the processing time gets higher.

Genetic programming coupled with the PMC provides a solution to such problems based on two modules: i) A rule builder and ii) a rule execution engine. The rule builder generates effective rules for characterizing fraud based on genetic programming. The rule execution engine accelerates the evaluation of such rules.

The input to the rules builder is historical data, tagged with the desired classification, e.g. fraud vs. valid transactions. The solution from such processing is usually a heterogeneous model, with different parts capturing the different modes of fraud.

This concept has been tested with a major credit card company, providing classification accuracy at par, of even better than existing technologies. Unfortunately, we are not able to disclose further details on this work.

5.5.4 Seismic processing

Characterization and modeling of hydrocarbon reservoirs are predominantly based on seismic surveys, but also to some extent novel modalities such as electromagnetic logging. Seismic modeling has achieved increasing resolution and accuracy, but do still require extensive human knowledge for accurate results. The modeling can be supported with further data, such as attributes extracted from the seismic cube.

When building such models, a large number of variables could be considered. Even if this gives the modeler more degrees of freedom, it might become overwhelming to sort out which variables, or even which combination of variables, that would be meaningful. A computerized tool that builds such statistically significant models which can be judged by the analyst, has the potential of helping in discovering patterns in the underlying data that otherwise might have been overseen.

Our work has been related to lithology prediction. In case at hand, a reservoir is known to contain hydrocarbons, with the oil dispersed in heterogeneous sand layers. To achieve maximum drainage, the following sequence of drilling is beneficial:

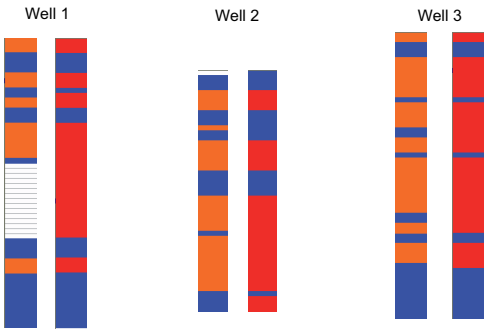

Figure 5.4: Three test wells with pairs of lithology logs (left) and predictions (rights) from training phase. For two of the wells, the data logs were incomplete (shown in white).

(i) Low permeability sand layers, which could only be drained under the initial high reservoir pressure.

(ii) High permeability sand layers, that will be sufficiently drained by the gas pressure alone.

(iii) Gas is produced after draining all the recoverable oil.

Separating the two different sand layers proved difficult based on traditional modeling. Using the seismic data, in addition to around 40 derived attribute cubes, the genetic programming platform was used to find classification patterns.

The models were found using machine learning based on training data from three test wells with known lithology as shown in figure 5.4. The predicted models had a high correlation with the training data as expected. Perfect correlations could have been achieved, but that would not have been a statistically significant result. These predictions where then applied to the overall seismic volume as shown in figure 5.5. Based on this information, the modeler can verify if the computer generated hypothesis is in accordance with other information available

5.6 Commonalities across application areas

The PMC based system has demonstrated its applicability over a large range of applications as described in the previous sections. The largest gains compared to alternative implementation lies in the high degree of acceleration of each individual chip, and the scalability obtained when using

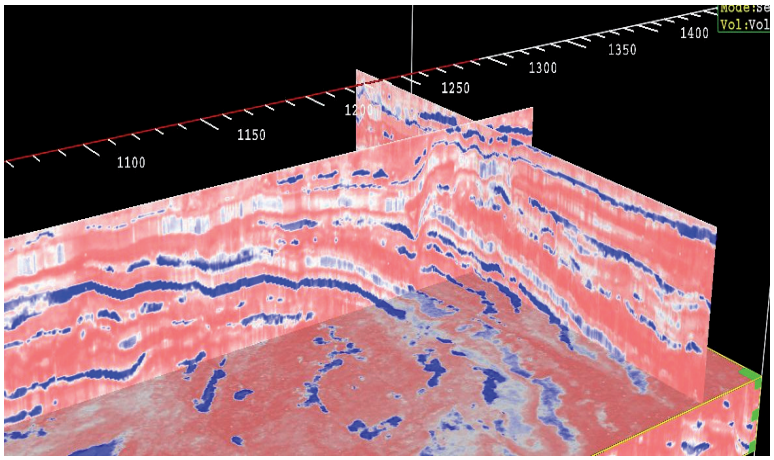

Figure 5.5: Predictions applied to seismic cube showing lateral conformation between the layers.

a collection of chips. As the PMC processing speed is dependent on the number of symbols within the query, and almost independent on the complexity when combining those symbols, short queries with numerous operations are beneficial.

The flexibility in query construction, and the low cost of evaluating such queries, has made the PMC a successful accelerator for genetic programming within tractable problem domains. Genetic programming can not be argued to be the best solution for all machine learning problems — probably it would be hard to give that argument for any machine learning technology — but it provides a processing requirement that is easily parallelized.

The applications have demonstrated linear scalability both for dynamic data or dynamic queries. It should be noted that if both data and queries were fully dynamic, there would not be enough bandwidth in the current setup to distribute the job. A scenario where both data and queries are static is an obvious simplification of any of the first two cases, and is easily handled as well.

Chapter 6

Evaluation of results

> *. . . thus each truth discovered was a rule available in the discovery of subsequent ones*
>
> René Descartes (1596–1650)

> *How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?*
>
> Homer Simpson

THROUGHOUT the research papers backing this thesis, the focus has been the *viability* of a MISD approach for pattern matching. Nevertheless, most of the work has been put into *implementing* the PMC and the surrounding systems. Any failure or success of the project does not solely rely on the theoretical properties for this architecture, but also the ability to build these into a real system. In this final chapter, the most important findings will be summarized. Aspects that could have been improved upon serves as directions for further work.

6.1 Summary of findings

6.1.1 Better usage of available memory bandwidth

As described in chapter 2, the widening gap between processing speed and memory bandwidth will prevent CPU technology from reaching its full performance by just increasing clock rates. Our approach — increasing the amount of processing done per data cycle — reduces the impact of the

performance gap. While I can not argue that our MISD architecture can solve any problem efficiently, this work has demonstrated its usefulness on a carefully selected, but still wide range of applications.

The same trend towards more parallel processing can be observed in commercial CPUs, where multi-core designs are becoming common across all high end product lines. These multiple cores are although competing for the same external chip bandwidth. To take advantage of the added processing capabilities, careful hardware-aware algorithm construction might play an even more important role than for previous CPU generations.

6.1.2 Scalable parallel architecture

Starting this project, there were many objections, both internally and externally, that Amdahl's law would prohibit a decent speedup when using several hundreds of thousand parallel processing elements. Amdahl's claim is not that massive parallelization is unfeasible, but rather that it will be restrained by serial dependencies. The obvious way around this, is to keep PMC usage within applications with a neglectable serial component.

Paper II has demonstrated linear scalability at 90 % of the theoretical capability, for a system with 81920 parallel processing elements. As explained in the paper, even higher utilization could have been achieved by fine tuning the system for the problem at hand. More importantly, we have observed a linear scalability, with no indication that the throughput per node would decrease for even larger systems.

6.1.3 Low cost — high performance

The main goal of this project was to build a low-cost alternative for processing non-numeric data. Although we have not yet reached a state with large commercial systems in operation, the findings so far indicate that the goal could be achieved.

The largest system in operation so far contained almost 500 PMC chips, with an accumulated $5 \cdot 10^{13}$ comparisons executed per second, across a data stream of 48 GB per second. Amortizing all costs for research, development and manufacturing into this single system, still keeps the price below five million US dollars. The system performance is equivalent to more than 10000 PCs. For such a PC cluster to be price competitive with the PMC, each node must cost less than 500 dollars, even if it was a one-off project.

Building the envisioned petacomp PMC cluster (10^{15} operations per second), will increase the spending, but the total cost would still be below ten million dollars. The alternative cost if built with standard PC technology would need to be less than 50 dollars per node to be competitive, a price point that that is currently unreachable. While high performance clusters like BlueGene (Adiga et al. 2002) have a more general architecture with wider applicability, the PMC system offers cost effective performance for an important subset of applications.

6.2 Propositions to the research questions

RQ1 *What are the system requirements for doing unstructured data processing efficiently?*

Unstructured data — as defined in this thesis — does not lend itself to preprocessing, for example into an index, without reducing the recall when querying the same data. Processing the data natively is most efficient under these conditions:

(i) The data storage need to provide high data bandwidth.

(ii) Linear access is most efficient for most storage technologies.

(iii) The processing circuitry should have the capability to do all required processing in a single pass of the data. The operations would be application dependent, but for the topics covered in this thesis the domination operations are symbol-to-symbol comparisons, and the combination of these into more complex queries.

RQ2 *What are the constraints for processing unstructured data with standard* CPU *technologies?*

A standard CPU can have high performance for algorithms where the flow of control is independent on the data values, as demonstrated for bit counting in section 2.3. Data dependent control flow would break two important technologies used to achieve high performance; SIMD processing and speculative out-of-order execution. This effect is evident even on rather simple problems like sorting (see section 2.2), and become a major bottleneck as data value dependencies grows larger, as discussed in Paper II, Paper IV and Paper V.

Secondly, sequential data processing can easily be parallelized across more processing units, as discussed in section 3.4. Reducing the in-

frastructure requirements of the processing unit from that of a ordinary CPU, would allow a higher degree of parallelism within a fixed volume and power budget.

RQ3 *Can other processing architectures than* SIMD *provide better performance for unstructured data processing?*

Parallel, data dependent control flow can be implemented in a MISD architecture. In the case of the PMC, the lack of data dependent execution eliminates the need for an instruction stream as such, as the operations for each query can be stored in configuration registers. Each query — the collection of configuration registers that defines the instructions to be applied to the data — can execute independent of the other queries, and without throttling the data bandwidth. Increasing complexity in the queries is managed by using more processing elements rather than compromising the throughput. A flexible processing element allocation scheme ensures that most units will have tasks assigned.

RQ4 *What are the obstacles for massively parallel systems in unstructured data processing?*

Section 4.3.1 described the instantaneous processing capacity p of the PMC system. Derived from this, the processing rate r is given as

$$r = f \cdot \sum q_i$$

there f is the operating frequency of the processing elements. Thus, in accordance with the discussion for RQ1, better performance can be achieved by increasing the rate of processing, or the number of parallel query evaluations.

Increasing f imposes an increase in power consumption with a factor of f^2. Increasing $\sum q_i$, the number of processing elements operating in parallel, will only give a linear increment in power consumption. The latter is thus more feasible from a scaling perspective, under the constraints of how many elements that can be packed onto a single die, and the ability to avoid serial dependencies that would prohibit speedup when employing more parallelism.

In practice, the frequency and degree of parallelism will be a compromise between these factors, where the PMC is one possible implementation.

RQ5 *What applications could benefit from unstructured data processing?*

Throughout this research, several application areas have emerged. Paper II through Paper VI describes the most analyzed examples. Chapter 5 introduces some further areas of applicability.

Overall, the benefits of the PMC in certain selected applications are substantiated. For some of these applications, the PMC might also be attractive from a commercial perspective.

6.3 Evaluation of the contributions

6.3.1 Summary of contributions

This thesis have six major contributions, enumerated C1 – C6, as described in section 1.5. C1 – C5 are the results of the research and analysis of MISD architecture, exemplified by the implementation of the PMC. C6 proves the validity of this approach by application examples.

6.3.2 Comparison with state-of-the-art

All the papers, except Paper VII, have a section describing detailed comparisons with state-of-the-art within the respective fields. From a top level perspective, the novelty of the proposed architecture have been examined thoroughly, especially in the process behind Patent I. This patent have been granted in all designated countries so far. The architecture could thus be considered a novel contribution to the current state-of-the-art.

Furthermore, Paper I through Paper VI, have demonstrated that this architecture has a performance at par with, or better than, other architectures for the selected problems. These comparisons have been on aspects such as pure performance, as well as performance relative to power consumption, system size and pricing.

6.3.3 Discussion

Although the PMC have demonstrated attractive capabilities, it is important to bear in mind that the targeted applications is only a niche area within data processing. Achieving better performance by constructing dedicated hardware for specific problems should not be a surprise to anyone.

Throughout the development of the PMC, the balancing of general capabilities versus performance in special cases have been under constant evaluation. In retrospect, I consider the FPGA prototyping as essential task, not only for getting an operative ASIC, but also for finding the best mix of features for the scenarios we had in mind during development.

That being said, the PMC architecture could still be tuned for even better performance in the demonstrated applications. Dependent on the transistor budget for a given chip, this could either be additions increasing the general applicability, or replacement of unneeded functions at the expense of further specialization.

Overall, the main contribution of this thesis is thus to prove the viability of MISD processing of unstructured data. With data collections currently having a growth ratio beyond the improvement in processing capabilities, new light might shine into the dark corner of Flynn's taxonomy.

6.4 Further work

Based on the work already completed, several extensions to the architecture could be investigated. The two main limitations prohibiting the PMC from tackling other attractive applications are discussed below.

6.4.1 Context save and restore

In the current implementation, data must arrive as a consecutive stream to be processed in context of the surrounding information. For static data, this is obviously not a problematic requirement. In some dynamic data scenarios, such as spam processing as described in Paper V, data sequencing can be reestablished before processing.

Other network applications that require massive pattern matching capability, especially in-stream processing like intrusion detection (Kuri et al. 2000), do not have this opportunity. Data in the same logical stream arrives in packets, multiplexed with other traffic. The packets within a single stream are not necessarily in chronological order. To some extent, this can be overcome by storing all ongoing traffic, and replaying the preceding packets for processing when a new packet arrives. Obviously, storing all packets of all ongoing streams soon becomes inviable.

One solution could be to add task switching capabilities to the PMC. This would require additional storage for inactive configurations, and the state of all intermediate registers in the processing tree. This could either

be on chip storage for a limited number of suspended tasks, or using the external memory at the expense of slower task switching.

The most flexible, yet high performance alternative, would be to have one set of shadow registers on chip. One register set could be dumped into and resorted from the external memory while the other is executing. As long as each task has a sufficient data payload, which anyhow is required for scalability, there would both be time and available memory bandwidth to make such a task switch in the background.

6.4.2 Spatial awareness beyond 1D

A single PMC is always looking at data as a one-dimensional stream. Although higher dimensionalities can be analyzed by deriving attributes into each data point as explained in chapter 5, the chip itself has no ability to know that distant data in a stream are adjacent — for example in a three dimensional interpretation of the same data — an important ability for image processing.

Finding a general solution to this problem might be hard. The effective usage of memory bandwidth by the PMC relies on linear access. Obtaining an arbitrary slice of a multidimensional space would require a fragmented access pattern in current storage media. Alternatively, several copies of the data could be stored, each optimized for access from different perspectives.

Glossary

IN this glossary most of the terms and acronyms used in this thesis are provided. Some of the entries have been adapted from dictionaries and glossaries from Merriam-Webster (Springfield, MA), the National Center for Biotechnology Information (Bethesda, MD), Microsoft (Redmond, WA), Invitrogen (Carlsbad, CA), the Technical University of Denmark (Kgs. Lyngby, Denmark), Monster Isp (Mount Vernon, OH), and the Free On-line Dictionary of Computing (http://foldoc.org/).

Arithmetic and logic unit (ALU). An arithmetic and logic unit is the core compute element within a CPU. An ALU typically takes two operands, and computes the result based on a selection of one out of several available functions.

Application-specific integrated circuit (ASIC). As it is designed for a very specific purpose, ASICs contrast with more general-purpose devices such as memory chips or x86 processors that can be used in many different applications. ASICs are used in a number of specific applications, such as processors for controlling engines or chips on a motherboard chip-sets. When produced in high volumes, ASICs have orders of magnitude higher cost-performance ratio than field-programmable gate arrays (FPGAs).

Basic local alignment tool (BLAST). A popular sequence comparison algorithm that is used to search for optimal local alignments between a sequence database and a pattern query. The BLAST algorithm is optimized for speed, and the initial seed search is done for a word of a specific length that scores at least some threshold when compared to the query using a substitution matrix. Word hits are then extended in either direction in an attempt to generate an alignment with optimal score. Note that when the text consists of nucleotides, practical implementations will require a perfect seed match between the word and the database.

Central processing unit (CPU). The main processing unit performing the general digital operations in a computer. The CPU is designed to run a group of instructions, or instruction set. CPU instructions can consist of adding and subtracting numbers, fetching information from memory, and other simple functions.

Deoxyribonucleic acid (DNA). Any of various nucleic acids that are usually the molecular basis of heredity and localized especially in the cell's nucleus. DNA consists of two chains of alternate links between deoxyribose and phosphate that are held together by hydrogen bonds in a double helix configuration.

Deterministic finite automata (DFA). A deterministic finite automata is a state machine where the next state is uniquely determined by a single input event.

Dynamic random access memory (DRAM). A form of semiconductor random access memory (RAM). Dynamic RAMs store information in integrated circuits containing capacitors. Because capacitors lose their charge over time, dynamic RAM boards must include logic to refresh (recharge) the RAM chips continuously. While a dynamic RAM is being refreshed, it cannot be read by the processor; if the processor must read the RAM while it is being refreshed, one or more wait states occur. Despite being slower, dynamic RAMs are more commonly used than static RAMs because their circuitry is simpler.

Floating point operations per second (FLOPS). FLOPS is an acronym used for denoting computer performance in scientific calculations, meaning floating point operations per second according to a benchmark, e.g. Linpack. One such operation could be the addition of two numbers.

Field-programmable gate array (FPGA). A microchip that may contain thousands of programmable logic gates. Good features of FPGAs include short development times, and FPGAs are often used for prototype or custom designs, including for example logic emulation. Applications that require high-volume production usually use application-specific integrated circuits (ASICs) instead.

Genetic programming (GP). A problem-solving algorithm that uses mutation and recombination to breed generations of computer programs that is intended to solve a certain problem. Compared with genetic

algorithms that operate directly on bit strings, GP operates on computer programs with a predefined architecture. In theory, the best computer programs improve with each generation, and the final solution approach the optimal solution.

Gigabytes (GB). A byte is a group of eight binary digits called bits, and a gigabyte is by definition 2^{30} or 1,073,741,824 bytes, as this is the power of 2 that is closest to one billion. In some cases, especially for hard drive storage systems, one gigabyte is also used for exactly one billion bytes.

Interagon query language (IQL). A simple expression language that defines how queries are constructed using characters, strings, and string set operators. The IQL is the preferred query language for an application accessing the pattern matching chip (PMC), as the language's expressiveness corresponds closely to the available functionality of the chip architecture.

Megabytes (MB). A byte is a group of eight binary digits called bits, and a megabyte is by definition 2^{20} or 1,048,576 bytes, as this is the power of 2 that is closest to one million.

Messenger ribonucleic acid (mRNA). A ribonucleic acid (RNA) that carries the code for a particular protein from the nuclear deoxyribonucleic acid (DNA) to a ribosome in the cytoplasm and acts as a template for the formation of that protein.

Multiple instruction stream - multiple data stream (MIMD). One of four categories in Flynn's taxonomy for classification of architectures along two axes, namely the number of instruction streams executing concurrently, and the number of data sets to which those instructions are being applied. A MIMD architecture is one where many instructions are concurrently applied to multiple data sets.

Multiple instruction stream - single data stream (MISD). One of four categories of Flynn's taxonomy for classification of architectures along two axes, namely the number of instruction streams executing concurrently, and the number of data sets to which those instructions are being applied. A MISD architecture is one where many instructions are concurrently applied to a single data set.

Nondeterministic finite automata (NFA). The next state of a nondeterministic finite automata depends not only on the current input event, but

also on an arbitrary number of subsequent input events. Until these subsequent events occur it is not possible to determine which state the machine is in.

Random access memory (RAM). Acronym for random access memory for information storage. Semiconductor-based memory that can be read and written by the central processing unit (CPU) or other hardware devices. The storage locations can be accessed in any order.

Peripheral component interconnect (PCI). A specification for high-performance, 32-bit or 64-bit input/output (I/O) buses. A PCI bus can be configured dynamically and is designed to be used by devices with high-bandwidth requirements.

Reduced Instruction Set Computing (RISC). A microprocessor design that focuses on rapid and efficient processing of a relatively small set of simple instructions that comprises most of the instructions a computer decodes and executes. RISC architecture optimizes each of these instructions so that it can be carried out very rapidly–usually within a single clock cycle. RISC chips thus execute simple instructions more quickly than general-purpose CISC (complex instruction set computing) microprocessors, which are designed to handle a much wider array of instructions. They are, however, slower than CISC chips at executing complex instructions, which must be broken down into many machine instructions that RISC microprocessors can perform.

Ribonucleic acid (RNA). Any of various nucleic acids that contain ribose and uracil as structural components, and are associated with the control of cellular chemical activities

Synchronous DRAM (SDRAM). A variant of DRAM where all access is synchronized to a clock signal.

Short interfering ribonucleic acid (siRNA). A short double-stranded ribonucleic acid of about 21 nucleotides with 3′ overhangs of two nucleotides that mediates the ribonucleic acid interference response in mammalian cells.

Single instruction stream - multiple data stream (SIMD). One of four categories in Flynn's taxonomy for classification of architectures along two axes, namely the number of instruction streams executing concurrently, and the number of data sets to which those instructions are

being applied. A SIMD architecture is one where a single instruction is concurrently applied to multiple data sets.

Single instruction stream - single data stream (SISD). One of four categories of Flynn's taxonomy for classification of architectures along two axes, namely the number of instruction streams executing concurrently, and the number of data sets to which those instructions are being applied. A SISD architecture is one where a single instruction is applied to a single data set.

Support vector machine (SVM). A generalized linear classifier that corresponds to the optimal classifier as defined by some maximum-margin criterion. The maximum-margin criterion provides regularization that helps the classifier to generalize better to unseen samples.

Very-large-scale integration (VLSI). VLSI originally referred to chips with many tens of thousands transistors, as a natural successor to large-scale integration (LSI) chips that contain more than thousand transistors. There have been efforts to name various levels of integrations above VLSI, but these are no longer in widespread use. Note that all microprocessors are VLSI or better.

Bibliography

N. R. Adiga, G. Almasi, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, A. A. Bright, J. Brunheroto, C. Cacaval, J. Castaños, W. Chan, L. Ceze, P. Coteus, S. Chatterjee, D. Chen, G. Chiu, T. M. Cipolla, P. Crumley, K. M. Desai, A. Deutsch, T. Domany, M. B. Dombrowa, W. Donath, M. Eleftheriou, C. Erway, J. Esch, B. Fitch, J. Gagliano, A. Gara, R. Garg, R. Germain, M. E. Giampapa, B. Gopalsamy, J. Gunnels, M. Gupta, F. Gustavson, S. Hall, R. A. Haring, D. Heidel, P. Heidelberger, L. M. Herger, D. Hoenicke, R. D. Jackson, T. Jamal-Eddine, G. V. Kopcsay, E. Krevat, M. P. Kurhekar, A. P. Lanzetta, D. Lieber, L. K. Liu, M. Lu, M. Mendell, A. Misra, Y. Moatti, L. Mok, J. E. Moreira, B. J. Nathanson, M. Newton, M. Ohmacht, A. Oliner, V. Pandit, R. B. Pudota, R. Rand, R. Regan, B. Rubin, A. Ruehli, S. Rus, R. K. Sahoo, A. Sanomiya, E. Schenfeld, M. Sharma, E. Shmueli, S. Singh, P. Song, V. Srinivasan, B. D. Steinmacher-Burow, K. Strauss, C. Surovic, R. Swetz, T. Takken, R. B. Tremaine, M. Tsao, A. R. Umamaheshwaran, P. Verma, P. Vranas, T. J. C. Ward, M. Wazlowski, W. Barrett, C. Engel, B. Drehmel, B. Hilgart, D. Hill, F. Kasemkhani, D. Krolak, C. T. Li, T. Liebsch, J. Marcella, A. Muff, A. Okomo, M. Rouse, A. Schram, M. Tubbs, G. Ulsh, C. Wait, J. Wittrup, M. Bae, K. Dockser, L. Kissel, M. K. Seager, J. S. Vetter, and K. Yates. An overview of the BlueGene/L supercomputer. In IEEE, editor, *SC2002: From Terabytes to Insight. Proceedings of the IEEE ACM SC 2002 Conference*, 2002. ISBN 0-7695-1524-X. URL http://www.sc-2002.org/paperpdfs/pap.pap207.pdf.

Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry, Proceedings of the 1996 AMS-IMS-SIAM*, volume 223 of *Contemporary Mathematics*, pages 1–56. Americal Mathematical Society, 1996. ISBN 0-8218-0674-2.

Alfred V. Aho, John E. Hopcroft, and Jeffery D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1982. ISBN 0-201-00023-7.

Nasir Al-Darwish. Formulation and analysis of in-place MSD radix sort algorithms. *Journal of Information Science*, 31(6):467–481, 2005.

Kursad Albayraktaroglu, Aamer Jaleel, Xue Wu, Manoj Franklin, Bruce Jacob, Chau-Wen Tseng, and Donald Yeung. Biobench: A benchmark suite of bioinformatics applications. In *2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–9, March 2005.

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215 (3):403–410, 1990.

Mohammed Amarzguioui, Torgeir Holen, Eshrat Babaie, and Hans Prydz. Tolerance for mutations and chemical modifications in a siRNA. *Nucleic Acids Res.*, 31(2):589–595, 2003.

Gene Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

Will Archer Arentz, Magnus Lie Hetland, and Bjørn Olstad. Retrieving musical information based on rhythm and pitch correlations. *Journal of New Music Research*, 34(2):151–159, 2005.

Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

Kent Beck. *Extreme Programming Explained*. Addison-Wesley Pub Co, Boston, MA 02116, USA, 2001.

Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, 35(12):29–38, 1992. ISSN 0001-0782.

Jon L. Bentley and Robert Sedgewick. Fast algorithms for sorting and searching strings. In *SODA'97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 360–369, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics. ISBN 0-89871-390-0.

Olaf René Birkeland and Ola Snøve Jr. The pattern matching chip, 2002. Technical note, available upon request.

Olaf René Birkeland, Magnar Nedland, and Ola Snøve Jr. Massively parallel MIMD system achieves high performance in a spam filter. In Gerhard R. Joubert, Wolfgang E. Nagel, Frans J. Peters, Oscar Plata, Paco Tirado, and Emilio Zapata, editors, *Parallel Computing: Current & Future Issues of High-End Computing. Proceedings of the International Conference ParCo 2005*, volume 33 of *John von Neumann Institute for Computing (NIC)*, pages 691–697, September 2005. ISBN 3-00-017352-8.

Olaf René Birkeland, Ola Snøve Jr., Arne Halaas, Magnar Nedland, and Pål Sætrom. The petacomp machine — a MIMD cluster for parallel pattern-mining. In *2006 IEEE International Conference on Cluster Computing (CLUSTER)*, September 2006. ISBN 1-4244-0328-6. Released on CDROM, IEEE catalog number 06TH8880C.

Dina Bitton, David J. DeWitt, David K. Hsaio, and Jaishankar Menon. A taxonomy of parallel sorting. *ACM Computer Surveys (CSUR)*, 16(3):287–318, 1984. ISSN 0360-0300.

Hans-Martin Blüthgen and Tobias G. Noll. A programmable processor for approximate string matching with high throughput rate. In *ASAP'00: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 309–316, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0716-6.

Darrell Boggs, Aravindh Baktha, Jason Hawkins, Deborah T. Marr, J. Alan Miller, Patrice Roussel, Ronak Singhal, Bret Toll, and K.S. Venkatraman. The microarchitecture of the Intel Pentium 4 processor on 90nm technology. *Intel Technology Journal*, 08(01), 2004. URL http://developer.intel.com/technology/itj/index.htm.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

Doug Burger, James R. Goodman, and Alain Kägi. Memory bandwidth limitations of future microprocessors. In *23rd International Symposium on Computer Architecture (ISCA)*, pages 78–89. IEEE Computer Society, May 1996.

Alan Chalmers and Jonathan Tidmus. *Practical Parallel Processing*. International Thompson Computer Press, 1996.

Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related prob-

lems. In *Automata, Languages and Programming, 29th International Colloquium, ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2002. ISBN 3-540-43864-5.

H. D. Cheng and K. S. Fu. VLSI architectures for string matching and pattern matching. *Pattern Recogn.*, 20(1):125–144, 1987. ISSN 0031-3203.

David M. Dahle, Jeffrey D. Hirschberg, Kevin Karplus, Hansjoerg Keller, Eric Rice, Don Speck, Douglas H. Williams, and Richard Hughey. Kestrel: Design of an 8-bit SIMD parallel processor. In *17th Conference on Advanced Research in VLSI*, 1997.

Derek M. Dykxhoorn, Carl D. Novina, and Phillip A. Sharp. Killing the messenger: short RNAs that silence gene expression. *Nat. Rev. Mol. Cell Biol.*, 4(6):457–467, 2003.

Alexandre E. Eichenberger, Kathryn O'Brien, Kevin O'Brien, Peng Wu, Tong Chen, Peter H. Oden, Daniel A. Prener, Janice C. Shepherd, Byoungro So, Zehra Sura, Amy Wang, Tao Zhang, Peng Zhao, and Michael Gschwind. Optimizing compiler for a CELL processor. In *14th International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 161–172. IEEE Computer Society, 2005. ISBN 0-7695-2429-X.

Vladmir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computer Surveys (CSUR)*, 24(4):441–476, 1992. ISSN 0360-0300.

Michael Farrar. Striped smith-waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23(2):156–161, 2007.

Fast Search & Transfer ASA. A processing circuit and a search processor circuit, 2000a. International publication number WO 00/29981.

Fast Search & Transfer ASA. Digital processing device, 2000b. International publication number WO 00/22545.

Wu-chun Feng. Green destiny + mpiblast = bioinfomagic. In *ParCo 2003*, pages 653–660, 2003.

Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C(21):948–960, September 1972.

M. J. Foster and H. T. Kung. Design of special-purpose VLSI chips: Example and opinions. In *ISCA'80: Proceedings of the 7th annual symposium on Computer Architecture*, pages 300–307, New York, NY, USA, 1980. ACM Press.

Jeffrey E.F. Friedl. *Mastering Regular Expressions*. O'Reilly, Cambridge, MA, 2nd edition, 2002.

Joseph Gebis, Sam Williams, Christos Kozyrakis, and David Patterson. VIRAM1: A media-oriented vector processor with embedded DRAM. In *41st Design Automation Student Design Contenst, San Diego, CA*, June 2004.

Richard Gerber, Aart J.C. Bik, Kevin B. Smith, and Xinmin Tian. *The Software Optimization Cookbook. High-Performance Recipes for the Intel Architecture*. Intel Press, 2nd edition, March 2006. ISBN 0-9764832-1-1.

Arne Halaas. A systolic VLSI matrix for a family of fundamental searching problems. *Integration, the VLSI Journal*, 1(3), 1983.

Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve Jr., and Olaf René Birkeland. A recursive MISD architecture for pattern matching. *IEEE Trans. on VLSI Syst.*, 12(7):727–734, 2004.

Gregory J. Hannon. RNA interference. *Nature*, 418(6894):244–251, 2002.

Torstein Heggebø. *Compaction of Symbolic Layout. Generation of Compact and Correct Mask Layout from Symbolic Descriptions of VLSI Circuits*. PhD thesis, Norwegian University of Science and Technology, 1989.

John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 4th edition, 2007. ISBN 978-0-12-370490-0.

Magnus Lie Hetland and Pål Sætrom. Temporal rule discovery using genetic programming and specialized hardware. In *Proc. of the 4th Int. Conf. on Recent Advances in Soft Computing*, 2002.

Magnus Lie Hetland and Pål Sætrom. A comparison of hardware and software in sequence rule evolution. In *Eight Scandinavian Conference on Artificial Intelligence*, 2003.

Magnus Lie Hetland and Pål Sætrom. Temporal rule discovery using genetic programming and specialized hardware. In Ahmad Lotfi and

Jonathon M. Garibaldi, editors, *Applications and Science in Soft Computing*, Advances in Soft Computing, pages 87–94. Springer-Verlag, 2004. Revised version of Hetland and Sætrom (2002).

Paul Hildebrandt and Harold Isbitz. Radix exchange — an internal sorting method for digital computers. *Journal of the ACM*, 6(2):156–163, 1959.

W. Daniel Hillis and Jr. Guy L. Steele. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, 1986. ISSN 0001-0782.

Masaki Hirata, Hachiro Yamada, Hajime Nagai, and Kousuke Takahashi. A versatile data string-search VLSI. *IEEE Journal of Solid-State Circuits*, 23(2):329–335, 1988.

Jeffrey D. Hirschberg, David M. Dahle, Kevin Karplus, Don Speck, and Richard Hughey. Kestrel: A programmable array for sequence analysis. *Journal of VLSI Signal Processing Systems for Signal Image and Video Technology*, 19(2):115–126, 1998.

Richard Hughey. Parallel hardware for sequence comparison and alignment. *CABIOS*, 12(6):473–479, 1996.

Kai Hwang and Fayé A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Book Company, 1985. page 32–35.

Intel Corporation. IA-32 Intel® architecture optimization reference manual. Technical report, Intel Corporation, 2004. This manual is available from http://developer.intel.com/design/Pentium4/documentation.htm.

Merrill E. Isenman and Dennis E. Shasha. Performance and architectural issues for string matching. *IEEE Transactions on Computers*, 39(2):238–250, 1990. ISSN 0018-9340.

James A. Kahle, Michael N. Day, H. Peter Hofstee, Charles R. Johns, Theodore R. Maeurer, and David Shippy. Introduction to the Cell multiprocessor. *IBM Journal on Research & Development*, 49(4/5):589–604, July/September 2005.

Graham Kirsch. Active memory: Micron's Yukon. In *17th International Parallel and Distributed Processing Symposium (IPDPS)*, page 89. IEEE Computer Society, 2003.

Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

Donald E. Knuth. *The Art of Computer Programming, vol 1–3*. Addison Wesley, 3rd edition, 1997.

Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, March/April 2005.

John R. Koza, David Andre, Forrest H Bennett III, and Martin Keane. *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Fransisco, CA, Apr 1999.

Christos Kozyrakis and David Patterson. Overcoming the limitations of conventional vector processors. In *30th International Symposium on Computer Architecture (ISCA)*, pages 399–409. IEEE Computer Society, Jun 2003. ISBN 0-7695-1945-8.

R. K. Krishnarnurthy, A. Alvandpour, V. De, and S. Borkar. High-performance and low-power challenges for sub-70 nm microprocessor circuits. In *Proceedings of the IEEE 2002 Custom Integrated Circuits Conference*, pages 125–128, 2002. ISBN 0-7803-7250-6.

R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In *36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, pages 81–92, December 2003. ISBN 0-7695-2043-X.

Josué Kuri, Gonzalo Navarro, Ludovic Mé, and Laurent Heye. A pattern matching based filter for audit reduction and fast detection of potential intrusions. In *In proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection*, pages 17–27. LNCS, 2000.

Clecio Donizete Lima and Tadao Nakamura. Exploiting loop-level parallelism with the shift architecture. In *Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)*, pages 184–194, 2002. ISBN 0-7695-1772-2.

Clecio Donizete Lima, Kentaro Sano, Hiroaki Kobayashi, Tadao Nakamura, and Michael J. Flynn. A technology-scalable multithreaded architecture. In *Proceedings of the 13th Symposium on Computer Architecture and High Performance Computing*, pages 82–89, September 2001.

Yen-Chun Lin and Jih-Wei Yeh. A scalable and efficient systolic algorithm for the longest common subsequence problem. *J. Inf. Sci. Eng.*, 18(4): 519–532, 2002.

Jim W. Lindelien. The value of accelerated computing in bioinformatics. See http://www.timelogic.com/whitepapers/decypher_benefits_e.pdf, 2002.

Victor Wing-Kit Mak, Kuo Chu Lee, and Ophir Frieder. Exploiting parallelism in pattern matching: an information retrieval application. *ACM Trans. Inf. Syst.*, 9(1):52–74, 1991. ISSN 1046-8188.

Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 06(01):4–15, 2002. URL http://developer.intel.com/technology/itj/index.htm.

Arne Maus. ARL, a faster in-place, cache friendly sorting algorithm. In *NIK'2002: Norsk informatikkonferanse*, pages 85–95. Society for Industrial and Applied Mathematics, 2002. URL http://www.nik.no/2002/Maus.pdf.

Peter M. McIlroy, Keith Bostic, and M. Douglas McIlroy. Engineering radix sort. *Computing Systems*, 6(1):5–27, 1993.

Michael T. McManus and Phillip A. Sharp. Gene silencing in mammals by small interfering RNAs. *Nat. Rev. Genet.*, 3(10):737–747, 2002.

O.M. Melko and A.R. Mushegian. Distribution of words with a predefined range of mismatches to a DNA probe in bacterial genomes. *Bioinformatics*, 20(1):67–74, 2004.

Panagiotis D. Michailidis and Konstantinos G. Margaritis. On-line approximate string searching algorithms: Survey and experimental results. *International Journal of Computer Mathematics*, 79(8):867–888, 2002.

Panagiotis D. Michailidis and Konstantinos G. Margaritis. Performance evaluation of load balancing strategies for approximate string matching application on an mpi cluster of heterogeneous workstations. *Future Generation Computer Systems*, 19(7):1075–1104, 2003. ISSN 0167-739X.

Panagiotis D. Michailidis and Konstantinos G. Margaritis. New processor array architectures for the longest common subsequence problem. *The Journal of Supercomputing*, 32(1):51–69, April 2005.

Donald Michie. Memo functions and machine learning. *Nature*, 218:19–22, 1968.

K. Mistry, C. Allen, C. Auth, B. Beattie, D. Bergstrom, M. Bost, M. Brazier, M. Buehler, A. Cappellani, R. Chau, C.-H. Choi, G. Ding, K. Fischer, T. Ghani, R. Grover, W. Han, D. Hanken, M. Hattendorf, J. He, J. Hicks, R. Huessner, D. Ingerly, P. Jain, R. James, L. Jong, S. Joshi, C. Kenyon, K. Kuhn, K. Lee, H. Liu, J. Maiz, B. McIntyre, P. Moon, J. Neirynck, S. Pae, C. Parker, D. Parsons, C. Prasad, L. Pipes, M. Prince, P. Ranade, T. Reynolds, J. Sandford, L. Shifren, J. Sebastian, J. Seiple, D. Simon, S. Sivakumar, P. Smith, C. Thomas, T. Troeger, P. Vandervoorn, S. Williams, and K. Zawadzki. A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and 100% pb-free packaging. In *Electron Devices Meeting (IEDM), IEEE International*, pages 247–250, December 2007. ISBN 978-1-4244-1508-3.

Amar Mukherjee. Hardware algorithms for determining similarity between two strings. *IEEE Transactions on Computers*, 38(4):600–603, 1989.

Tadao Nakamura, Masatsugu Hashimoto, Geng Chun, and Norio Izumi. Visual architecture. *INFORMATION*, 6(2):215–230, 2003. ISSN 1343-4500.

Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001a.

Gonzalo Navarro. NR-grep: a fast and flexible pattern matching tool. *Software Practice and Experience (SPE)*, 31:1265–1312, 2001b.

Gonzalo Navarro and Mathieu Raffinot. *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, Cambridge, UK, 2002.

Magnar Nedland, Børge Svingen, and Magnus L. Hetland. The Interagon Query Language - a reference guide. Interagon AS (Trondheim, Norway) whitepaper, see http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf, 2002a.

Magnar Nedland, Børge Svingen, and Magnus L. Hetland. The Interagon Query Language – a reference guide, 2002b. Available on request: info@interagon.com.

Jin Hwan Park and K. M. George. Efficient parallel hardware algorithms for string matching. *Microprocessors and microsystems*, 23(3):155–168, 1999.

David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent RAM: IRAM. *IEEE Micro*, 17(2):34–44, Apr 1997.

David A. Patterson. Latency lags bandwith. *Communications of the ACM*, 47(10):71–75, 2004. ISSN 0001-0782.

Jeremy Pickens. Feature selection for polyphonic music retrieval. In *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 428–429, 2001.

Knut Magne Risvik. *Scaling Internet Search Engines: Methods and Analysis*. PhD thesis, NTNU, Trondheim, Norway, May 2004.

Knut Magne Risvik and Rolf Michelsen. Search engines and web dynamics. *Computer Networks*, 39(3):289–302, 2002.

Ian N. Robinson. Pattern-addressable memory. *IEEE Micro*, 12(3):20–30, 1992. ISSN 0272-1732.

Torbjørn Rognes. ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Res.*, 29(7):1647–1652, 2001.

Torbjørn Rognes and Erling Seeberg. Six-fold speed-up of Smith-Waterman sequence database searching using parallel processing on common microprocessors. *Bioinformatics*, 16(8):4926–4936, 2000.

Halsey L. Royden. *Real Analysis*. Macmillan, 3rd edition, 1988. ISBN 0024041513.

Ola Sætrom, Ola Snøve Jr., and Pål Sætrom. Weighted sequence motifs as an improved seeding step in microRNA target prediction algorithms. *RNA*, 11(7):995–1003, 2005a.

Pål Sætrom. Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063, 2004.

Pål Sætrom. *Hardware accelerated genetic programming for pattern mining in strings*. PhD thesis, Norwegian University of Science and Technology, 2005.

Pål Sætrom and Magnus Lie Hetland. Multiobjective evolution of temporal rules. In *Eight Scandinavian Conference on Artificial Intelligence*, 2003a.

Pål Sætrom and Magnus Lie Hetland. Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'03)*, pages 145–151, 2003b.

Pål Sætrom and Ola Snøve Jr. A comparison of siRNA efficacy predictors. *Biochem. Biophys. Res. Commun.*, 321(1):247–253, 2004.

Pål Sætrom, Ragnhild Sneve, Knut I. Kristiansen, Ola Snøve Jr., Thomas Grünfeld, Torbjørn Rognes, and Erling Seeberg. Predicting non-coding RNA genes in *Escherichia coli* with boosted genetic programming. *Nucleic Acids Res.*, 33(10):3263–3270, 2005b.

Pål Sætrom, Olaf Birkeland, and Ola Snøve Jr. *Genetic Programming Theory and Practice IV*, chapter Boosting improves stability and accuracy of genetic programming in biological sequence classification. Springer-Verlag, 2007. ISBN 0-387-33375-4.

Geir Kjetil Sandve, Magnar Nedland, Øyvind Bø Syrstad, Lars Andreas, Eidsheim, Osman Abul, and Finn Drabløs. Accelerating motif discovery: Motif matching on parallel hardware. In *6th Workshop on Algorithms in Bioinformatics (WABI)*, 2006.

David Sankoff and Joseph Kruskal, editors. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley, Cambridge, MA, 1983.

Raghu Sastry, N. Ranganathan, and Klinton Remedios. CASM: A VLSI chip for approximate string matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):824–830, 1995. ISSN 0162-8828.

Bertil Schmidt, Heiko Schröder, and Manfred Schimmler. Protein sequence comparison on the instruction systolic array. In *Parallel Computing Technologies, 6th International Conference, PaCT, Novosibirsk, Russia*, pages 498–509, 2001.

Bengt-Olaf Schneider and Jarek Rossignac. M-buffer: A flexible misd architecture for advanced graphics. *Computers & Graphics*, 19(2):239–246, March–April 1995.

Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 2005. ISBN 0619217642.

Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147(1):195–197, 1981.

Ola Snøve Jr. *Analysis of RNAi intermediates using the pattern matching chip*. PhD thesis, Norwegian University of Science and Technology, 2005.

Ola Snøve Jr. and Torgeir Holen. Many commonly used siRNAs risk off-target activity. *Biochem. Biophys. Res. Commun.*, 319(1):256–263, 2004.

Ola Snøve Jr., Magnar Nedland, Ståle H. Fjeldstad, Håkon Humberset, Olaf René Birkeland, Thomas Grünfeld, and Pål Sætrom. Designing effective siRNAs with off-target control. *Biochem. Biophys. Res. Commun.*, 325(3):769–773, 2004.

Ola Snøve Jr., Håkon Humberset, Olaf René Birkeland, and Pål Sætrom. Sequence Explorer: interactive exploration of genomic sequence data, 2005. Manuscript.

Xinmin Tian, Milind Girkar, Aart Bik, and Hideki Saito. Practical compiler techniques on efficient multithreaded code generation for OpenMP programs. *Computer Journal*, 48(5):588–601, 2005.

Panagiotis G. Tzionas, Philippos G. Tsalides, and Adonios Thanailakis. A new, cellular automaton-based, nearest neighbor pattern classifier and its VLSI implementation. *IEEE Transactions on VLSI systems*, 2(3):343–353, September 1994.

Aad J. van der Steen and Jack J. Dongarra. Overview of recent supercomputers. Technical report, EuroBen, October 2004. This 14th issue of the annual report is available from http://www.top500.org/ORSC/2004/.

Jan van Lunteren, Ton Engbersen, Joe Bostian, Bill Carey, and Chris Larsson. XML accelerator engine. In *First International Workshop on High Performance XML Processing*, May 2004.

David W. Wall. Limits of instruction-level parallelism. In *ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 176–188, New York, NY, USA, 1991. ACM Press. ISBN 0-89791-380-9.

Maurice V. Wilkes. The memory gap and the future of high performance memories. *SIGARCH Computer Architecture News*, 29(1):2–7, 2001. ISSN 0163-5964.

Dan E. Willard. Applications of range query theory to relational data base join and selection operations. *Journal on Computer and System Sciences*, 52 (1):157–169, 1996.

Hachiro Yamada, Masaki Hirata, Hajime Nagai, and Kousuke Takahashi. A high-speed string-search engine. *IEEE Journal of Solid-State Circuits*, 22 (5):829–834, 1987.

Tomoyuki Yamada and Shinichi Morishita. Accelerated off-target search algorithm for siRNA. *Bioinformatics*, 21(8):1316–1324, 2005.

Kwang-I Yu, Shi-Ping Hsu, and Peggy Otsubo. The fast data finder - an architecture for very high speed data search and dissemination. In *Proceedings of the First International Conference on Data Engineering, April 24-27, 1984, Los Angeles, California, USA*, pages 167–174. IEEE Computer Society, 1984. ISBN 0-8186-0533-2.

Phillip D. Zamore. RNA interference: listening to the sound of silence. *Nat. Struct. Mol. Biol.*, 8(9):746–750, 2001.

Papers

Paper I

A recursive MISD architecture for pattern matching

A Recursive MISD Architecture for Pattern Matching

Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve, Jr., and Olaf René Birkeland

***Abstract*—Many applications require searching for multiple patterns in large data streams for which there is no preprocessed index to rely on for efficient lookups. An multiple instruction stream–single data stream (MISD) VLSI architecture that is based on a recursive divide and conquer approach to pattern matching is proposed. This architecture allows searching for multiple patterns simultaneously. The patterns can be constructed much like regular expressions, and add features such as requiring subpatterns to match in a specific order with some fuzzy distance between them, and the ability to allow errors according to prescribed thresholds, or ranges of such. The current implementation permits up to 127 simultaneous patterns at a clock frequency of 100 MHz, and does 1.024×10^{11} character comparisons per second.**

***Index Terms*—Approximate search, multiple instruction stream–single data stream (MISD), online pattern matching, parallel architecture, VLSI.**

I. Introduction

ONLINE multipattern approximate searching applies to situations where several patterns are to be matched concurrently in data where no persistent index can be built to accommodate efficient lookups. Few algorithms exist for searching multiple complex approximate patterns simultaneously, and most of them are filters that lose efficiency for increasing error levels [1].

Consider a string, $S = s_1 s_2, \ldots, s_m$, where s_i is a character from a finite alphabet Σ of size $|\Sigma| = \sigma$. The problem is to simultaneously search S for the occurrence of a set of predefined patterns, $P = \{p_1, p_2, \ldots, p_n\}$, of varying complexity. The patterns p_j considered in this work are constructed much like regular expressions [2], although with some differences. Notably, these include Hamming distance [3] operations on substrings and arbitrary complex boolean functions on subpatterns. Furthermore, hit lingering is allowed to specify coarser approximations, i.e., "match several patterns if they occur within some distance of each other." Order conditions combined with latency is also an additional feature compared with regular expressions; that is, the user may specify that several patterns are to match in an ordered manner and with a (possibly unspecified) distance separating them. The architecture does not yet support the full language of regular expressions; there are limitations when it comes to nested repeating patterns, that is, repeated patterns that consist of subpatterns that are themselves repeating.

Manuscript received April 11, 2003; revised November 6, 2003.

A. Halaas and B. Svingen are with the Department of Computer and Information Science, Norwegian University of Science and Technology, No-7491 Trondheim, Norway (e-mail: Arne.Halaas@idi.ntnu.no).

M. Nedland, P. Sætrom, O. Snøve, Jr., and O. R. Birkeland, are with the Interagon AS, Medisinsk Teknisk Senter, NO-7489 Trondheim, Norway (e-mail: Olaf.Birkeland@interagon.com).

Digital Object Identifier 10.1109/TVLSI.2004.830918

The VLSI implementation uses 0.20-μm CMOS technology that yields a 114 mm^2 die size, and a total of 11.7 million transistors. The architecture's design clock frequency is 100 MHz. Current applications for the Peripheral Component Interconnect (PCI) card on which the chips are mounted include, but are not limited to, optimal selection of reverse transcriptase polymerase chain reaction (RT-PCR) primers [4], DNA sequence assembly [5], and building libraries of siRNA oligonucleotides for mRNA knock-down experiments [6]. Furthermore, the chip has been used to enhance the practicability of genetic programming in pattern mining [7], [8]: the programs that evolve are patterns that can be evaluated in hardware, thus allowing larger datasets, bigger populations, and longer runs than would otherwise be possible.

A brief review of online multipattern approximate searching, as well as previous attempts to implement flexible pattern matching in hardware, is given in Section II. The required functionality is formally described in Section III, while Section IV presents the proposed MISD architecture. The motivation behind this architecture comes from the recursive definitions in Section III. Section V outlines practical design considerations that have been made, and Section VI compares the proposed architecture to other architectures. Finally, in Section VII, we discuss the architecture and makes suggestions for future work.

II. Previous Work

Several excellent surveys exist on the subject of flexible pattern matching in strings [1], [9]–[11]. Among numerous applications are filters for intrusion detection [12], information retrieval [13], and sequence similarity in biology [14]. Multiple approximate pattern matching is used in machine learning, where the concepts of ensembles [15], and mixtures of experts [16] use collections of patterns to increase the performance of the overall model.

Several hardware approaches for exact and approximate string matching have been proposed [17]–[28]. In [10], parallel comparators, associative memories, cellular arrays, and finite-state automata are listed as the architectures that have been used in hardware string matching solutions. Recently, a lot of attention has been given to the problem of edit distance computing and matching [29]–[34] in the context of sequence retrieval and matching from DNA and protein sequence databases. In [35], a review of relevant parallel hardware for sequence comparison and alignment is given. Note, however, that benchmarking is a difficult task due to significant algorithm discrepancies between the systems.

III. Functionality

The architecture design aims to support a set of query semantics:

1) concatenation of basic strings and patterns;
2) alphanumerical comparisons of simple strings;
3) boolean operations on subexpressions;
4) hamming distance filtering;
5) hit lingering (latency);
6) regular expressions.

A formal query language [36], [37] has been constructed to support the given functionality (details omitted). The outlined query modes will be described in the following.

Def. 1: A string $S = s_1 s_2, \ldots, s_m$ is an ordered set of m characters from a finite alphabet Σ of size $|\Sigma| = \sigma$. The ith prefix of S is the string $S_i = s_1, \ldots, s_i$.

Def. 2: A pattern $P = p_1 p_2, \ldots, p_n$ is the concatenation of n subpatterns that are expressions in some query language. The length of the pattern is given by $|P| = \sum_{j=1}^{n} \pi_j$, where π_j is the maximum number of characters in the string that is matched by the jth subpattern. Multiple patterns are separated by commas. For example, $p_1, p_2, \ldots, p_n$ denotes n separate patterns.

Def. 3: The hit function $H(S_i, p_j)$: $S \times P \rightarrow \{0, 1\}$ is a function returning 1 if a pattern p_j matches some suffix of S_i, and 0, otherwise. For instance, if $S_i =$ "regular" and $p_j =$ "ar", then $H(S_i, p_j) = 1$.

Items 1)–6) can now be formalized using Defs. 1, 2, and 3. The concatenation of two patterns requires a hit function

$$H(S_i, p_{j-1}p_j) = H(S_{i-\pi_j}, p_{j-1}) \wedge H(S_i, p_j). \quad (1)$$

When the subpatterns are the smallest possible constructs (single characters) the hit function represents the exact matching of strings. To illustrate, consider the pattern $P =$ "tg" and the string $S =$ "atg". Matching P on the third prefix of S can then be formalized as $H(\mathrm{atg}, \mathrm{tg}) = H(\mathrm{at}, \mathrm{t}) \wedge H(\mathrm{atg}, \mathrm{g})$.

An alphanumerical comparison is the operation of matching substrings alphabetically or numerically related to a given pattern. Given an alphanumeric operator, $\alpha \in \{\leq, <, >, \geq\}$, the hit function becomes

$$H(S_i, \alpha(p_{j-1}p_j)) = H(S_{i-\pi_j}, \alpha(p_{j-1})) \vee (H(S_{i-\pi_j}, p_{j-1}) \wedge H(S_i, \alpha(p_j))). \quad (2)$$

Thus, given the pattern $P = \leq$ "tg" and the string $S =$ "agg", the hit operation can be formalized as $H(\mathrm{agg}, \leq \mathrm{tg}) = H(\mathrm{ag}, \leq \mathrm{t}) \vee (H(\mathrm{ag}, \mathrm{t}) \wedge H(\mathrm{agg}, \leq \mathrm{g}))$.

Using this formalism on boolean operators is straightforward, and the hit function yields

$$H(S_i, f(p_1, p_2)) = f(H(S_i, p_1), H(S_i, p_2)) \quad (3)$$

where f denotes the boolean function. Given the pattern $P =$ "t" | "g", where | denotes the boolean *or* function, and the string $S =$ "g", the hit function becomes $H(\mathrm{g}, \mathrm{t} \mid \mathrm{g}) = (H(\mathrm{g}, \mathrm{t}) \mid H(\mathrm{g}, \mathrm{g}))$.

Let $\eta(p_0, \ldots, p_m, n)$ denote a pattern that requires n of m different patterns to match simultaneously. The hit function can be written

$$H(S_i, \eta(p_1, \ldots, p_m, n)) = \sum_{k=1}^{m} H(S_i, p_k) \geq n.$$

A concatenated pattern version, which requires the matching of n of m subexpressions, becomes

$$H(S_i, \eta(p_1, \ldots, p_m, n)) = \sum_{k=1}^{m} H(S_{i-\sum_{\ell=k+1}^{m} \pi_\ell}, p_k) \geq n. \quad (4)$$

Note that (4) becomes the familiar Hamming distance threshold when the subpatterns are single characters. As a result, matching the pattern $P = \eta(\text{"tg"}, 1)$ in the substring $S =$ "agg" then becomes $H(\mathrm{agg}, \eta(\mathrm{tg}, 1)) = (H(\mathrm{ag}, \mathrm{t}) + H(\mathrm{agg}, \mathrm{g})) \geq 1$.

Hit lingering allows a match to have a prolonged effect in that it can continue to be reported for some (possibly unlimited) time. That is, the construct requires a pattern, P, to match a prefix S_j within a distance, d, before the current prefix S_i, i.e.,

$$H(S_i, \delta(P, d)) = H(S_j, P) \wedge 0 \leq d \leq i - j. \quad (5)$$

Thus, when the pattern is $P = \delta(\mathrm{t}, 1)$ and the string is $S = \mathrm{atg}$, then $H(S_3, P) = 1$, because $H(\mathrm{atg}, \delta(\mathrm{t}, 1)) = H(\mathrm{at}, \mathrm{t}) \wedge 0 \leq 1 \leq 3 - 2$.

Furthermore, using the hit function formalism, it is possible to define constructs like *near* (two patterns should occur within some distance of each other) and *before* (one pattern should occur before the other within some distance). The former construct can be written

$$H(S_i, \delta(p_1, d)) \wedge H(S_i, \delta(p_2, d))$$

while the latter becomes

$$H(S_{i-\pi_2}, \delta(p_1, d)) \wedge H(S_i, p_2).$$

The defined hit functions suffice to match regular expressions without Kleene closures since both unions and concatenations are supported. Also, every expression containing a skip $P?$, which is shorthand for $(P|\epsilon)$, can be rewritten into the union of an expression containing P and an expression with P removed.

Repeated patterns, $P^+ = PP^*$, can also be described by the framework introduced in this section. The matching of the concatenation between a repeating pattern, p_2, and a preceding pattern, p_1, is formally expressed by the hit function

$$H(S_i, p_1 p_2^+) = H(S_{i-n\cdot\pi_2}, p_1) \wedge_{k=0}^{n-1} H(S_{i-k\cdot\pi_2}, p_2). \quad (6)$$

Thus, when $P = \mathrm{tg}^+$ and $S = \mathrm{tgg}$, the hit function is $H(\mathrm{tgg}, \mathrm{tg}^+) = H(\mathrm{t}, \mathrm{t}) \wedge H(\mathrm{tg}, \mathrm{g}) \wedge H(\mathrm{tgg}, \mathrm{g}) = 1$.

The following section describes an architecture for implementing these hit functions in hardware.

IV. MISD Architecture

Recall that (1)–(4) in Section III are recursive. That is, the evaluation of $H(S_i, P)$ requires partitioning P into its atomic components, and combining the individual results using the appropriate functions. The recursion can be visualized by drawing a recursion tree where the result is found by propagating the results from the leaf nodes, i.e., the pattern's atomic components, to the root of the tree. Some of the hit function definitions require that S_i is evaluated in parallel, i.e., (3), while others are strictly sequential, i.e., (1), (2), and (4).

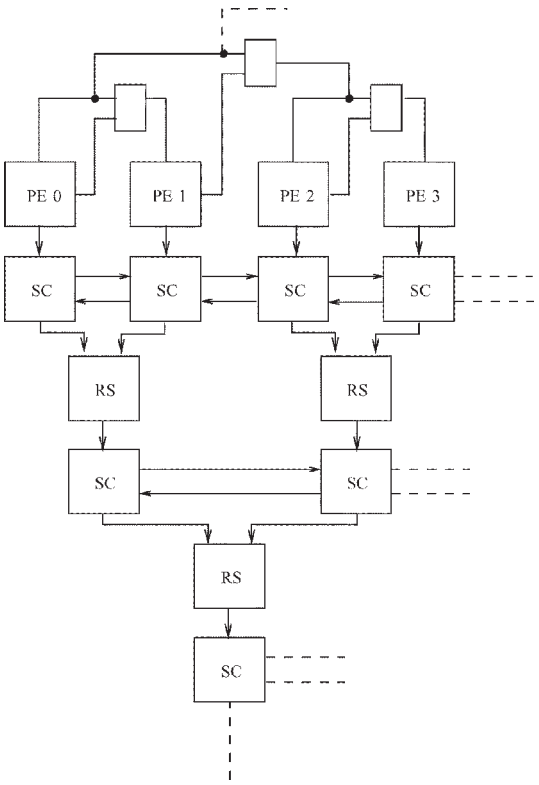

Fig. 1. Search core with data distribution tree (top), processing elements, and result processing nodes (bottom). Distribution nodes receive data in sequence or in parallel. Sequence control (SC) is applied to determine the left and right neighbor's current hit status, while result selection nodes (RS) propagate their results according to their configured hit function.

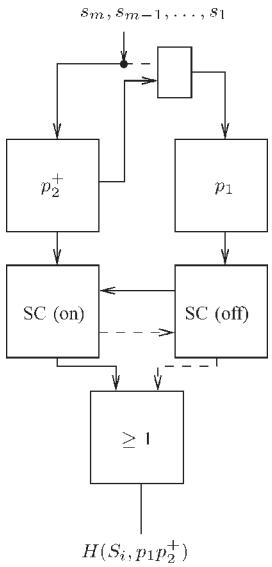

Fig. 2. The tree illustrates how the pattern $p_1 p_2^+$ would be implemented in the proposed architecture, using sequence control for the left subtree responsible for matching occurrences of p_2. Note that the bottom node reports the final hit disregarding any input from its right child (dotted line).

These observations motivate an architecture with two complete binary trees that are responsible for data distribution and result processing, respectively. The nodes of the former tree should be able to distribute either the data from its parent or from the rightmost leaf node of its left subtree. That is, all subtrees on all levels can receive the data in sequence or in parallel. The leaf nodes performing single character comparisons are processing elements (PEs) that are shared between the trees. These compare their current character, s_i, to a preconfigured value, p_j. (The characters are shifted in by the data distribution tree.) The operations $p_i = s_j$, $p_i \geq s_j$, $p_i \leq s_j$, and $p_i \neq s_j$ are sufficient for the proposed functionality, including alphanumerical comparisons. In addition, they facilitate additional features, such as matching characters that belong to alphanumerical ranges.

The internal nodes of the result processing tree receive their input from their children. Denote the results from a node's left and right child H_ℓ and H_r, respectively. The node's result is given by a dynamically configurable function $H\colon H_\ell \times H_r \rightarrow H$. Note that the result is reported as the final hit if the node is the root of the pattern's recursion tree, or propagated to its parent node otherwise. The hit functions $H_\ell = H_r$, $H_\ell > H_r$, $H_\ell \geq H_r$, $H_\ell + H_r$, $H_\ell + H_r \geq c$, $H_\ell + H_r \leq c$, and $H_\ell + H_r = c$ are sufficient to support the outlined functionality of Section III. Here, c is some positive integer value. In addition, any node should have the possibility of being disabled, meaning that it should continuously report 0. To enable alphanumerical comparisons as specified in (2), the equality signal from the leaf nodes is also propagated up the result processing tree. That is, the PEs output $p_i = s_j$, and the internal result processing nodes would propagate $\mathrm{eq}_\ell \wedge \mathrm{eq}_r$, where eq_ℓ and eq_r denotes the equality signal from a node's left and right child, respectively. A conceptual view of the architecture is given in Fig. 1.

Recall from Section III that watching concatenations containing repeating patterns, i.e., $p_1 p_2^+$, amounts to knowing if there was a hit for the first part of the expression at a position $n \cdot \pi_2$ characters earlier, as well as $n - 1$ consecutive matches for the second part. Consider the construct illustrated in Fig. 2, which exemplifies how the problem may be solved in the proposed double binary tree architecture. The first part of the pattern is matched by the right subtree, which corresponds to the box labeled p_1. This subtree receives data sequentially from its left sibling responsible for matching the second part of the pattern, namely p_2. However, the p_2 part cannot propagate a positive result to its parent node unless either 1) its right sibling responsible for matching p_1 also has a hit or 2) the node itself reported a hit π_2 characters ago. Note that the former condition must be met before entering (possibly) multiple occurrences of the latter situation. Hence, all nodes must receive information from their neighboring node on all levels of the result processing tree. This kind of *sequence control* is actually facilitated in both directions as this, in some cases, allows cheaper implementation of skipping patterns (details omitted). The sequence control signals are illustrated with horizontal arrows between neighboring result processing nodes in Figs. 1 and 2. In addition, a flip-flop chain must be associated with each node. Thus, by feeding the chain with a bit according to its current hit state, the kth bit of the chain represents the hit state k clock cycles earlier.

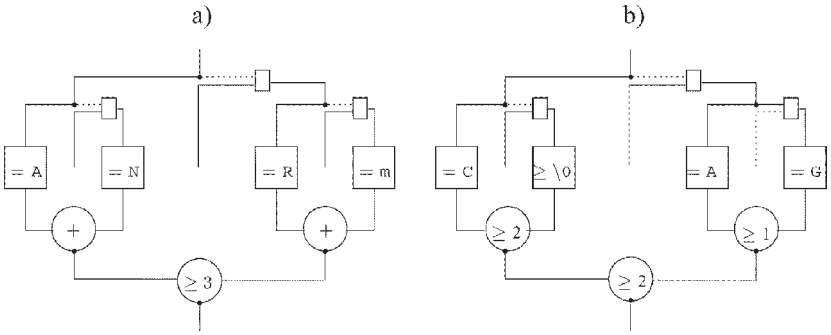

Fig. 3. Illustration of PMC configuration for the queries a) {mRNA:p >= 3}, that is match all strings of length four that match at least three out of four characters in mRNA, and b) (G | A).C, i.e., a G or an A followed by a wild card and a C.

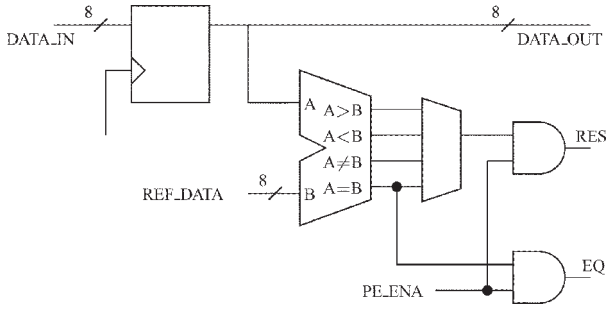

Fig. 4. The PE implementation.

Fig. 3 shows two architecture configurations. The left tree matches all strings of length four with up to one mismatch with the pattern mRNA. The solid lines represent the data path, which is serial between all nodes in this case. The first level result processing nodes transmit $H_\ell + H_r$, while their parent outputs the boolean $H_\ell + H_r \geq 3$. Since the PEs are configured to match the characters of the pattern, the top result processing node will produce the desired result.

Note that the datapath is not sequential for both subtrees in the second example: the rightmost subtree has parallel data distribution, which means that the PEs of this tree receive the same data. As the result processing node gives $H_\ell + H_r \geq 1$ this translates to a hit if either A or a G is currently shifted in. The left subtree has sequential data distribution as in the previous example, and is responsible for matching the subpattern .C : the wildcard matching PE is simply configured to match every character with a byte value greater than or equal to zero.

V. Implementation

The architecture may be implemented with as many PEs as are allowed by practical implementation constraints. More PEs makes it possible to evaluate more expressions, each representing more complex patterns, in parallel. Area constraints dictate the maximum number of levels in the architecture to 10. Hence, there are 1024 PEs on the level that is shared between the trees (level 0). The implementation of the PE's is as illustrated in Fig. 4. Setting PE_ENA low amounts to disabling the PE, that is both the RES and EQ signals are 0 no matter the result of the current comparison. Data is shifted in from the distribution tree, and the ALU performs the four comparisons between the currently held character and the preconfigured reference character. Note that the bus width is 8 bits, which limits the size of applicable alphabets to $\sigma = 256$ if the patterns are to use a single PE per character comparison (larger alphabets can be used at the expense of more PEs).

The parallel and sequential data distribution of the upper binary tree can easily be accomplished using 2:1 multiplexers on all levels as illustrated in Section IV. However, continuously dividing into eight subtrees until all subtrees on the lowest level contain two PEs, results in an implementation using a series of $2:1$, $3:1$, and $4:1$ multiplexers. The longest chain will be for the rightmost PE whose data path consists of tree $4:1$, and one $2:1$ multiplexer. The data from the stream is thus fed fed to the root node of the data distribution tree, and simultaneously distributed to several collections of PEs belonging to separate queries. The implementation is shown in Fig. 5.

Future implementation involving more PEs will require wider or more levels of multiplexers. For example, doubling the current number of PEs could be implemented with only one additional multiplexer level. However, by using the implementation in Fig. 5, another level of multiplexers is only needed when the number of PEs is equal to 2^{3n+1}, $n \in \{0, 1, \ldots\}$. This means that the current data distribution tree supports up to 4096 PEs without the addition of a new multiplexer level.

For increased clock rates, the data distribution can be pipelined at the expense of pipelining registers as well as delay chains to align data properly. The data distribution is not the timing critical path in the current design. The parallel data distribution can run at the design speed without pipelining. The ability for both serial and parallel data distribution, as well as combinations of these, is crucial for efficient use of all PEs.

Patterns that assume complexities beyond the atomic components and the simplest of combinations, are usually more significant in applications. Hence, the number of bits available for specifying latency increases with tree level. However, all levels are able to set all bits, which yields infinite latency. The number of bits available are $2, 4, 8, 8, 16, 16, \ldots, 16$ for levels $0, 1, \ldots, 10$, respectively. Due to strict area constraints in the implementation it was decided to allow sequence control at levels 0, 2, 4, 6, and 8 only. The length of the flip-flop chains are set to the maximum length of the data path for any given subtree, thus restricted to 1, 4, 16, 64, and 256 at the

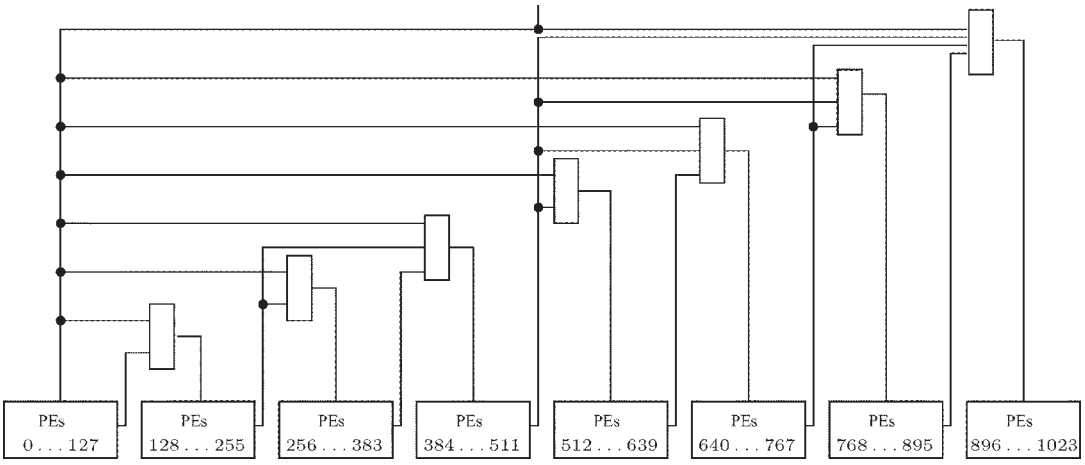

Fig. 5. Data distribution tree implementation using 2 : 1, 3 : 1, and 4 : 1 multiplexers.

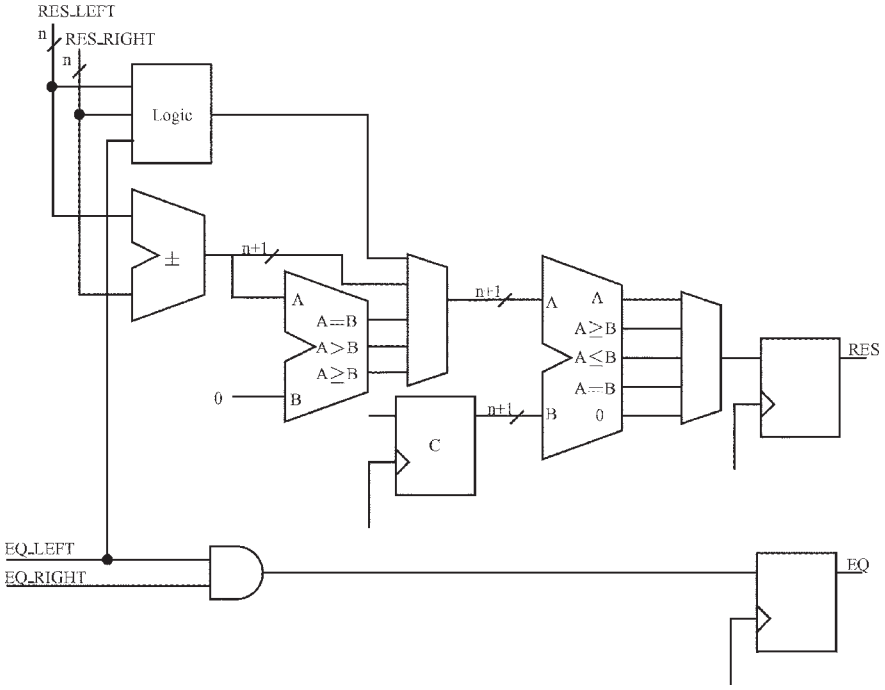

Fig. 6. Core functionality of the result processing nodes, excluding the advanced functionality of sequence control and latency.

respective levels. Note, however, that this does not restrict the functionality, as there is no need to extend the knowledge of a hit beyond the maximum datapath of a given subtree. The core functionality of the result processing nodes, excluding the advanced sequence control and latency features, is implemented as shown in Fig. 6. The first ALU calculates the sum or difference of the input from its children's results. If taking the difference, an optional comparison with zero identifies the subtree with most hits. Alternatively, an alphanumeric comparison is performed at this stage, involving the equality signal from the most significant (left) subtree. The first multiplexer thus receives either a scalar value (when summarizing hits) from the left and right subtrees, or a boolean value encoded in the least significant bit (LSB). This is optionally compared to a configurable constant, c. Furthermore, the combination of the two ALUs can implement any binary function when receiving boolean inputs.

Fig. 7 shows the implementation of sequence control with a flip-flop chain for matching repeating patterns, accompanied by a latency unit that enables holding a result for a predefined number of clock cycles. Results are found as a combination of the output from the connected result processing node (see Fig. 6) and its left and right neighbors. The final result is fed back to the same neighbors. Note that results are either flowing left- or rightward thus avoiding asynchronous loops.

On a practical note, the architecture needs to report hits to the host system for postprocessing. This is accomplished by assigning local *hit managers* to the result processing nodes. The

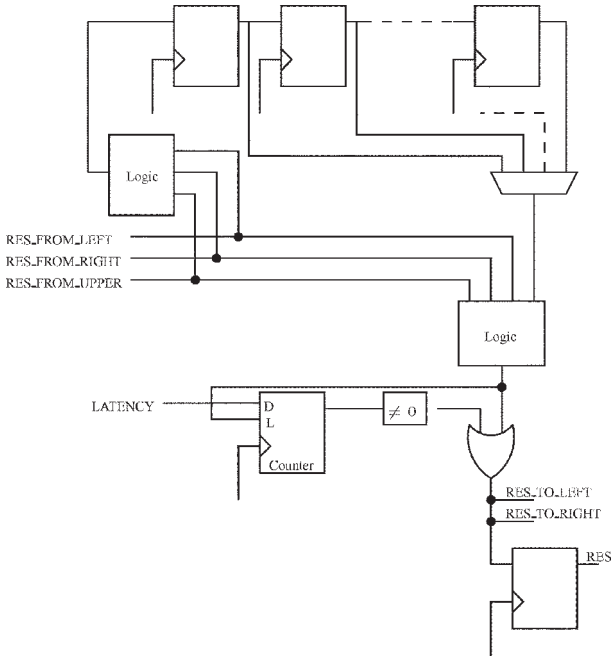

Fig. 7. Sequence control and latency unit implementation.

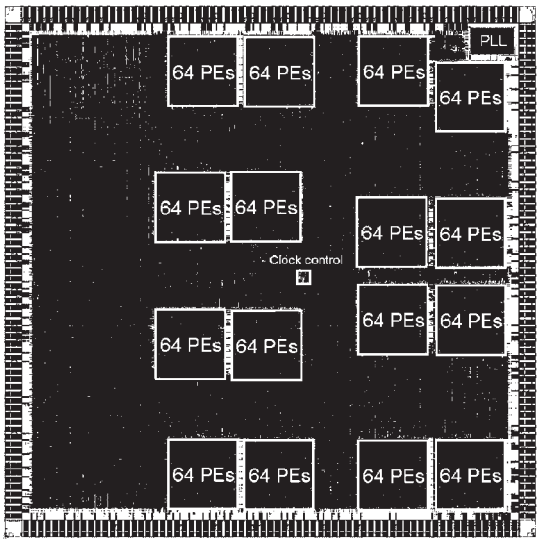

Fig. 8. Chip layout.

hit managers report the address of hits back to host memory using direct memory access (DMA). Alternatively, the hit manager can report the document number where a hit occurred. This is accomplished with the help of a separate unit called the *document manager*, which scans the datastream before it reaches the data distribution tree. The document manager counts the number of documents in the stream by looking for a preconfigured character sequence. The current implementation supports a document separator sequence of up to four characters. Additionally, to prevent the chip from flooding the host system with results, the hit managers can be configured to only report one result from each document or within an address range.

It is assumed that queries requiring very few PEs occur infrequently, thus, limiting the need for area consuming hit managers at lower levels. Hence, hit managers are only assigned to the result processing nodes on level four or higher in the result processing tree. Consequently, the architecture is limited to handling a maximum of 127 parallel queries, where each query has a minimum of 16 PEs available.

For easy support of features such as case insensitive searches, a byte remapping table has been included in the chip. Before the search data is passed to the document manager and data distribution tree, each byte in the stream is remapped by reading its value from the table. Thus case insensitive searches can be performed by mapping "a"-"z" to "A"-"Z", and ensuring that the PEs only matches upper case characters.

The chip is programmed through a set of 32 bit registers, which can be divided into configuration and control registers. The configuration registers consist of one register for each PE, two registers for each node in the twin binary trees (the nodes in the data distribution tree are configured using a single bit for each node, where 0 means serial distribution and 1 means parallel), and two registers for each hit manager. The document manager is programmed using two registers, while the remapping table uses 64 registers. The control registers are used for instance to start or stop searches, or to read general status information. As a frontend to the chip, we have developed a compiler for translating queries in the query language [36], [37] to register configurations. This compiler is part of a larger API, which can be used for integrating the chip in different search applications.

Fig. 8 shows a plot of the physical layout on the chip, implemented in standard cell technology. Groups of 64 PEs were created, and separately routed. Sixteen such macros are included in the design, and can be seen as square regions in the layout. The remaining logic was placed and routed as a flat design.

VI. Comparisons

The architecture proposed in this article is primarily designed for search problems where one would rapidly find the occurrences of one or more patterns in a large database. This means that once a query is supplied by a user, both the delay before the query is evaluated and the time needed to search the database should be as small as possible. An important factor for reaching the former goal, is that the architecture is able to evaluate queries in parallel whenever possible. In other words, as few PEs as possible should remain idle during a search. Our architecture accomplishes this through the use of a flexible data distribution tree and by associating hit managers with the nodes in the result gathering tree. This gives us the possibility to pack queries for parallel evaluation using a simple Huffman procedure. To the best of our knowledge, no other proposed hardware design can achieve this ease and flexibility in parallel query evaluation.

The proposed architecture also supports a wider range of functionality than many other recently proposed architectures, whose main features are edit distance computations [29]–[34]. On the other hand, general edit distance computations are not supported by our architecture. Instead, the more limited

TABLE I
TECHNICAL DATA FOR DIFFERENT ASICS (ADAPTED FROM [33])

System	Number of PEs	Number of Transistors	Silicon Area	Maximum Clock Rate	Max. Character Comparisons	Technology
Proposed	1024	1.17×10^7	114 mm^2	100 MHz	1.024×10^{11} ch/s	0.20 μm CMOS
[33]	64	3.4×10^5	53 mm^2	132 MHz	8.448×10^9 ch/s	0.6 μm CMOS
Kestrel [30], [31]	64	1.4×10^6	60 mm^2	33 MHz	7.04×10^8 ch/s	0.5 μm CMOS
CASM [29]	7	2.047×10^4	30.7 mm^2	40 MHz	2.8×10^8 ch/s	2 μm CMOS
SSE [20], [21]	512	2.176×10^5	110 mm^2	10 MHz	5.12×10^9 ch/s	1.6 μm CMOS

Hamming distance filtering functionality can be used, which is identical to using edit distances without insertions and deletions. This lack of functionality may be offset by the fact that our architecture makes it possible to create expressions where mismatches at different positions in the string have different weights. This functionality is important in several potential applications (see for instance, [38]).

To summarize, Table I compares the technical data of our solution to other recently published text processors. The comparison also includes an older architecture [20], [21], which, like our architecture, does not provide the full edit distance comparisons of the other architectures in Table I. The string–search engine (SSE) of [20], [21] combines an associative memory with finite state automata and is mainly used for (near)[1] exact string searches with variable length wildcards.

Initial tests using our ASIC implementation mounted on PCI cards with SDRAM for holding the dataset, have shown that the chip can achieve its desired search rate of 100 MB/s (data not shown).

VII. DISCUSSION

A functionality for performing approximate pattern matching has been outlined and formally described in terms of hit functions. An MISD VLSI architecture has been proposed. The hardware is suitable for searching an online data stream using queries expressed in languages that support the described functionality.

The current implementation runs at a clock frequency of 100 MHz, and is able to search for up to 127 queries, depending on complexity, at the rate of 100 MB/s. The implementation has 1024 processing elements, so a single chip is able to perform up to 1.024×10^{11} character comparisons per second.

Work has been undertaken to implement a data processing system based on the VLSI chip presented here. Several possible solutions are being investigated, with the primary focus being integration of multiple chips on PCI cards with onboard SDRAM as the primary data source. An implementation with 16 chips on a single PCI card, each chip having 128 MB of dedicated SDRAM, is now in the process of being tested. A standard PC fitted with four of these cards is theoretically able to perform about 7×10^{12} character comparisons per second.

REFERENCES

[1] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[1]The finite state automaton used can match strings with an edit distance of one.

[2] J. Friedl, *Mastering Regular Expressions*, 2nd ed. Cambridge, MA: O'Reilly, 2002.
[3] *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, D. Sankoff and J. Kruskal, Eds., Addison-Wesley, Reading, MA, 1983.
[4] W. Freeman, S. Walker, and K. Vrana, "Quantitative RT-PCR: pitfalls and potential," *Biotechniques*, vol. 26, no. 1, pp. 112–122, 1999.
[5] S. Smith, W. Welch, A. Jakimcius, T. Dahlberg, E. Preston, and D. Dyke, "High throughput DNA sequencing using an automated electrophoresis analysis system and a novel sequence assembly program," *Biotechniques*, vol. 14, no. 6, pp. 1014–1018, 1993.
[6] M. McManus and P. Sharp, "Gene silencing in mammals by small interfering RNAs," *Nature Rev. Genetics*, vol. 3, no. 10, pp. 737–747, 2002.
[7] M. L. Hetland and P. Sætrom, "Temporal rule discovery using genetic programming and specialized hardware," in *Proc. 4th Int. Conf. Recent Advances in Soft Computing*, 2002.
[8] P. Sætrom and M. L. Hetland, "Unsupervised temporal rule mining with genetic programming and specialized hardware," in *Proc. 2003 Int. Conf. Machine Learning and Applications (ICMLA'03)*, 2003, pp. 145–151.
[9] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surveys*, vol. 33, no. 1, pp. 31–88, 2001.
[10] G. Stephen, *String Searching Algorithms*, Singapore: World Scientific, 1994.
[11] P. Michailidis and K. Margaritis, "On-line approximate string searching algorithms: Survey and experimental results," *Int. J. Comput. Math.*, vol. 79, no. 8, pp. 867–888, 2002.
[12] J. Kuri, G. Navarro, L. Mé, and L. Heye, "A pattern matching based filter for audit reduction and fast detection of potential intrusions," in *Proc. 3rd Int. Workshop Recent Advances in Intrusion Detection*, LNCS, 2000, pp. 17–27.
[13] N. Belkin and W. Croft, "Information filtering and information retrieval – 2 sides of the same coin?," *Commun. ACM*, vol. 35, no. 12, pp. 29–38, 1992.
[14] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped blast and psi-blast: A new generation of protein database search programs," *Nucleic Acids Res.*, vol. 25, pp. 3389–3402, 1997.
[15] M. Perrone and L. Cooper, "When networks disagree: Ensemble methods for hybrid neural networks," in *Neural Networks for Speech and Image Processing*, R. Mammone, Ed. London, U.K.: Chapman & Hall, 1993, pp. 126–142.
[16] M. Jordan and R. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm," Univ. Calif. Berkeley, Berkeley, CA, Tech. Rep. AIM-1440, 1993.
[17] M. J. Foster and H. T. Kung, "The design of special-purpose VLSI chips," *Computer*, vol. 13, no. 1, pp. 26–40, 1980.
[18] A. Halaas, "A systolic VLSI matrix for a family of fundamental searching problems," *Integration, VLSI J.*, vol. 1, no. 3, 1983.
[19] H. Cheng and K. Fu, "VLSI architectures for string matching and pattern matching," *PR*, vol. 20, pp. 125–141, 1987.
[20] H. Yamada, M. Hirata, H. Nagai, and K. Takahashi, "A high-speed string-search engine," *IEEE J. Solid-State Circuits*, vol. 22, pp. 829–834, Oct. 1987.
[21] M. Hirata, H. Yamada, H. Nagai, and K. Takahashi, "A versatile data string-search VLSI," *IEEE J. Solid-State Circuits*, vol. 23, pp. 329–335, Apr. 1988.
[22] A. Mukherjee, "Hardware algorithms for determining similarity between two strings," *T-COMP*, vol. 38, pp. 600–603, 1989.
[23] A. Halaas, "Arguments and architectures for ASIC's in future information retrieval systems," in *In Proc, IFIP Workshop Design & Test of ASICs*, 1990.
[24] M. E. Isenman and D. E. Shasha, "Performance and architectural issues for string matching," *IEEE Trans. Computers*, vol. 39, pp. 238–250, 1990.

[25] V. W.-K. Mak, K. C. Lee, and O. Frieder, "Exploiting parallelism in pattern matching: An information retrieval application," *ACM Trans Information Systems*, vol. 9, pp. 52–74, Jan. 1991.
[26] J. Park and K. George, "Efficient parallel hardware algorithms for string matching," *Microprocessors and Microsystems*, vol. 23, no. 3, pp. 155–168, 1999.
[27] Y. Lin and J. Yeh, "A scalable and efficient systolic algorithm for the longest common subsequence problem," *J. Inform. Sci. Eng.*, vol. 18, no. 4, pp. 519–532, 2002.
[28] K.-I. Yu, S.-P. Hsu, and P. Otsubu, "The fast data finder – An architecture for very high speed data search and dissemination," in *Proc. 1st. Int. Conf. Data Engineering*, 1984, pp. 167–174.
[29] R. Sastry, N. Ranganathan, and K. Remedios, "Casm: A VLSI chip for approximate string-matching," *PAMI*, vol. 17, no. 8, pp. 824–830, Aug. 1995.
[30] D. M. Dahle, J. D. Hirschberg, K. Karplus, H. Keller, E. Rice, D. Speck, D. H. Williams, and R. Hughey, "Kestrel: Design of an 8-bit SIMD parallel processor," in *Proc. 17th Conf. Advanced Research in VLSI*, 1997, pp. 145–162.
[31] J. Hirschberg, D. Dahle, K. Karplus, D. Speck, and R. Hughey, "Kestrel: A programmable array for sequence analysis," *J. VLSI Signal Processing Systems for Signal Image and Video Technol.*, vol. 19, no. 2, pp. 115–126, 1998.
[32] J. Lindelien. "The Value of Accelerated Computing in Bioinformatics," Whitepaper. TimeLogic, Crystal Bay, NV. [Online]. Available: http://www.timelogic.com/whitepapers/decypher_benefits_e.pdf
[33] H.-M. Blüthgen and T. G. Noll, "A programmable processor for approximate string matching with high throughput rate," in *Proc. Int. Conf. Application-Specific Systems, Architectures, and Processors (ASAP'00)*, 2000, pp. 309–316.
[34] B. Schmidt, H. Schroder, and M. Schimmler, "Protein sequence comparison on the instruction systolic array," in *Proc. 6th Int. Conf., Parallel Computing Technologies*, vol. 2127, Lecture Notes in Computer Science, V. E. Malyshkin, Ed., Novosibirsk, Russia, Sept. 2001, pp. 498–509.
[35] R. Hughey, "Parallel hardware for sequence comparison and alignment," *CABIOS*, vol. 12, no. 6, pp. 473–479, 1996.
[36] M. Nedland, B. Svingen, and M. Hetland. (2002) "The Interagon Query Language – A User's Guide," Whitepaper. Interagon AS, Trondheim, Norway. [Online]. Available: http://www.interagon.com/pub/whitepapers/IQL.overview-latest.pdf
[37] ——, (2002) "The Interagon Query Language – A Reference Guide," Whitepaper. Interagon AS, Trondheim, Norway. [Online]. Available: http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf
[38] M. Amarzguioui, T. Holen, E. Babaie, and H. Prydz, "Tolerance for mutations and chemical modifications in a siRNA," *Nucleic Acids Res.*, vol. 31, no. 2, pp. 589–595, 2003.

Arne Halaas is a Professor in algorithm construction at the Norwegian University of Science and Technology, Trondheim, Norway, where he was Head of the Computer and Information Department from 1982 to 1984 and from 1993 to 1994. From 1981 to 1982, he was a Visiting Professor at the University of Kaiserslautern, Kaiserslautern, Germany. From 1994 to 1995, he was also a Visiting Professor at Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier, Montpellier, France. Since 1981, he has worked with special purpose search engines and cofounded three companies operating in this field: Turbit AS (1987), Fast Search & Transfer ASA (1997), and Interagon AS (2002). Fast Search & Transfer ASA is now a major global actor on search engines, while Interagon AS has specialized on complex search tasks, primarily in bioinformatics, requiring high-performance solutions for imprecise, small-fragment searching, and pattern discovery.

Prof. Halaas has served on the Editorial Board of the VLSI Journal *Integration*, since it was established in 1983. He took an active part in establishing the International Federation for Information Processing (IFIP) working group 10.5 on VLSI systems in 1981, and has been involved for over a decade in the arrangements of its biannual series of VLSI conferences.

Børge Svingen received the M.Sc. degree in computer science from the Norwegian University of Science and Technology, Trondheim, Norway, in 1996.

In 1997, he cofounded Fast Search & Transfer ASA, Oslo, Norway, and then in 2002 he cofounded Interagon AS, Trondheim, Norway. He has worked with machine learning, genetic programming, pattern matching and discovery, special-purpose hardware, and architecture and algorithm design. He is now the Chief Technology Officer of IMP Technology AS, Kråkerø, Norway, a database search engine company.

Magnar Nedland received the M.Sc. degree in computer science from the Norwegian University of Science and Technology, Trondheim, Norway, in 2000.

From 2000 to 2001, he was a Systems Engineer with Fast Search & Transfer ASA, Oslo Norway. In 2002, he cofounded Interagon AS, where he is currently a Research Scientist and specializes in C++ software development and architecture design for pattern-matching integrated circuits. He has been involved with the hardware design and development since 1998.

Pål Sætrom received the M.Sc. degree in Computer Science from the Norwegian University of Science and Technology, Trondheim, Norway, in 2000, where he is currently pursuing the Ph.D. degree using machine learning in biological discovery.

He was a Systems Engineer with Fast Search & Transfer ASA, Oslo, Norway, from 2000 to 2001. In 2002, he cofounded Interagon AS, Trondheim, Norway, where he is currently a Research Scientist. He has been involved with hardware design and development since 1998, and has used the chip as a hardware accelerator for challenging genetic programming experiments in his research.

Ola Snøve, Jr. received the M.Sc. degree in mechanical engineering from the Norwegian University of Science and Technology, Trondheim, Norway, in 2000.

From 2000 to 2001, he was a Systems Engineer with Fast Search & Transfer ASA, Oslo, Norway. In 2002, he cofounded Interagon AS, Trondheim, Norway, where he is currently a Research Scientist working with biology applications and business development for the chip architecture.

Olaf René Birkeland received the M.Sc. degree in computer science from the Norwegian University of Science and Technology, Trondheim, Norway, in 1993.

From 1993 to 1998, he was the Director of Engineering with MRT Micro ASA, Oslo Norway/Intelens Inc., Boca Raton, FL. From 1998 to 2001, he was Senior Research and Development Manager with Fast Search & Transfer, Oslo, Norway, ASA. In 2002, he cofounded Interagon AS, Trondheim, Norway, where he is currently Chief Technology Officer. He is a specialist in the development of high-performance hardware for search systems and video processing, and core competencies include ASIC design and development, electronics design, hardware architecture, software design, algorithm design, and system architecture.

Paper II

The petacomp machine — A MIMD cluster for parallel pattern-mining

The Petacomp Machine — A MIMD Cluster for Parallel Pattern-mining

Olaf René Birkeland[1], Ola Snøve Jr.[1], Arne Halaas[2], Magnar Nedland[1], Pål Sætrom[1,2]

[1]Interagon AS
St. Olavs hospital
Trondheim, Norway

[2]Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Abstract

Multiple instruction stream-single data stream (MISD) architectures have not found many practical applications in supercomputing. We present a multiple instruction stream-multiple data stream (MIMD) cluster implementation that uses MISD search processors with extreme pattern mining performance as a building block.

We use PCI cards that hold sixteen search processors with local memory to build a relatively small cluster of five PC nodes with six PCI cards each. This cluster can handle anything between 64 independent queries at 48 GB per second, to 30,720 independent queries at 100 MB per second.

The cluster's performance characteristics are such that we can easily scale the system to 10^{15} operations per second with containable overhead using just 100 nodes. Further, the solution has lower power consumption and memory bandwidth requirements than comparable technologies.

1 Introduction

Supercomputers nowadays are either single machines with sophisticated architectures or many relatively simple machines in a cluster configuration. The trend is towards clusters as illustrated by the popularity of Linux clusters. Furthermore, IBM's BlueGene/L cluster machine is number one on the November 2005 list of the world's top supercomputers on `http://www.top500.org`. Clusters are relatively cheap alternatives to integrated machines, and have traditionally enabled people to work with problems that have been too demanding for standard workstations, but that do not require the use of every clock cycle of a vector machine (capacity versus capability computing) [35].

Still, cluster supercomputers are not cheap, and cluster solutions based on standard components carries several costs in addition to acquiring the hardware. For example, the estimated cost for a 65,536 node BlueGene/L is less than $800,000,000, but still requires 9 GWh of electricity annually for the machine itself [26]. The cooling, maintenance, and hardware replacement costs of Google's 15,000 node search cluster are considerable [2].

Our aim was to build a small cluster of machines with special-purpose search processors to enable pattern mining with performance comparable to modern supercomputers at a fraction of the cost. The cluster consists of several PCs that are equipped with PCI boards that contain multiple instruction stream-single data stream (MISD) search processors with local memory. With the possible exception of systolic arrays [7], MISD architectures have been considered impractical [16], and our search processor may be one of the first MISD architectures to find practical applications (cf. [35]). Figure 1 shows the various building blocks of our pattern mining cluster, achieving a performance as given by Table 1.

2 Motivating example from modern genetics

DNA molecules can be viewed as two sequences where characters in a four-letter alphabet base-pair due to preferential binding in GC and AT pairs. RNA molecules are normally single stranded, which means there is only one sequence, but RNA molecules can also form double-stranded structures by GC and AU base-pairing.

Many methods in genetics use short DNA or RNA probes to selectively bind to longer molecules that have regions with base-pairing potential. For instance, short interfering RNAs (siRNAs) are 21 base-pair RNAs that can bind to messenger RNAs—the intermediates that are translated to proteins—and thereby decrease the protein output of its target. These siRNAs should be as specific to their targets as possible in order to avoid side-effects, and it is therefore desirable to build a library of specific probes.

Formally, you have an alphabet of four characters that

1-4244-0328-6/06/$20.00

Figure 1. A MISD processor die containing 1024 parallel elements are packaged and mounted in quantities of 16 chips on a PCI accelerator card. Multiple cards are inserted into a PC node, which servers as the building block for the cluster.

can be divided into two different pairs that have a strong preference for each other. A major determinant for success is the degree of similarity between the probes and the target sequence as measured by the Hamming distance as similarity to sequences other than the target sequences results in poor performance.

We use the notation of [25], and let $\delta(\mathbf{x}, \mathbf{y})$ denote the Hamming distance between two equally long strings $\mathbf{x}$ and $\mathbf{y}$. We define the k-neighborhood of $\mathbf{x}$ as all strings $\mathbf{y}$ that satisfy $\delta(\mathbf{x}, \mathbf{y}) \leq k$. Note that multiple probe matches within some region is usually only counted once. For example, the specificity of a microarray probe is not compromised due to multiple binding sites within the same mRNA transcript. Therefore, we let $\mathbf{T} = (\mathbf{t}_1, \ldots, \mathbf{t}_m)$ denote a target database with m documents that may correspond to genes or other biological entities.

A library of specific oligos can be built for any entity and in general consists of patterns that are unique to that particular entity with some degree of fuzziness—that is, the pattern is found only in the target entity even if a matching region is allowed to differ somewhat from the exact pattern. A special case is the k-neighborhood library, which for the entity $\mathbf{t}_i$ consists of all oligonucleotides, $\mathbf{x} \in \mathbf{t}_i$, where $\delta(\mathbf{x}, \mathbf{y}) > k$ for all $\mathbf{y} \in \mathbf{T} - \mathbf{t}_i$; that is, all probes mismatch in at least k positions with all other equally long oligos from other transcripts. The k-neighborhood library can be useful when selecting siRNA probes, as you want to find binding regions with a high k to other messenger RNAs to avoid side-effects.

The above problem can be solved by measuring the similarity between each candidate subsequence in the target and each subsequence in the rest of the target database. That is, we repeatedly search the target database with a large set of query sequences, and in the case where we want to design oligo probes against each target in the database, the number of searches approach the database size. Thus, the search problem consists of a high number of readily available queries that is screened against a static document collection. What is more, all queries can be screened against the database in parallel with very little overhead.

3 Cluster implementation

In the following, we will describe the design and implementation of a high-performance search cluster, intended to solve problems similar to the k-neighborhood library problem. More specifically, the cluster is designed to solve search problems that share the important characteristics of (i) being dividable into independent subproblems that can be solved in parallel with minimal overhead; (ii) consisting of a large number of queries; and (iii) having a relatively static dataset. Note in particular that our search architecture's functionality permits far more advanced queries than will be used in the motivating example, but in the interest of simplicity, we will describe the architecture in the context of the k-neighborhood problem that use simple mismatch

similarities.

The design will be described bottom up, starting with our previously published special purpose search architecture [14], the PCI card implementation, and how these are the cornerstones in our cluster of PCs with otherwise ordinary specifications.

3.1 A MISD search architecture

We developed an application specific integrated circuit, the Pattern Matching Chip (PMC), designed to obtain extreme performance on search problems involving complex patterns [11, 12, 14]. Each chip provides 1.024×10^{11} character comparisons per second and permits searching up to 64 independent patterns at 100 MB per second.

To understand the main principle of our design, visualize the overall operation of the PMC as a stream of data flowing through the chip from left to right. The data is distributed to 1,024 processing elements (PEs) via a binary data distribution tree. The results from the PEs' comparisons are then used to obtain the final output in a result processing tree. We illustrate the chip's basic function with two simple queries, namely `x|y` and `xy`. That is, either an `x` or a `y` in the former, and an `x` directly followed by a `y` in the latter. Figure 2a) shows how this is mapped to two PEs matching the respective characters, have them receive the data stream in parallel, and make the result processing tree perform a boolean `OR` operation by reporting a match if the sum of its two children's results is greater than or equal to one. Note that the data flow is illustrated by solid lines in the data distribution tree. Similarly, the expression `xy` is matched if the data flow becomes sequential—that is, the rightmost PE receives the data flow from its left neighbor—and the result processing node's operation is changed to the `AND` operator, as shown in figure 2b).

These two queries illustrate how the PMC can utilize parallel and sequential data distribution, and use the boolean `OR` and `AND` operators in the result gathering tree to obtain the desired result.

The processing elements and the result processing nodes can perform several more advanced operations. The functionality is sufficient to implement limited regular expression matching [13] excluding nested Kleene closures with constant response time in arbitrary data. A description of the architecture along with advanced configuration examples have been published elsewhere [14], and a detailed technical note is available from the authors upon request.

3.2 A PCI search card for PC integration

We designed the PMC chip with an integrated PCI interface for wide system compatibility. The host system has full control over each individual PMC through transparent PCI-PCI bridges. Each card holds 16 PMC chips along with local memory, typically 2 GB per card, which gives 128 MB of dedicated memory per PMC. With 1,024 PEs per chip, the card carries an accumulated 16,384 PEs within one full length PCI card. The card is powered through the PCI slot and has a peak power consumption less than 25W. The distribution of processing across several chips results in no local hot spot, providing sufficient heat management through passive cooling of the card. Consequently, there are no moving parts, and that results in less power consumption, less noise, and increased reliability. These favorable features are highly important when scaling into a larger system.

Using exclusively low-profile surface mount components, the cards can be stacked side by side in adjacent PCI slots without restricting the system's airflow. In a typical system, this allows six PCI cards to be inserted into each server, or a total of 98,304 PEs per machine at 100 MHz each. This accumulates to about 10^{13} operations per second. Any server grade power supply easily handles this added system load of 150W.

3.3 Resource scheduling for optimal scalability

The search problems we are considering scale in two independent dimensions, namely (i) query volume and (ii) data size. Risvik describes a general framework for designing clusters to handle this kind of search applications [29]. The main principles in this framework are partitioning of data to handle larger data sizes, and duplication of data to handle larger query volumes. These design principles have also been used in the Google search engine [2].

We use the above partitioning principles in our cluster implementation. First, because of the 128 MB of memory dedicated to each PMC, we partition a given dataset of size d MB on $p = \lceil d/128 \rceil$ PMCs. To minimize communications between nodes in the cluster when joining the search results, we generally divide the data on PMCs located on the same cluster node. Nevertheless, our cluster implementation can also handle searches in larger datasets that must be partitioned on several nodes. For example, in a cluster where each node has 6 search cards, giving a maximum size per node of $6 \times 16 \times 128$ MB $= 12$ GB, we partition a 20 GB dataset on two nodes. In the following, we will, however, only discuss search problems where the dataset can fit on one node.

Second, we duplicate the dataset on the remaining PMCs in the cluster. Thus, in a cluster with n nodes, each equipped with m search cards, we could at search up to $\left\lfloor \frac{n \cdot m \cdot 16}{p} \right\rfloor$ instances of the dataset in parallel. Requiring that each dataset is located on the same node reduces the number of instances to $n \cdot \left\lfloor \frac{m \cdot 16}{p} \right\rfloor$.

Given some highly parallel search problem, having a

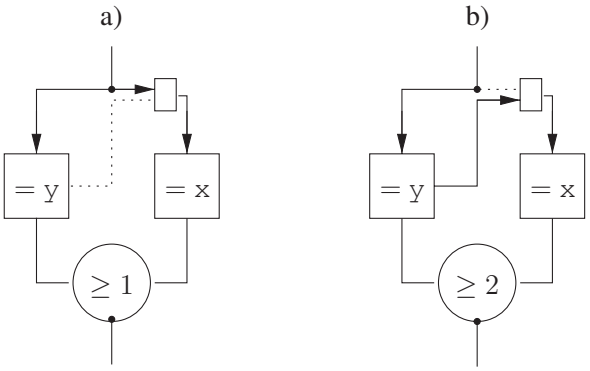

Figure 2. Data flow, character matching, and result processing for queries a) `x|y` and b) `xy`. The data is fed to the righthand element through a multiplexer for either a) parallel or b) serial evaluation.

large number of queries that can be parallelized with little overhead, our current partitioning and scheduling algorithm works as follows:

1. *Cluster distribution.* The total query space is divided evenly on the nodes in the cluster.

2. *Node distribution.* Each node process duplicates the dataset to be searched on its available PMCs, and runs a multi-threaded parallel search on these PMCs.

To illustrate this scheduling process, consider the k-neighborhood library problem described in Section 2. First, we evenly distribute the $|T|$ entities against which the library should be designed to the nodes in the cluster. Second, a separate thread at each node parses the entities, generates queries for each of the subsequences to be evaluated, and pushes the queries on a synchronized search queue. The PMC search threads read the queue, run the searches, and write the results to a common result pool for post processing. In Section 4.4, we compare the performance of this solution to the theoretical maximum performance of our cluster.

3.4 A cluster with great flexibility

Our current search cluster consists of five rack-mounted PCs, each being single Pentium4® CPU systems with 1 GB RAM running disk-based Debian Linux as is freely available from `http://www.debian.org`. The CPU speed ranges from 2.4 GHz to 2.8 GHz. Each node has six search cards for a total of $5 \cdot 10^{13}$ comparisons per second. We currently connect the nodes through a 100 Mbps Ethernet switch, but as each node has a gigabit Ethernet interface, we can easily upgrade the cluster network to 1 Gbps if needed.

As Table 1 indicates, we can reach 1 peta comparisons per second with 100 cluster nodes. To get this size, we can extend our current fully meshed design by adding additional routers and switches. Alternatively, we can use the tree-based design of [29]. The mesh design is more flexible, but the tree design will reduce network communications, as this can allow dedicated nodes for post-processing if required. We would therefore prefer the tree design when, for instance, searching datasets that must be distributed over more than one cluster node.

4 Results

We now present the characteristics of our cluster solution. That is, we present the requirements for maximal search throughput in the cluster, the memory bandwidth characteristics, and the cluster's power consumption. Finally, we compare the cluster's observed performance to its theoretical maximum performance on the k-neighborhood screening problem outlined in Section 2.

4.1 Theoretical search throughput and scalability

As outlined in Section 3.3, the two main scalability factors to consider are query volume and data size. The following calculations assume that the entire dataset can be held in the PMC's local memory. During any search, the CPU will upload the configuration for the next search into local memory. The configuration upload time for a single pass in each server is

$$t_c = \frac{M \cdot C}{S} \tag{1}$$

where M is the number of PMCs in each server, C is the configuration image size for each chip, and S is the effective

Table 1. Parameters for different system sizes using the current PCI card. Memory per PMC can be configured between one 64 Mbit SDRAM chip to four 1Gbit devices. 128 MB per PMC is used in this table.

System	PMCs	PEs	Comparisons/s	Memory	Bandwidth	Power
One chip	1	1,024	10^{11}	128 MB	100 MB/s	1 W
PCI card	16	16,384	$1.6{\cdot}10^{12}$	2 GB	1.6 GB/s	25 W
One server	96	98,304	10^{13}	12 GB	9.6 GB/s	350 W
5 nodes	480	491,520	$5{\cdot}10^{13}$	60 GB	48 GB/s	1750 W
100 nodes	9600	9,830,400	10^{15}	1.2 TB	1 TB/s	35 kW

PCI bandwidth. In one machine with ninety-six PMCs, t_c becomes approximately 30 ms given an effective PCI bandwidth of 50 MB per second.

The search time for a single pass of the data is given as

$$t_s(n) = \frac{n}{\theta_{max}}, \tag{2}$$

where n is the amount of data distributed to each chip, and θ_{max} is the search speed of a single PMC.

Hence, the effective search throughput for each chip is given as

$$\theta_s(n) = \begin{cases} \theta_{max} & \text{if } t_c < t_s(n) \\ \frac{t_s(n)}{t_c}\,\theta_{max} & \text{otherwise} \end{cases} \tag{3}$$

We may combine equations 1 through 3 to obtain the minimum dataset n that must be used for a system of M PMCs to run at peak bandwidth. A single PMC must search more than three megabytes per pass, and a six card machine with ninety-six PMCs consequently must be configured with almost 300 megabytes or more. As the search time is linearly dependent on the data size, the search throughput will not be altered if data is added beyond the minimum requirement.

Adding more queries can be handled either by running queries in parallel on more PMCs, alternatively evaluating groups of queries serially. Either way, more queries or data only requires a linear increase in run time or compute resources.

4.2 MISD provides better usage of available memory bandwidth

In a conventional SIMD processor architecture, the memory bandwidth is usually a major performance bottleneck. Additional memory latency reductions demanded by increasing CPU clock rates, come at the expense of further increased bandwidth requirements [6]. Traditional processors compensates the lack of bandwidth with SIMD vector processing to minimize instruction stream bandwidth [24]. Still, there is a high load on the memory subsystem. System testing shows a range from 0.2 bytes of data traffic per instruction for computationally intensive programs, to 7 bytes per instruction for memory intensive benchmarks [28].

The MISD architecture of the PMC implies that each memory access is used more efficiently, i.e. by being processed simultaneously by several PEs. Even a petacomp cluster consisting of 100 servers (see Table 1) will achieve full performance with $100 \cdot 6 \cdot 16 \cdot 100$ MB/s $= 1$ TB/s bandwidth. Ordinary low-cost SDRAM can provide this. Note that the PMC architecture does not have a separate instruction stream requiring bandwidth during searches as the instructions are stored in configuration registers inside each chip.

4.3 Reduced power consumption

With an energy-efficient MISD architecture, the PMC system requires very little power to operate. In a server configuration, 35 pJ is used per character comparison including system overhead (cf. table 1). As a comparison, a CPU requires about 150 times more energy per character comparison assuming 200 W system power and the optimistic theoretical performance described in Section 6.3.

The power consumption by the PMC itself is around 10 pJ per comparison. The Cell processor [18] uses in comparison roughly 5 pJ based on a (unrealistic) 100% utilization of 16-wide SIMD instruction issue on every clock. Based on the same assumptions, the Sun UltraSPARC T1 processor with less parallelism and higher chip power consumption uses around 250 pJ per comparison [21].

4.4 Application performance with near linear scalability

As outlined in Section 2, the k-neighborhood screening problem can be solved by measuring the similarity between each candidate subsequence in the target and each subsequence in the rest of the target database. In the following, we present our k-neighborhood screening solution.

To do a k-neighborhood screening, we use the PMC's Hamming distance functionality [14]. The binary tree struc-

ture of the PMC's data distribution and result gathering tree results in that a k-neighborhood screening of a string of length n will use a number of PEs equal to n rounded up to the nearest power of 2, that is

$$\pi(n) = 2^{\lceil \log_2 n \rceil} \tag{4}$$

PEs. As a single PMC has 1,024 PEs, it can handle $1024/\pi(n)$ k-neighborhood screenings in parallel. So, for example, one PMC can screen 32 25mers at once (a 25mer is a nucleotide subsequence of length 25). Note that the number of parallel screenings is independent of the size of the neighborhood.

A single PMC can screen up to 128 MB at a rate of 100 MB per second. Thus, if we only consider databases smaller than 128 MB, a single PMC can theoretically get a throughput (short oligonucleotide queries per second) of

$$\theta(d, n) = \frac{1024}{\pi(n)} \cdot \frac{100}{d}, \tag{5}$$

where d is the size of the sequence to be screened (in MBs) and $\pi(n)$ is defined in (4). By considering more than one PMC, we can extend this result to an arbitrary data size:

$$\theta(d, n, p) = \theta(d, n) \cdot p, \tag{6}$$

where p is the number of PMCs used and $\theta(d, n)$ is defined in (5). This means that four PMCs screening 25mers in a database of 200 MB will have a throughput of 64 25mers per second. A node in our search cluster, having six PCI cards, achieves a theoretical throughput of 1,536 25mers per second on the same database.

To test the system, we built libraries that contain the most specific 25mers from any genomic transcript in the latest Ensembl release of Human cDNA [4]. This dataset is 65 MB, which means that it fits on a single PMC and that the PMCs can run at full search speed (see Equation (3)).

Figure 3 shows the true performance plotted against the theoretical performance when using more PMCs in the oligonucleotide screening application for a single cluster node. Note that the performance is nearly linearly scalable when adding mode nodes, hence increasing the number of PMCs will increase the performance accordingly. The system was set up with non-overlapping configuration, search and result processing. As these can be pipelined, the $\sim 90\%$ utilization can be further increased.

In general, the standard similarity search algorithms from computational biology cannot be used as these either lack in performance as is the case with Smith-Waterman [32] or in sensitivity as is the case with BLAST [1] (cf. [33]). To put our performance figures in perspective, we compare them to the results reported by [25], who used a dynamic programming algorithm to create a complete k-neighborhood library for a small dataset of 10^6 nucleotides. They screened 10^4 25mers in approximately one hour on a single CPU of a Compaq GS80 server. This gives a throughput of approximately three 25mers per second on this dataset. Because the search time of their algorithm scales linearly with the size of the database, they would have a throughput of about 0.04 25mers per second (or about three 25mers per minute) when screening the human transcriptome. This means that they would need about 10^3 CPUs to reach the throughput of a single PMC and nearly 10^5 CPUs to reach the throughput of a single PMC server.

Recently, Yamada and Morishita [36] reported an index-based solution for k-neighborhood screenings of 19mers, with $k \leq 4$, to be used in RNAi experiments. Using a database of human transcripts (its exact size not listed), they report a throughput of 140 ($k = 3$) and 37 ($k = 4$) 19mers respectively per second on a Dell Precision 650 with a 3.2 GHz Xeon CPU and 2 GB main memory. Although this throughput is comparable to that of a single PMC chip (for $k = 3$ and $k = 4$ one of their CPUs is equivalent to 3.1 and 0.82 PMCs), their solution do not share the PMCs scalability, power efficiency and flexibility for handling other known aspects of sequence similarity than k-neighborhood [10, 31]. Table 2 summarizes these comparisons.

5 Other pattern mining applications

The k-neighborhood screening problem referred to throughout this paper, is an example of a search problem where you want to find instances in the dataset that satisfy some known properties. The opposite problem occurs when you have a dataset with some known properties, and you want to create a model that characterizes parts of—or even the complete—dataset. In the former problem, you know the query and want to find the data that the query matches; in the latter, you want to find the queries that characterize the data. The latter problem is also known as data mining.

We have previously described a boosted, hardware accelerated, genetic programming algorithm [30], which creates models that characterize sequences belonging to some conceptual class. In [30], we used this algorithm to create models that predicted whether short oligonucleotides were effective when used in antisense and RNAi experiments.

Cluster-based solutions are common when using machine learning to solve data mining problems (for example [8, 22]), and there are two main reasons for this. First, using machine learning requires several independent experiments to establish good method accuracy (for example bootstrap [9] and cross-validation [5, 34]—compared in [20]). These independent experiments require no coordination, and are consequently easy to run in parallel [8]. Second, many data mining problems require large CPU re-

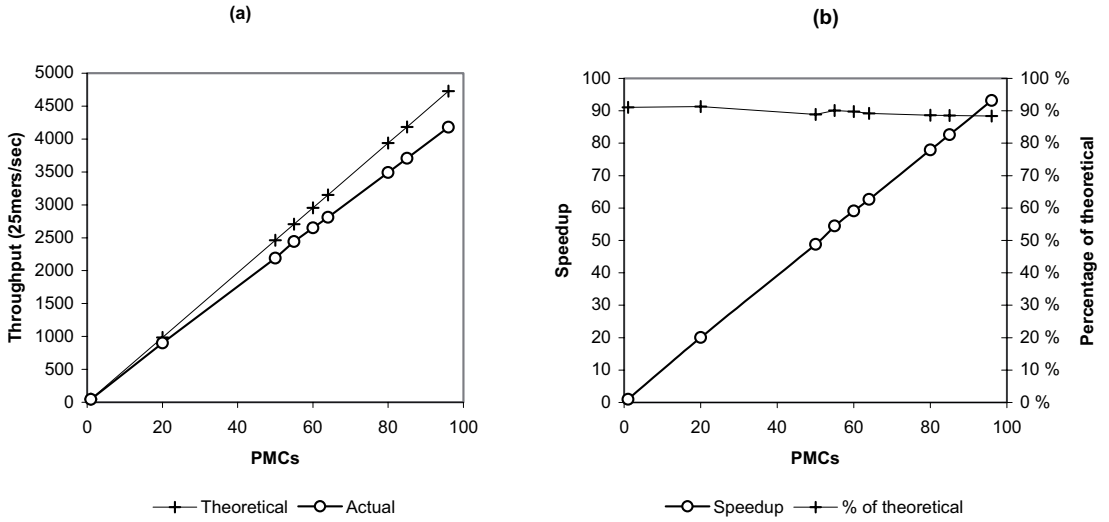

Figure 3. Performance statistics for 25mer screenings of the human transcriptome on a single cluster node. The theoretical performance is plotted with the actual performance (a). A single PMC achieves around 90% of the theoretical maximum performace. Adding mode PMCs results in linear speedup compared to one chip (b). Each siRNA is screened versus a dataset of 65 MB.

Table 2. Throughput and energy consumption for different siRNA screening implementations versus a 65 MB dataset. Throughput in the PMC-based solutions is not dependent on the size of the k-neighborhood. The PMC based solutions have the same throughput for all siRNAs of length 17–32.

System	25mers/s	Power	Energy per 25mer
One PMC node	4180	350 W	0.084 J
5 node PMC cluster	20900	1750 W	0.084 J
Melko et al	0.04	1150 W	28750 J
Yamada et al ($k = 3$)†	140	460 W	3.3 J
Yamada et al ($k = 4$)†	37	460 W	12.4 J

†Yamada et al used less demanding 19mers in their results.

sources to be solved, hence parallelizing the algorithm may be the only way to get results [22, 23].

We use the above approaches to parallelize our data mining algorithm on the search cluster. First, we do several independent runs on each cluster node—the number of parallel runs depending both on the number of PMCs available and the total CPU load. Second, we partition large datasets on several PMCs, not only to handle datasets larger than 128 MB, but also to speed the search process on smaller datasets (as per Equation 3). In addition to bioinformatics, we have applied this within seismic data processing, financial knowledge mining, and network surveillance [3].

6 Potential system enhancements

Depending on the application, it may be preferable with slightly different designs. In this section, we list some alternatives that can easily be implemented on our existing search card if required by a specific application.

6.1 I/O processor on card

Adding an I/O processor to each accelerator card is the obvious next step for improving the system performance. These improvements come from one or more of the following factors:

- *Reduced configuration time.* Maintaining 16 PMCs on each card, these are now configured from the local I/O processor instead of the CPU. This alters the configuration time in Equation (1) by reducing M to a fixed value of 16. Correspondingly, optimal query throughput can be achieved with as little as 0.5 MB of data per PMC for each query.

- *Faster data loading.* Loading of data can be handled locally without CPU intervention. This could be from a shared system resource like the disk drive, or with direct connection to a storage subsystem. Combining broadcasted writes with a sufficient data storage, a throughput of 800 MB per second are achievable.

- *Reduced main CPU load.* The I/O processor can parse and map high level queries to PMC register configurations. This offloads the CPU computationally and in bandwidth as only compact high level queries need to be sent to each card. In practice, each I/O processor could run the cluster node's applications.

An alternative to implementing an I/O processor on the card would be to reduce the host server to a minimum. This could for example be a blade server managing a limited number of PMCs. Most commercial blade servers include a proprietary expansion slot, thus such a design would have to be customized for a specific server brand. The blade server solution would be more costly and larger than using an I/O processor, but with the benefit of a single processor architecture for the software.

6.2 Network interface on card

In the network surveillance applications the data can be fed directly to the card by adding a network interface. This would most likely be combined with an I/O processor as described in section 6.1, which will handle the network stack as well as all of the PMC resources.

6.3 CPUs instead of PMCs

The high production volumes of CPUs allow extensive design and manufacturing efforts. Consequently one should expect a CPU to achieve higher clock rates than a standard cell ASIC like the PMC. With an increasing number of parallel pipelines in the CPU, the CPU's number of operations per second will approach the PMC's. For example, a 3.8 GHz Pentium4® processor can in one clock cycle execute eight byte comparisons with the streaming single instruction multiple data (SIMD) unit, potentially in parallel with two ALU-operations [17]. Theoretically, this accumulates to 38 billion comparisons per second, approaching the 100 billion comparisons for a single PMC chip. Despite this narrow gap in performance, the PMC architecture has several advantages that makes it more feasible for high-performance pattern mining clusters than a standard CPU.

- *The PMC operates closer to peak performance.* The SIMD architecture of the CPU allows parallel comparisons, but not parallel branching dependent of individual results. For anything but long fixed keyword comparisons, which are easily handled by indexing rather than brute force comparisons, the CPU will not reach its peak performance. The MISD architecture of the PMC chip, or the MIMD architecture of our cluster, is more appropriate.

- *The PMC executes patterns directly, without overhead.* Evaluating regular expressions is much more than the low level comparisons. While the PMC has separate units for handling these functions, a CPU must use the same processing core as for the comparisons themselves. Fine grained pattern matching results in a large number of data dependent branching, which effectively kills the performance of super-scalar specular out-of-order execution.

- *The PMC's design has a much higher potential.* In this comparison, the CPU has a technology advantage being fabricated on a 90 nm process. With a similar

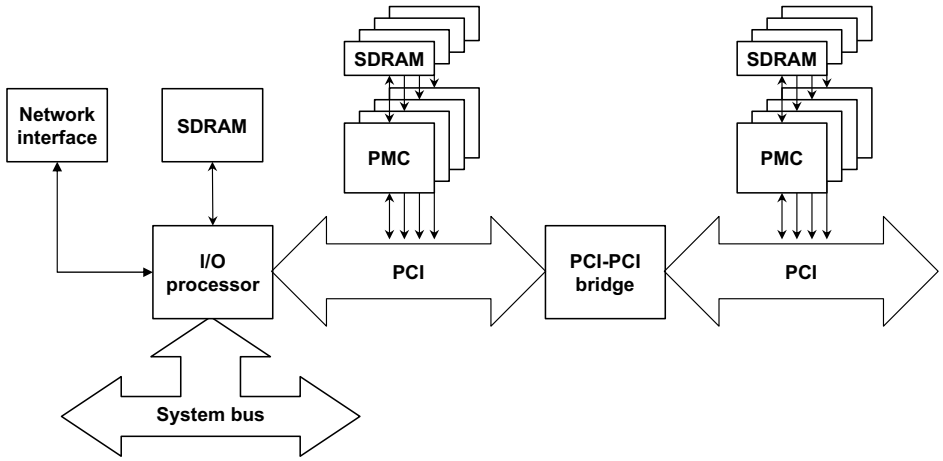

Figure 4. Block diagram of accelerator card with built in I/O processor and optional network interface. The I/O processor might also connect directly to other peripherals like a storage subsystem.

process the PMC would integrate five times more processing elements, in addition to a potential increase in operating speed.

- *The PMC has much lower power requirements.* The thermal design power of the CPU used is 115 W, excluding required support circuitry and memory. For the PMC, this number is 1 W for the chip alone, 1.5 W including peripherals and memory.

Taking these factors into account, the performance advantage of the PMC increases, especially when building a petacomp cluster (see Table 1). Using `nrgrep` [27] on a 1 GHz Pentium3® as a benchmark for evaluating regular expressions, a *single* PMC demonstrated a three orders of magnitude increase in speed [15]. The PMC system also scaled better with increasing data volumes.

6.4 Integration of PMC and memory

Even denser systems can be built by integrating memory and processing on the same die. This approach is limited to relatively small memory arrays, e.g. 8 MB per chip, as embedded memory can not be packed as dense as in a separate memory chip. As an intermediate alternative, a multi chip module (MCM) can be constructed.

Any such integration eliminates the memory sizing flexibility with the current configuration, as well as the cost advantage of using standard memory components. The integration also implies suboptimal implementation of both the compute and memory function [19, 28]. It would thus only be viable for applications with moderate memory volume demands.

7 Summary of this work

We have presented a supercomputing cluster for pattern mining purposes. The MIMD cluster is based on search processors with MISD architectures, and it seems that this is one of the first MISD implementations ever to find practical applications. The search processor's architecture is patented [11, 12] and details on its functionality and performance in a single chip configuration has been published elsewhere [14].

We have demonstrated that the performance of our cluster is orders of magnitude higher for pattern mining purposes than can be obtained with similar-sized clusters of machines with ordinary CPUs. The high computational density is due to a power efficient core element with a high utilization of the available memory bandwidth.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, 1990.

[2] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[3] O. R. Birkeland, M. Nedland, and O. Snøve Jr. Massively parallel MIMD system achieves high performance in

a spam filter. In *Proceedings of Parallel Computing (ParCo)*, September 2005. to be printed.

[4] E. Birney, D. Andrews, P. Bevan, M. Caccamo, G. Cameron, Y. Chen, L. Clarke, G. Coates, T. Cox, J. Cuff, V. Curwen, T. Cutts, T. Down, R. Durbin, E. Eyras, X. Fernandez-Suarez, P. Gane, B. Gibbins, J. Gilbert, M. Hammond, H. Hotz, V. Iyer, A. Kahari, K. Jekosch, A. Kasprzyk, D. Keefe, S. Keenan, H. Lehvaslaiho, G. McVicker, C. Melsopp, P. Meidl, E. Mongin, R. Pettett, S. Potter, G. Proctor, M. Rae, S. Searle, G. Slater, D. Smedley, J. Smith, W. Spooner, A. Stabenau, J. Stalker, D. Storey, A. Ureta-Vidal, C. Woodwark, M. Clamp, and T. Hubbard. Ensembl 2004. *Nucleic Acids Res.*, 32(1):468–470, 2004.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[6] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. In *23rd International Symposium on Computer Architecture (ISCA)*, pages 78–89. IEEE Computer Society, May 1996.

[7] A. Chalmers and J. Tidmus. *Practical Parallel Processing*. International Thompson Computer Press, 1996.

[8] I. Dutra, D. Page, V. Santos Costa, J. Shavlik, and M. Waddell. Toward automatic management of embarrassingly parallel applications. In *Euro-Par 2003 Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 509–516, 2003.

[9] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

[10] S. M. Elbashir, J. Martinez, A. Patkaniowska, W. Lendeckel, and T. Tuschl. Functional anatomy of siRNAs for mediating efficient RNAi in drosophila melanogaster embryo lysates. *EMBO J.*, 20(23):6877–6888, 2001.

[11] Fast Search & Transfer ASA. Digital processing device, 2000. International publication number WO 00/22545.

[12] Fast Search & Transfer ASA. A processing circuit and a search processor circuit, 2000. International publication number WO 00/29981.

[13] J. E. Friedl. *Mastering Regular Expressions*. O'Reilly, Cambridge, MA, 2nd edition, 2002.

[14] A. Halaas, B. Svingen, M. Nedland, P. Sætrom, O. Snøve Jr., and O. R. Birkeland. A recursive MISD architecture for pattern matching. *IEEE Trans. on VLSI Syst.*, 12(7):727–734, 2004.

[15] M. L. Hetland and P. Sætrom. A comparison of hardware and software in sequence rule evolution. In *Eight Scandinavian Conference on Artificial Intelligence*, 2003.

[16] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Book Company, 1985. page 32–35.

[17] Intel Corporation. IA-32 Intel® architecture optimization reference manual. Technical report, Intel Corporation, 2004. This manual is available from `http://developer.intel.com/design/Pentium4/documentation.htm`.

[18] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM Journal on Research & Development*, 49(4/5):589–604, July/September 2005.

[19] G. Kirsch. Active memory: Micron's Yukon. In *17th International Parallel and Distributed Processing Symposium (IPDPS)*, page 89. IEEE Computer Society, 2003.

[20] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of the 14th IJCAI*, pages 1137–1143, 1995.

[21] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, March/April 2005.

[22] J. R. Koza, David Andre, F. H. Bennett III, and M. Keane. *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Fransisco, CA, Apr 1999.

[23] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[24] C. Kozyrakis and D. Patterson. Overcoming the limitations of conventional vector processors. In *30th International Symposium on Computer Architecture (ISCA)*, pages 399–409. IEEE Computer Society, Jun 2003.

[25] O. Melko and A. Mushegian. Distribution of words with a predefined range of mismatches to a DNA probe in bacterial genomes. *Bioinformatics*, 20(1):67–74, 2004.

[26] N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In IEEE, editor, *SC2002: From Terabytes to Insight. Proceedings of the IEEE ACM SC 2002 Conference*, 2002.

[27] G. Navarro. NR-grep: a fast and flexible pattern matching tool. *Software Practice and Experience (SPE)*, 31:1265–1312, 2001.

[28] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A case for intelligent RAM: IRAM. *IEEE Micro*, 17(2):34–44, Apr 1997.

[29] K. M. Risvik. *Scaling Internet Search Engines: Methods and Analysis*. PhD thesis, NTNU, Trondheim, Norway, May 2004.

[30] P. Sætrom. Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063, 2004.

[31] S. Saxena, Z. O. Jónsson, and A. Dutta. Implications for off-target acitivity of small inhibitory RNA in mammalian cells. *J. Biol. Chem.*, 278(45):44312–44319, 2003.

[32] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147(1):195–197, 1981.

[33] O. Snøve Jr. and T. Holen. Many commonly used siRNAs risk off-target activity. *Biochem. Biophys. Res. Commun.*, 319(1):256–263, 2004.

[34] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

[35] A. J. van der Steen and J. J. Dongarra. Overview of recent supercomputers. Technical report, EuroBen, October 2004. This 14th issue of the annual report is available from `http://www.top500.org/ORSC/2004/`.

[36] T. Yamada and S. Morishita. Accelerated off-target search algorithm for siRNA. *Bioinformatics*, 21(8):1316–1324, 2005.

Paper III

Sequence Explorer: interactive exploration of genomic sequence data

Sequence Explorer: interactive exploration of genomic sequence data

Ola Snøve Jr. [a] Håkon Humberset [a] Olaf René Birkeland [a]
Pål Sætrom [a,*],

[a] *Interagon AS, Medisinsk teknisk senter, NO-7489 Trondheim, Norway*

Abstract

Current solutions for complex motif searching in DNA and protein sequences are not interactive as users usually wait tens of seconds before the results can be viewed. We propose a hardware-accelerated client-server solution that is fast enough to retain the interactive feeling even when screening whole genomes.

We structured our framework for interactive sequence analysis around query, dataset, filter, and result presentation modules. The query and dataset specification enable simultaneous, interactive screening of multiple complex queries against several datasets. The filters impose restrictions such as only allowing hits to be reported if they occur in coding regions, and the different result presentations include histograms and hit lists.

Our results show that interactive searching is possible even though response times vary significantly depending on filter, network bandwidth and hit frequencies. With a relatively small server, we obtain response times of about one and a half second on gigabytes of data when queries are sufficiently specific to avoid network bottlenecks due to high hit frequencies.

Key words: siRNA, RNA interference, efficacy prediction

Searching RNA, DNA, and protein sequence data usually means looking for similarities between a query sequence and some database. The Smith-Waterman algorithm [12] is the most sensitive algorithm, but it is very CPU intensive, which is why several heuristic approaches have emerged with great success.

* Corresponding author. Fax: +47 455 94 458
Email addresses: ola.snove@interagon.com (Ola Snøve Jr.),
haakon.humberset@interagon.com (Håkon Humberset),
olaf.rene.birkeland@interagon.com (Olaf René Birkeland),
paal.saetrom@interagon.com (Pål Sætrom).

Notable heuristics include FASTA [8], BLAST [1], ParAlign [9], and PatternHunter [6]. Similarity search algorithms may, however, be too advanced when searching for occurrences and distributions of simple motifs such as repetitive sequences or transcription factor binding sites in genomic data. Regular expressions are widely used for pattern matching in text [3], and specialized versions of the familiar algorithms have been proposed for DNA and protein sequences [11]. Betel and Hogue showed that low-level pattern matching was valuable in identifying genetic targets in a cancer characterized by a high frequency of mutations in coding regions containing mononucleotide repeats [2].

Due to novel algorithms, improved implementations, and increased CPU speed, similarity search algorithms now have acceptable response times when running on large publicly funded and freely available supercomputers. Pattern matching algorithms are also impressive for some purposes, but the process is still not interactive when screening for complex patterns in large volumes of sequence data. We hypothesize that many ideas do not realize their full potential because biologists can not (i) query sequence databases with biological questions; (ii) view the results at the appropriate abstraction level; and (iii) explore the search space and develop their hypotheses interactively. For example, when searching the genome looking for disease genes that display some sequence features found in a family of known genes, a researcher may initially want to focus on genomic positions that are close to known promoter loci. Furthermore, comparing high-level results such as hit rate distributions from several different genomes might be valuable. Finally, observing the results while varying the search parameters such as distance bounds and fuzziness may reveal important differences between genomes.

We have developed a client-server solution that aims to provide interactive searches at different abstraction levels. The client's graphical user interface consists of four main panes that correspond to pattern, filter, dataset, and result specifications. A common feature for all panes is that layered specifications is possible; that is, the user can specify multiple questions and result views that will be executed simultaneously. A screening against all specified datasets is performed whenever the queries change, which means that result differences due to changes in query parameters are observed almost instantaneously. As queries are automatically scheduled for execution when they are constructed, we have removed the familiar "submit" button because we felt that it would prevent the application from being truly interactive. To avoid excessive scheduling, we have introduced a quiet time frame from the last character entry to query submission. Furthermore, to maintain interactivity, an ongoing search is automatically aborted if its results have not been reported at the time its corresponding query is altered in the client. The user may pose restrictions on queries by adding filters such as requiring that only hits in coding regions should be reported. We aim for result presentations that

enable researchers to quickly grasp the important information without being distracted by annotation that is only needed if the results justify careful investigation. That being said, the application provides linkouts to annotation from the region surrounding individual hits.

We use special purpose search processors on PCI search cards to accelerate standard workstations for pattern-matching purposes. The high performance of these processors is critical to get interactive searches in gigabytes of data. Each chip can screen anything from one to sixty-four patterns against 100 MB depending on the queries's complexity [4]. A single chip is three orders of magnitude faster for regular expression-searching than "nr-grep" [7] running on a 1 GHz Pentium III with 256 MB of memory [5]. Furthermore, there are sixteen chips with local memory on the search card, which means that the theoretical throughput of each card is 1.6 GB per second.

The search processor is designed to match fuzzy patterns in arbitrary data. We have developed the Interagon Query Language (IQL) that uses regular expression-like syntax to take advantage of the search processor's functionality. Although similar to regular expressions, the language has features not feasible in software. Especially, this is the case for *n of m* expressions; that is, "match n out of m subparts" where the latter can be everything from a single sequence of characters to complex patterns defined by the language. Moreover, the possibility of specifying that two patterns of arbitrary complexity should be separated by some length, or be present in a specific order, is useful. (See the tutorial in the supplementary information for practical examples on using IQL for interactive exploration of DNA sequence data.) IQL is neither specialized for DNA searching nor optimized with respect to this particular application, but should have sufficient functionality to illustrate the potential of interactive searching. The language does, for example, easily support both Prosite patterns and position weight matrices.

Our results show that we can interactively search entire genomes by using special-purpose pattern-matching hardware to accelerate a standard workstation. Typical response times are a few seconds depending on the query complexity. Because of network transfer limitations, simple patterns with high hit rates get high response times. To reduce this negative effect, we do not report all the results for queries with high hit rates. Simple patterns are however seldom very informative, and we therefore question whether this limitation has practical consequences. Furthermore, filters sometimes hamper the performance, especially if there are many hits being post processed. We are currently working on finding ways to implement faster filters.

We believe that our interactive search tool will be valuable for iterative hypothesis testing and refinement. By integrating a machine learning algorithm that automatically creates pattern hypotheses (for example, [10]), researchers

can also investigate problems where they only have some qualitative description of the desired solution, and thus cannot formulate the initial pattern hypothesis themselves. The interactive tool can then be used to further investigate or refine the pattern hypotheses generated by the machine learning algorithm. We are currently investigating this approach.

Supplementary information

A Java client querying a hosted server is freely available upon request at `http://www.interagon.com/demo/`. Four chromosomes are available in this demo version of the server. In addition to the demo application, a tutorial describing system requirements, installation, and use of the program is available along with a technical note on the Interagon Query Langauge.

Acknowledgements

We thank H.E. Krokan, F. Drabløs, A. Halaas, T.B. Grünfeld, S.H. Fjeldstad, and M. Nedland for valuable help during the development of this demo. The work was supported by the Norwegian Research Council, grants 151899/150 and 151521/330, and the bioinformatics platform at the Norwegian University of Science and Technology, Trondheim, Norway.

References

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215 (3):403–410, 1990.

[2] Doron Betel and Christopher W.V. Hogue. Kangaroo - a pattern-matching program for biological sequences. *BMC Bioinformatics*, 3(1): 20–22, 2002.

[3] Jeffrey E.F. Friedl. *Mastering Regular Expressions*. O'Reilly, Cambridge, MA, 2nd edition, 2002.

[4] Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve Jr., and Olaf Renè Birkeland. A recursive MISD architecture for pattern matching. *IEEE Trans. on VLSI Syst.*, 12(7):727–734, 2004.

[5] Magnus Lie Hetland and Pål Sætrom. A comparison of hardware and software in sequence rule evolution. In *Eight Scandinavian Conference on Artificial Intelligence*, 2003.

[6] Bin Ma, John Tromp, and Ming Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[7] Gonzalo Navarro. NR-grep: a fast and flexible pattern matching tool. *Software Practice and Experience (SPE)*, 31:1265–1312, 2001.

[8] William R. Pearson and David J. Lipman. Improved tools for biological sequence comparison. *J. Mol. Biol.*, 85(8):2444–2448, 1988.

[9] Torbjørn Rognes. ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Res.*, 29(7):1647–1652, 2001.

[10] Pål Sætrom. Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063, 2004.

[11] S.S. Sheik, Sumit K. Aggarwal, Anindya Poddar, N. Balakrishnan, and K. Sekar. A fast pattern matching algorithm. *J. Chem. Inf. Comput. Sci.*, 44(4):1251–1256, 2004.

[12] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147(1):403–410, 1981.

Paper IV

Designing effective siRNAs with off-target control

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Biochemical and Biophysical Research Communications 325 (2004) 769–773
BBRC
www.elsevier.com/locate/ybbrc

Designing effective siRNAs with off-target control

Ola Snøve Jr.[1], Magnar Nedland[1], Ståle H. Fjeldstad, Håkon Humberset, Olaf R. Birkeland, Thomas Grünfeld, Pål Sætrom*

Interagon AS, Medisinsk teknisk senter, NO-7489 Trondheim, Norway

Received 7 October 2004
Available online 6 November 2004

Abstract

Successful gene silencing by RNA interference requires a potent and specific depletion of the target mRNA. Target candidates must be chosen so that their corresponding short interfering RNAs are likely to be effective against that target and unlikely to accidentally silence other transcripts due to sequence similarity. We show that both effective and unique targets exist in mouse, fruitfly, and worm, and present a new design tool that enables users to make the trade-off between efficacy and uniqueness. The tool lists all targets with partial sequence similarity to the primary target to highlight candidates for negative controls.

Keywords: siRNA design; RNAi; Gene silencing; Efficacy prediction; Uniqueness; Specificity; Off-target effects

Sequence-specific knockdown of mRNA is a naturally occurring mechanism in many organisms: posttranscriptional gene silencing in plants [1], quelling in fungi [2], and RNA interference (RNAi) in flies [3], nematodes [4], and mammals [5]. They all have in common that Dicer, a ribonuclease III enzyme, initiates the silencing pathways by cleavage of long double-stranded RNA into shorter duplexes [6]. These short interfering RNAs (siRNAs) are 21–23 nucleotides long and have characteristic 3′ overhangs of two nucleotides [7]. The thermodynamic properties of the siRNA determine which of the two strands is incorporated into the RNA induced silencing complex [8,9], a ribonucleoprotein complex that mediates sequence-specific cleavage of mRNA by recognition of sites complementary to its RNA component [10].

The sequence-specificity of RNAi is still unclear. For example, one group reported that a single central mismatch between the siRNA and its target mRNA is enough to abolish silencing in *Drosophila* [3], whereas another group published conflicting results [11]. Yet another group has shown that siRNAs targeting the human tissue factor generally tolerated single mismatches but that mismatches at the 3′ end of the strand complementary to the mRNA were more harmful than 5′ mismatches [12]. Microarray approaches have not been able to settle the controversy as both widespread [13,14] and non-existing [15,16] off-target gene regulation has been reported. Moreover, there is also a risk that siRNAs may function as microRNAs [17,18], a class of non-coding RNAs that are incorporated in a ribonucleoprotein complex called microRNP that may repress protein translation of mRNA with partial complementarity to the microRNA (see [19] for a review).

Even the earliest siRNA design rules stated that potential sequences should be checked for similarity with

* Corresponding author. Fax: +47 455 94 458.
E-mail addresses: ola.snove@interagon.com (O. Snøve Jr.), magnar.nedland@interagon.com (M. Nedland), staale.fjeldstad@interagon.com (S.H. Fjeldstad), haakon.humberset@interagon.com (H. Humberset), olaf.birkeland@interagon.com (O.R. Birkeland), thomas.grunfeld@interagon.com (T. Grünfeld), paal.saetrom@interagon.com (P. Sætrom).
[1] These authors contributed equally to this work.

0006-291X/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.bbrc.2004.10.097

other genes to ensure that only a single transcript is targeted [20]. We recently showed, however, that most commonly used siRNAs risk off-target gene regulation due to sequence similarity with other transcripts [21].

About one in five randomly selected siRNAs will be effective at silencing their targets. Several criteria and algorithms for rational design of siRNAs have been proposed [20,8,9,22–30]. The methods aim to increase the probability of selecting effective siRNAs. We recently reported that only the Reynolds et al. [22], Ui-Tei et al. [24], Amarzguioui and Prydz [27], and Sætrom (GPboost) [30] methods seem to have a high and stable performance across several independent datasets [31].

Several commercial vendors have offered pools of siRNAs targeting the same transcript to increase the probability of getting target knockdown. (Note that a pool of four siRNAs where each has a 50% probability of being effective has an accumulated 94% chance of being effective assuming independent probabilities.) This approach may not be appropriate, at least not for therapeutic purposes. First, the risk for off-target gene regulation increases with pooled siRNAs as more potential targets with (partial) similarity exist. Second, even siRNAs have been shown to trigger the interferon response [32,33], apparently in a concentration-dependent manner [14]. Third, RNAi may be prone to saturation, which means that unprocessed siRNAs remain in the cell free to enter other cellular pathways [34]. It is therefore important to find targets that are effectively silenced at the lowest possible concentrations and pooled approaches may not be amenable to this requirement.

We aim to bridge the gap between efficacy algorithms and uniqueness requirements and will show that many siRNA target sites that are predicted to be highly effective and sufficiently unique are available for most targets. An online application where the users are able to make qualified trade-offs between predicted efficacy and risk for off-target activity accompanies the results and will be presented throughout this article.

Materials and methods

Datasets. For this study, we performed complete off-target screenings on the mouse, fruitfly, and worm transcriptomes from Ensembl [35]; more specifically, *Mus musculus* version 32b (NCBI: m32), *Drosophila melanogaster* version 3a (NCBI: BGDP 3.1), and *Caenorhabditis elegans* version 116a (NCBI: WS 116).

Hardware. Our online server runs Debian Woody Linux on a 2.8 GHz Pentium 4 processor with 1024 MB RAM and special purpose search processors. The Pattern Matching Chip (PMC; Interagon AS, Trondheim, Norway) is an application-specific integrated circuit (ASIC) designed to provide orders of magnitude higher performance than that of comparable regular expression matchers [36]. The server is equipped with five PCI cards, which amounts to 80 PMCs and a capacity to screen 64 complex patterns against 8.0 GB/s.

Uniqueness algorithm. We evaluate the risk for off-target effects by screening siRNAs for uniqueness in the transcriptome using our special purpose search processors. BLAST [37] is not applicable for this purpose as it is prone to miss potentially important matches when the queries are short [21]. Other heuristics such as FASTA [38], ParAlign [39], and PatternHunter [40] use similar pruning schemes as BLAST to avoid searching parts that are unlikely to contain matches; thus, only the time-consuming Smith–Waterman algorithm [41] is guaranteed to yield complete results. In this particular application we run ungapped Smith–Waterman with special treatment of G:U wobble basepairing; more advanced constructs with insertions and deletions as well as weighting of mismatch positions are also possible using our hardware.

Efficacy algorithms. Several siRNA efficacy predictors run on our online server, including that of Sætrom [30], Amarzguioui and Prydz [27], Hsieh et al. [23], Reynolds et al. [22], Schwarz et al. [8], Chalk et al. [28], Takasaki et al. [29], and Ui-Tei et al. [24]. The algorithms are implemented as previously described by our group in [31].

Availability. Our online demo version screens the mouse transcriptome and is available on http://www.interagon.com/demo/ (requires registration). Full siRNA libraries are available for all sequenced species in commercial and academic partnerships.

Results

Our siRNA design tool is largely based on our previous work with siRNA efficacy [31] and off-target risk [21]. Fig. 1 shows several screenshots from the demo version that is available online.

We have previously shown that unique siRNAs are available, at least for the human transcriptome [21], and that four publicly available efficacy algorithms have a high and stable performance across several datasets [31]. But how many sufficiently unique siRNAs have a high efficacy prediction, and vice versa? Assuming that the probabilities p_u that a sequence is unique and p_e that a sequence is effective are independent yields the probability $p_u p_e$ that the sequence is both unique and effective (according to efficacy predictors).

Table 1 shows the average value of p_e with 95% confidence intervals for *M. musculus*, *D. melanogaster*, and *C. elegans*, given different levels of specificities for the GPboost efficacy predictor [30]. (The output of a GPboost classifier is on a scale from -1 to 1, and 19mers with scores above a threshold value are regarded as effective. Increasing the threshold yields higher specificity, but at the expense of a lower sensitivity.) The values are the average of p_e as has been exhaustively computed for different levels of specificity under the assumption that efficacy is independent of uniqueness.

Fig. 2 shows that unique 19mers will be available for most transcripts of a certain length on all levels up to three mismatches; that is, the siRNAs are unique for one target site even if up to three mismatches are allowed between the siRNA and other sites in the transcriptome. The average transcript of 2000 bp will therefore contain more than one unique siRNA on all uniqueness levels except for $(3,0)$ even if you allow only one percent false negative efficacy predictions ($Sp = 0.99$). Note that about half of the transcripts are expected to contain effective siRNAs at the $(3,0)$ level.

Fig. 1. Screenshots of (A) the selection screen where the users input the RNA sequence or accession number and choose an siRNA efficacy predictor; (B) the scatter plot showing predicted efficacy versus uniqueness for each siRNA; (C) the result overview where the siRNAs are ranked according to predicted efficacy and uniqueness; and (D) the alignment of the ranked siRNAs with potential off-target regions.

Table 1
p_e with 95% confidence intervals when the siRNA efficacy predictor has specificities of 0.99, 0.95, 0.90, and 0.75

Species	Specificity			
	0.99	0.95	0.90	0.75
M. musculus	0.127 ± 0.008	0.214 ± 0.011	0.282 ± 0.012	0.392 ± 0.012
D. melanogaster	0.120 ± 0.002	0.203 ± 0.003	0.270 ± 0.005	0.378 ± 0.009
C. elegans	0.184 ± 0.034	0.294 ± 0.049	0.376 ± 0.056	0.498 ± 0.063

Surprisingly, the assumption that efficacy and uniqueness are independent features may not be entirely valid as demonstrated by the line for *C. elegans* in Fig. 3. These are the true dependencies as the curves have been calculated exhaustively; that is, we have determined both predicted efficacy and uniqueness level for all

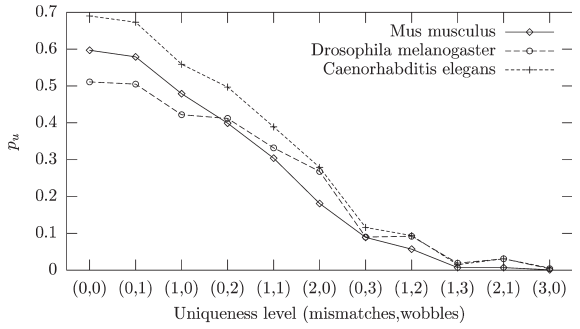

Fig. 2. Probabilities of an siRNA being unique even if a certain number of mismatches and G:U wobbles are allowed between the siRNA and its target.

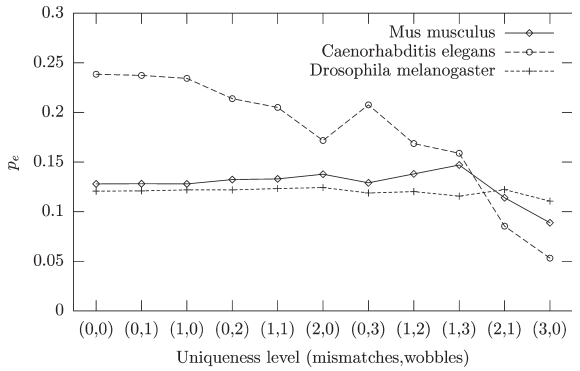

Fig. 3. p_e depends on the uniqueness level, but shows small variations. The given curves for p_e are given for specificity 0.99 for the efficacy predictions; other specificities show similar results (data not shown).

19mers in the given transcriptomes. The averages presented in Table 1 should, however, represent good approximations as the differences in p_e are relatively small. Moreover, it appears that *C. elegans* has a higher fraction of effective siRNAs, and that efficacy decreases with higher uniqueness, whereas *D. melanogaster* and *M. musculus* do not show this dependency. We hypothesized that the lower GC-content in *C. elegans* resulted in higher efficacy prediction values; however, such a dependency was not confirmed when generating random 19mers with identical base composition (data not shown).

It may not be important if a siRNA unintentionally targets a mRNA that is known to be unrelated to the pathway under study. We therefore list all potential off-target matches so that the users are able to evaluate the risk with respect to the biology of their experiments.

Discussion

We have shown that the average transcript of 2000 bp in *M. musculus*, *D. melanogaster*, and *C. elegans* will contain several siRNAs that are both unique and effective. Thus, effective RNAi with risk of off-target effects should be viable for most transcripts from these genomes.

The presented siRNA design tool lists all candidates for off-target mRNA depletion, given that the mechanism depends only on mismatches and G:U wobbles between siRNA and target. We suggest that targets on this list become candidates for negative controls in silencing experiments. We propose, however, that the potential for translational repression by microRNAs will become an even bigger challenge in siRNA design. We therefore work on including algorithms for microRNA off-target effect predictions in future versions of the design tool.

Acknowledgments

We thank O. Sætrom, F. Drabløs, and A. Halaas for valuable comments on the manuscript.

References

[1] D. Baulcombe, Fast forward genetics based on virus-induced gene silencing, Curr. Opin. Plant Biol. 2 (2) (1999) 109–113.
[2] N. Romano, G. Macino, Quelling: transient inactivation of gene expression in *Neurospora crassa* by transformation with homologous sequences, Mol. Microbiol. 6 (22) (1992) 3343–3353.
[3] S. Elbashir, J. Martinez, A. Patkaniowska, W. Lendeckel, T. Tuschl, Functional anatomy of siRNAs for mediating efficient RNAi in *Drosophila melanogaster* embryo lysates, EMBO J. 20 (23) (2001) 6877–6888.
[4] A. Fire, S. Xu, M. Montgommery, S. Kostas, S. Driver, C. Mello, Potent and specific genetic interference by double-stranded RNA in *Caenorhabditis elegans*, Nature 391 (6593) (1998) 806–811.
[5] S. Elbashir, J. Harborth, W. Lendeckel, A. Yalcin, K. Weber, T. Tuschl, Duplexes of 21-nucleotide RNAs mediate RNA interference in cultured mammalian cells, Nature 411 (6836) (2001) 494–498.
[6] E. Bernstein, A. Caudy, S. Hammond, G.J. Hannon, Role for a bidentate ribonuclease in the initiation step of RNA interference, Nature 409 (6818) (2001) 295–296.
[7] P. Zamore, T. Tuschl, P. Sharp, D. Bartel, RNAi: double-stranded RNA directs the ATP-dependent cleavage of mRNA at 21 to 23 nucleotide intervals, Cell 101 (1) (2000) 25–33.
[8] D.S. Schwarz, G. Hutvágner, T. Du, Z. Xu, N. Aronin, P.D. Zamore, Asymmetry in the assembly of the RNAi enzyme complex, Cell 115 (2003) 199–208.
[9] A. Khvorova, A. Reynolds, S.D. Jayasena, Functional siRNAs and miRNAs exhibit strand bias, Cell 115 (2003) 209–216.
[10] J. Martinez, T. Tuschl, RISC is a 5′ phosphomonoester-producing rna endonuclease, Genes Dev. 18 (9) (2004) 975–980.
[11] A. Boutla, C. Delidakis, I. Livadaras, M. Tsagris, M. Tabler, Short 5′-phosphorylated double-stranded RNAs induce RNA interference in *Drosophila*, Curr. Biol. 11 (22) (2001) 1776–1780.
[12] M. Amarzguioui, T. Holen, E. Babaie, H. Prydz, Tolerance for mutations and chemical modifications in a siRNA, Nucleic Acids Res. 31 (2) (2003) 589–595.
[13] A. Jackson, S. Bartz, J. Schelter, S. Kobayashi, J. Burchard, M. Mao, B. Li, G. Cavet, P. Linsley, Expression profiling reveals off-

target gene regulation by RNAi, Nat. Biotechnol. 21 (6) (2003) 635–637.
[14] S. Persengiev, X. Zhu, M. Green, Nonspecific, concentration-dependent stimulation and repression of mammalian gene expression by small interfering RNAs, RNA 10 (1) (2004) 12–18.
[15] J.-T. Chi, H. Chang, N. Wang, D. Chang, N. Dunphy, P. Brown, Genomewide view of gene silencing by small interfering RNAs, Proc. Natl. Acad. Sci. USA 100 (11) (2003) 6343–6346.
[16] D. Semizarov, L. Frost, A. Sarthy, P. Kroeger, D. Halbert, S. Fesik, Specificity of short interfering RNA determined through gene expression signatures, Proc. Natl. Acad. Sci. USA 100 (11) (2003) 6347–6352.
[17] J. Doench, C. Petersen, P. Sharp, siRNAs can function as miRNAs, Genes Dev. 17 (4) (2003) 438–442.
[18] S. Saxena, Z. Jonsson, A. Dutta, Implications for off-target activity of small inhibitory RNA in mammalian cells, J. Biol. Chem. 278 (45) (2003) 44312–44319.
[19] D.P. Bartel, Micro RNAs: genomics, biogenesis, mechanism, and function, Cell 116 (2) (2004) 281–297.
[20] S. Elbashir, J. Harborth, K. Weber, T. Tuschl, Analysis of gene function in somatic mammalian cells using small interfering RNAs, Methods 26 (2) (2002) 199–213.
[21] O. Snøve Jr., T. Holen, Many commonly used siRNAs risk off-target activity, Biochem. Biophys. Res. Commun. 319 (1) (2004) 256–263.
[22] A. Reynolds, D. Leake, Q. Boese, S. Scaringe, W.S. Marshall, A. Khvorova, Rational siRNA design for RNA interference, Nat. Biotechnol. 22 (3) (2004) 326–330.
[23] A. Hsieh, R. Bo, J. Manola, F. Vazquez, O. Bare, A. Khvorova, S. Scaringe, W. Sellers, A library of siRNA duplexes targeting the phosphoinositide 3-kinase pathway: determinants of gene silencing for use in cell-based screens, Nucleic Acids Res. 32 (3) (2004) 893–901.
[24] K. Ui-Tei, Y. Naito, F. Takahashi, T. Haraguchi, H. Ohki-Hamazaki, A. Juni, R. Ueda, K. Saigo, Guidelines for the selection of highly effective siRNA sequences for mammalian and chick RNA interference, Nucleic Acids Res. 32 (3) (2004) 936–948.
[25] K. Luo, D. Chang, The gene-silencing efficiency of siRNA is strongly dependent on the local structure of mRNA at the targeted region, Biochem. Biophys. Res. Commun. 318 (1) (2004) 303–310.
[26] P. Pancoska, Z. Moravek, U. Moll, Efficient RNA interference depends on global context of the target sequence: quantitative analysis of silencing efficiency using Eulerian graph representation of siRNA, Nucleic Acids Res. 32 (4) (2004) 1469–1479.
[27] M. Amarzguioui, H. Prydz, An algorithm for selection of functional siRNA sequences, Biochem. Biophys. Res. Commun. 316 (4) (2004) 1050–1058.
[28] A. Chalk, C. Wahlestedt, E. Sonnhammer, Improved and automated prediction of effective siRNA, Biochem. Biophys. Res. Commun. 319 (1) (2004) 264–274.
[29] S. Takasaki, S. Kotani, A. Konagaya, An effective method for selecting siRNA target sequences in mammalian cells, Cell Cycle, Epub ahead of print.
[30] P. Sætrom, Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming, Bioinformatics (in press), Epub ahead of print).
[31] P. Sætrom, O. Snøve Jr., A comparison of siRNA efficacy predictors, Biochem. Biophys. Res. Commun. 321 (1) (2004) 247–253.
[32] C. Sledz, M. Holko, M. de Veer, R. Silverman, B. Williams, Activation of the interferon system by short-interfering RNAs, Nat. Cell Biol. 5 (9) (2003) 834–839.
[33] A. Bridge, S. Pebernard, A. Ducraux, A.-L. Nicoulaz, R. Iggo, Induction of an interferon response by RNAi vectors in mammalian cells, Nat. Genet. 34 (3) (2003) 263–264.
[34] L. Scherer, J. Rossi, Approaches for the sequence-specific knock-down of mRNA, Nat. Biotechnol. 21 (12) (2003) 1457–1465.
[35] E. Birney, D. Andrews, P. Bevan, M. Caccamo, G. Cameron, Y. Chen, L. Clarke, G. Coates, T. Cox, J. Cuff, V. Curwen, T. Cutts, T. Down, R. Durbin, E. Eyras, X. Fernandez-Suarez, P. Gane, B. Gibbins, J. Gilbert, M. Hammond, H. Hotz, V. Iyer, A. Kahari, K. Jekosch, A. Kasprzyk, D. Keefe, S. Keenan, H. Lehvaslaiho, G. McVicker, C. Melsopp, P. Meidl, E. Mongin, R. Pettett, S. Potter, G. Proctor, M. Rae, S. Searle, G. Slater, D. Smedley, J. Smith, W. Spooner, A. Stabenau, J. Stalker, D. Storey, A. Ureta-Vidal, C. Woodwark, M. Clamp, T. Hubbard, Ensembl 2004, Nucleic Acids Res. 32 (1) (2004) 468–470.
[36] A. Halaas, B. Svingen, M. Nedland, P. Sætrom, O. Snøve Jr., O.R. Birkeland, A recursive MISD architecture for pattern matching, IEEE Trans. VLSI Syst. 12 (7) (2004) 727–734.
[37] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (3) (1990) 403–410.
[38] W. Pearson, D. Lipman, Improved tools for biological sequence comparison, J. Mol. Biol. 85 (8) (1988) 2444–2448.
[39] T. Rognes, ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches, Nucleic Acids Res. 29 (7) (2001) 1647–1652.
[40] B. Ma, J. Tromp, M. Li, PatternHunter: faster and more sensitive homology search, Bioinformatics 18 (3) (2002) 440–445.
[41] T. Smith, M. Waterman, Identification of common molecular subsequences, J. Mol. Biol. 147 (1) (1981) 403–410.

Paper V

Massively parallel MIMD system achieves high performance in a spam filter

Massively parallel MIMD architecture achieves high performance in a spam filter

O.R. Birkeland[a], M. Nedland[a], O. Snøve Jr.[a]

[a]Interagon AS, Medisinsk teknisk senter, NO-7489 Trondheim, Norway

Supercomputer manufacturing is usually a race for floating point operations, and most therefore opt for a design that allows for the highest possible clock frequency. Many modern applications are, however, limited not only by the number of operations that can be performed on the data, but by the available memory bandwidth. We review the main features of a MISD architecture that we have introduced earlier, and show how a system based on these chips is able to scale with respect to query and data volume in an email spam filtering application. We compare the results of a minimal solution using our technology with the performance of two popular implementations of deterministic and nondeterministic automatons, and show how both fail to scale well with neither query nor data volumes.

1. Introduction

Mainstream parallel computer systems have diverse design targets. Projects like BlueGene/L aim for the highest performance on numerical calculations [6], whereas designs like Green Destiny target efficiency, reliability and availability instead [2]. A common characteristic of most designs is the dominating supercomputer requirement for floating point operations, but several applications, for instance within bioinformatics and network traffic monitoring, are limited by the available memory bandwidth rather than the raw floating point compute power [8,1]. Brute force data mining, that is linear searching through raw data, is one such application.

Fine grain parallelism have been proposed by others, both as minute compute nodes [4] or integration of memory and processing circuitry [8,5]. In our work, we propose an architecture that takes advantage of one of the main assets of synchronous memory technology, which is high sustainable bandwidth during linear access. We review a MIMD architecture for pattern matching with regular expression-type queries, with high performance, high density and low infrastructure requirements. By tailoring the compute logic for the given problem, the system's size and complexity can be reduced to a minimum. Focusing on non-computational applications, we are able to build systems with higher density and more operations per second than traditional clusters and supercomputers.

One application that requires scalability with respect to both query and data volume is email spam filtering. Spam is prevented with a large number of approaches, but most involve (among others) using regular expressions to find common spam patterns. The number of required spam patterns are increasing, as well as the volume of email, resulting in an increasing demand for compute power. Some of the common regular expression evaluators like DFA scanners have severe scalability issues. We propose a scalable architecture for evaluation queries such as regular expressions, that achieves high performance in a compact system by use of fine grain parallel processing.

2. System architecture

We concentrate on the design considerations for this architecture, including general purpose versus special purpose hardware; fine grain versus coarse grain parallelism; compute density; cost of ownership; and requirements for development resources.

Our project started with the observation that an index-based algorithm requires that the keywords are known beforehand, and the realization that an extremely rapid brute force linear search eliminates the need for an indexing step. A continuous linear search also implies that searching becomes independent of the queries's complexity. Since the execution time for such a search is $O(n)$, it becomes important to use the peak memory bandwidth, and have as many independent memory channels as possible.

We developed a fine-grain parallel architecture called the pattern matching chip (PMC) [3], where each node can be as small as a single ASIC and a single memory device. Each PMC contains 1,024 individual processing elements operating on a shared data stream. We constructed a MIMD system with PCI cards, each containing 16 PMC chips, accelerating a standard PC to perform 10^{13} symbol comparisons per second. Each chip has a dedicated memory bank; thus, the memory volume and bandwidth scale at the same rate as the number of processing elements (PEs).

16 or more PEs are used for each query, depending on query size. This allows for up to 64 simultaneous queries per chip. Additional queries can be stored in the local memory, requiring 16 kB for each configuration for all 1024 PEs. Thus several thousand queries can be resident for each PMC, and be used with minimal system involvement, only to specify which queries to use. In spam filtering, the email data can be uploaded to local memory once, and then interactively processed by several sets of queries. If the data rate is lower than the scan rate, the difference can be reclaimed in the form of an increased number of queries. For example, one chip could scan 64 queries at 100 MB/s data rate, or 6400 queries at 1 MB/s data rate. Query configurations, as well as data, can be loaded to local SDRAM, even during searches, without any performance penalties. The performance is linear with respect to the number of queries, the length of the queries, and the data rate. It is also linearly scalable with more PMC chips.

The processing speed is limited by the memory bandwidth in each node. Consequently, the PMC is tailored to run at the memory data rate. As we did not aim for the highest processing clock frequency, a low power design was possible, and that proves essential when small building blocks are used to build dense systems. As all memory accesses take place as linear scans, the required power for searching though the data is minimized as well. Any non-predictable access pattern involves protocol overhead, for example in switching pages within the memory chip, during which a lot of time and energy are consumed.

The same methodology is applied to the configuration of the chip. The entire configuration is read once into the chip and stored in configuration registers. Thus, there are no instruction memory accesses during searches. As a side effect, the full memory bandwidth is available for data transfer.

The large number of small PEs on each chip implies no local hot spot. The same applies when several chips are used on the PCI card, without a requirement for active cooling. The density that can be achieved in the system is thus governed by the transistor density on silicon (PMC and SDRAM), rather than the power consumption density.

Another feat of this design is that smaller cores also imply less engineering. As we target applications where data can easily be distributed for processing, no elaborate interconnection scheme is required. The only custom part in the design is the PMC, and the remaining components are sourced from standard technologies like SDRAM and PCI. Correspondingly, the use of a small, repetitive, orthogonal core element also makes the compiler design simpler. Such compilation also involves minimal optimization steps, and executes very fast. DFA scanners like `lex` could achieve high scan rates, but with an unpredictable compile time.

The architectural concepts have been proven in a five PC cluster, achieving $5 \cdot 10^{13}$ operations per second with near-linear scalability. Performance is achieved by massively distributed compute

resources, which translates to 500,000 processing elements in the aforementioned cluster. Furthermore, our evaluation system has a total of 60 GB of memory, with an aggregated bandwidth of 48 GB per second.

We will compare the performance of the PMC with two popular software algorithms in a email spam filtering application. Our reason for choosing this application as our example, is that we are able to show how we can tailor the performance to yield the appropriate ratio of search speed to query throughput.

3. Problem definition

Spam—unsolicited commercial emails or unsolicited bulk emails—has become a tremendous problem for email users, and is increasing. Users around the world receive billions of spam messages each day and are indirectly forced to bear the cost of delivery and storage of these unwanted emails. A number of ways exist to deal with the problem—they range from simple blacklists of bad senders, words, or IP addresses to complex automated filtering approaches. One of these popular filters is SpamAssassin, a program that uses hundreds of weighted rules to obtain a score for each email header (see `http://spamassassin.apache.org/` for additional details). Patterns are constructed to match popular words used by spammers and a genetic algorithm optimizes the weights so that the combined predictor achieves optimal performance on a manually curated training set. Users must determine the actions that should be taken when an email receives a score that puts a spam label on the message according to the user's predefined thresholds.

SpamAssasin, and similar automatic filters, ensures that a message cannot be deemed spam based on a single rule, and conversely, that a message cannot bypass the filter just by avoiding a single rule. Such filters must evaluate many complex rules per message, which may not be a problem if you have a limited number of users on your system. Centralized hubs, such as those operated by internet service providers, however, must process a substantial query volume with high bandwidth message streams. This almost invariably leads to performance problems because most solutions does not scale well with both query and message volume.

Spam rules of an automatic filter are examples of queries that can be efficiently processed by our PMC. A trigger word such as `viagra` can be written in many ways, including for instance `v|agra`, `vìagrá`, `v*i*a*g*r*a`, and so on. The name of Pfizer's popular treatment for erectile dysfunction can actually be written in well above hundred different ways that must be dealt with by spam filters. We wanted to investigate the performance of widely used regular expression algorithms and compare their performance against that of our PMC-accelerated workstation.

We choose to compare our PMC system with efficient implementations of deterministic and non-deterministic automatons. Deterministic automatons are expected to be fast at the expense of large compile times and memory requirements, whereas their non-deterministic counterparts are expected to have a more compact representation, but as their name suggests, they are nondeterministic in their execution speed.

We compared the performance of our system against two software algorithms; one was a deterministic finite automaton (DFA) generator called `flex` and the other was `nrgrep`, which is essentially a simulated nondeterministic suffix automaton. (See Materials and Methods for the appropriate references.)

When regular expressions are input, `flex` compiles C code for the appropriate DFAs that are constructed to match the expressions. In turn, this code must be compiled into executables that can then be run to scan for the specified patterns in a data stream. `Nrgrep` uses a technique called bit-parallelism to simulate automaton behavior instead of actually generating them.

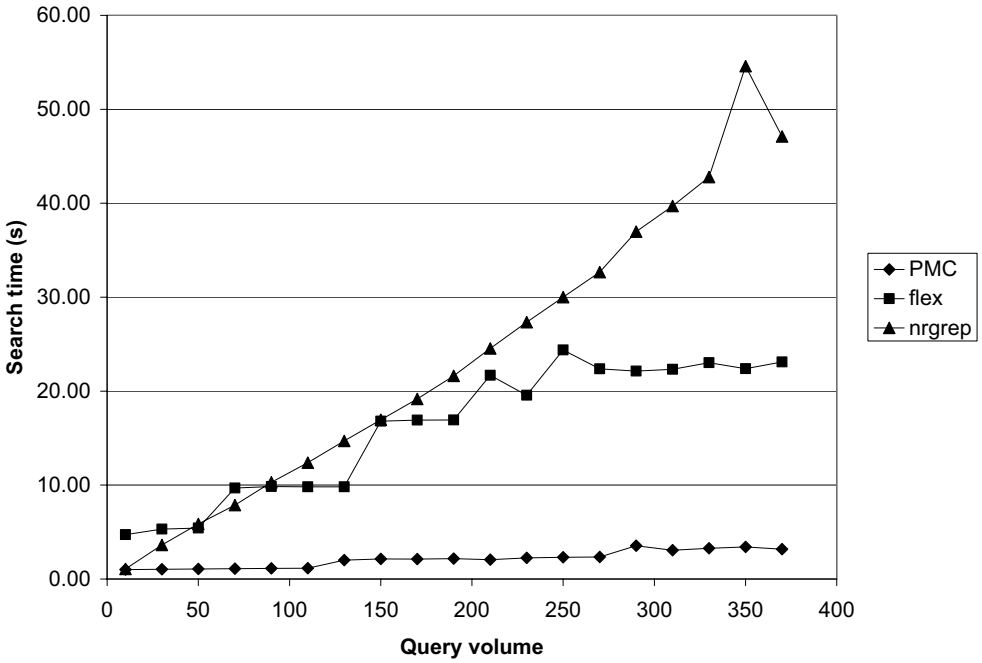

Figure 1. Search time versus query volume for different solutions.

4. Results

In our benchmarks, we risk comparing apples and oranges. For example, should we take into consideration the time it takes to compile the `flex` scanners or should we assume that the application is sufficiently static to render even `flex` compilation negligible eventually. Even though this may not necessarily be true in a real spam filtering solution, we have assumed that dynamic updating of queries will not affect the performance of the `flex` algorithm. Figure 1 shows the search time versus query volume for the PMC, `flex`, and `nrgrep`, respectively. The test was run using 10 duplicate entities of the dataset (see Materials and Methods), but the result is similar when using 1, 2, and 5 entities.

In our benchmark, we see that scalability is the main obstacle of `flex` and `nrgrep`. As the figure shows, we are able to keep a low gradient with respect to the query volume, in contrast to the other algorithms. Typical query volumes in a real-time application hosted by an internet service provider is about 10,000 queries, which emphasizes the need to be able to scale well in that dimension.

Figure 2 shows how the algorithms compare with the PMC solution when scanning with 250 queries in 1, 2, 5, and 10 times the data volume of the downloaded spam examples. As shown, the `nrgrep` and `flex` algorithms run into a much steeper gradient with respect to the dataset size than do the PMC, which is important considering that an internet service provider needs to scan hundreds if not thousands of emails per second.

To summarize, our PMC-based solution scales better than `flex` and `nrgrep` both with respect to query volumes and dataset size. Even though it appears that the `flex` algorithm is the closest competitor in the above benchmarks, it will probably be impractical in production systems due to

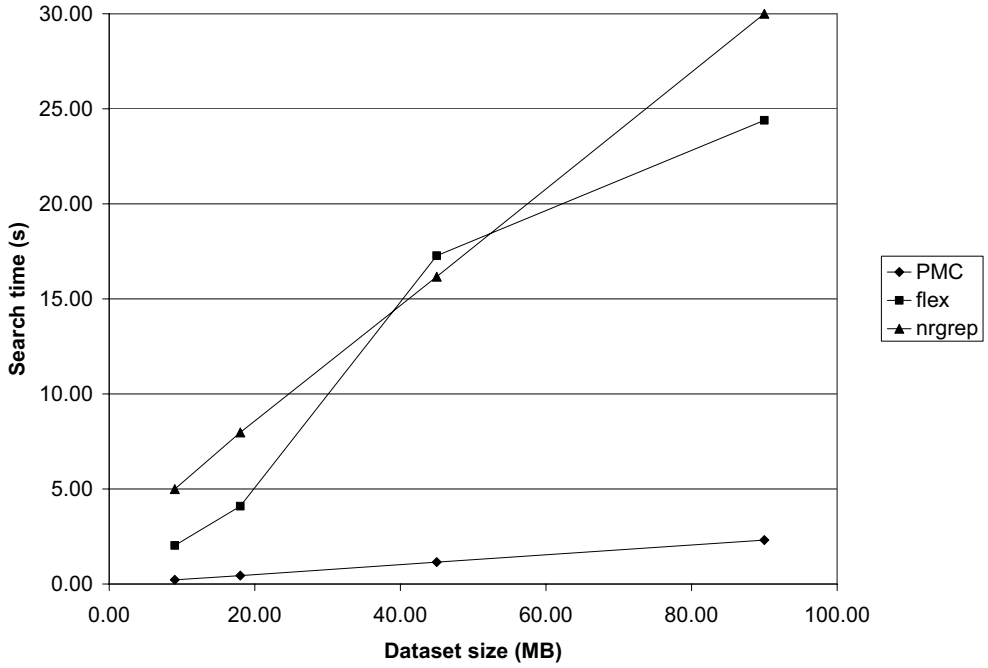

Figure 2. Search time versus dataset size for various solutions.

a lengthy and unpredictable compilation process. In the above benchmarks, we used only patterns that `flex` could handle in order to get a fair evaluation, but we experienced that `flex` broke down completely on several occasions. Consider the following listing, which is the first part of a query that `flex` were not able to handle:

$$(\text{earn}|\text{make}).\{1,20\}[0-9][0-9][0-9]+.\{1,30\}\ldots$$

Here, repetition of wildcards succeeds either of the words "earn" or "make". Note that the "e" in "make" could mean the beginning of "earn" and if a number occurs it could either be a wildcard or one of the three or more numbers that are to follow the wildcards. Thus, the pattern cannot easily be represented by a DFA, as illustrated by the program's performance breakdown. Patterns such as these were removed from the queries that comprised the benchmark set, even though neither the PMC nor the `nrgrep` algorithm experienced performance problems due to these patterns. A spam filtering must therefore either use other algorithms than `flex` or avoid rules that affect its performance at the potential expense of lower spam filtering performance.

5. Discussion

We have introduced a special-purpose search processor into a spam filtering solution to achieve higher performance and better scalability. As shown, the solution scales well in both the query volume and dataset size dimensions. In addition to higher screening performance and better scalability, our solution has the advantage of having negligible configuration processes compared with

the lengthy and unstable compilation of DFAs by `flex`. The `nrgrep` simulates a nondeterministic automaton, but does not generate it explicitly, which is why we are unable to compare its compilation performance to that of `flex`. Note, however, that the scaling performance is slightly worse than that of `flex`, which is not surprising considering that the `flex` numbers were obtained with precompiled scanners.

There is also substantial room for improvements in our solution. First, we can achieve six times the performance of these benchmarks by adding more PCI cards to the workstation. In these tests, we used a single card even though up to six cards can easily be fitted into a standard workstation provided there are enough PCI slots. Furthermore, the PMCs's lookup table (LUT) can be used to map any byte value from the input stream to another [3], which means that several patterns can be collapsed into a single expression. For example, `viagra`, `VIaGrA`, `v|agrA`, and so on can all be matched by `viagra` if uppercase letters are mapped to lowercase and `|` are mapped to `i`. Mapping schemes can be developed for characters that are often used to rewrite trigger words to avoid automatic filters that rely on correct spelling, but to remain easily comprehensible to the human eye.

6. Materials and Methods

6.1. Dataset

We downloaded data from the public spam repository SpamArchive.org. The set comprised 1,705 emails totaling 8,998,303 bytes of data, and a compressed version can be downloaded at `ftp://spamarchive.org/pub/archives/submit/691.r2.gz`. To test the methods's scalability with respect to data volumes, we concatenated two, five, and ten entities of this dataset.

6.2. Algorithms

As described in the main text, we benchmarked the performance of our PMC system against the `nrgrep` and `flex` algorithms. The fast lexical analyzer generator (`flex`) is a free implementation of the well-known `lex` program with some new features (see `http://www.gnu.org/software/flex/` for details, including a manual).

Nondeterministic reverse grep (`nrgrep`) is a member of the `grep` family of search algorithms, and uses bit-parallelism to simulate a nondeterministic suffix automaton. See the excellent book by Navarro and Raffinot for a detailed treatment of DFAs, NFAs, bit-parallelism, and the use of these concepts in regular expression matching [7].

6.3. Hardware

We used a standard workstation with an Intel Pentium 4 2.8 GHz processor with 1,024 MB DDR-SDRAM running the Woody release of Debian Linux (see `http://www.debian.org`). Tests that involve special purpose search processors were run on the same workstation with a single PMC card installed.

References

[1] Doug Burger, James R. Goodman, and Alain Kägi. Memory bandwidth limitations of future microprocessors. In *23rd International Symposium on Computer Architecture (ISCA)*, pages 78–89. IEEE Computer Society, May 1996.

[2] Wu-chun Feng. Green destiny + mpiblast = bioinfomagic. In *ParCo 2003*, pages 653–660, 2003.

[3] Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve Jr., and Olaf René Birkeland. A recursive MISD architecture for pattern matching. *IEEE Trans. on VLSI Syst.*, 12(7):727–734, 2004.

[4] W. Daniel Hillis and Lewis W. Tucker. The CM-5 connection machine: a scalable supercomputer. *Communications of the ACM*, 36:31–40, 1993.

[5] Graham Kirsch. Active memory: Micron's Yukon. In *17th International Parallel and Distributed Processing Symposium (IPDPS)*, page 89. IEEE Computer Society, 2003.

[6] N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In IEEE, editor, *SC2002: From Terabytes to Insight. Proceedings of the IEEE ACM SC 2002 Conference*, 2002.

[7] Gonzalo Navarro and Mathieu Raffinot. *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, Cambridge, UK, 2002.

[8] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent RAM: IRAM. *IEEE Micro*, 17(2):34–44, Apr 1997.

Paper VI

Boosting improves stability and accuracy of genetic programming in biological sequence classification

Boosting improves stability and accuracy of genetic programming in biological sequence classification

Pål Sætrom[1,2], Olaf René Birkeland[1], and Ola Snøve Jr.[1,3]

[1] Interagon AS, Laboratoriesenteret, NO-7006 Trondheim, Norway
{paal.saetrom,olaf.birkeland,ola.snove}@interagon.com
[2] Department of Computer and Information Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
[3] Department of Cancer Research and Molecular Medicine, Faculty of Medicine, Norwegian University of Science and Technology, NO-7489 Trondheim, Norway

Summary. Biological sequence analysis presents interesting challenges for machine learning. With an important problem – the recognition of functional target sites for microRNA molecules – as an example, we show how multiple genetic programming classifiers improve accuracy and stability. Moving from single classifiers to bagging and boosting with crossvalidation and parameter optimization requires more computing power. A special-purpose search processor for fitness evaluation renders boosted genetic programming practical for our purposes.

Key words: Bioinformatics, microRNA, gene prediction, RNAi

1 Introduction

In a typical conversation between a biologist and a computer scientist, the former will often refer to the enormous amount of biological sequence information and slip in a sentence about exponential growth of available data. In reality, however, the current fifty odd billion characters of sequence data will not scare off many programmers, and the relative growth is likely to slow down or be upheld only due to redundant information [3]. Add to that the fact that any real application that you would work on is likely a much smaller problem that allows you to slash most of the data volume and you may start to think that computational biology is a piece of cake.

Unfortunately, biology is so incredibly complex that almost any problem turns out to be challenging. Famous Professor Emeritus of Stanford University, Donald Knuth, once said that he was confident that biology will easily keep scientists busy for 500 years [13]. He is probably not far off.

In this paper, we will focus on the need for robust learning methods in bioinformatics in general, and the applicability of boosted genetic programming in particular. In addition, we will show how we use special-purpose pattern matching hardware to speed up the learning process, which despite limited data volumes becomes necessary when you have to make millions of passes over the same data.

Molecular biology is in many ways digital. DNA codes for RNA that codes for protein, which is a simplistic version of the central dogma for genetic information transfer [6]. Digitalized, we may view this as a two-step process where DNA's letters A,C,G, and T can be translated to U,G,C, and A in RNA's alphabet. From a sequence of RNA letters, protein factories called ribosomes translate triplets of RNA characters to the amino acids that constitute proteins [4]. DNA is a double-stranded helix where the strands base-pair according to Watson-Crick rules, which means A to T and C to G. RNA is preferentially single-stranded, but RNA may also base-pair in an A to U and C to G configuration. Characters that preferentially base-pair to each other are complementary, which means that A is complementary to T in DNA and U in RNA, whereas G is complementary to C. Strands are said to be antiparallel, which refers to the internal configuration of the nucleotides that we view as characters. Sometimes, we need to specify the end we are talking about, and DNA and RNA strands are therefore said to run in 5' to 3' direction (see chapter 1 in [15] for a detailed discussion of nucleic acids and their properties).

Sequence analysis in computational biology usually means to find sequences that are similar to a given template or to find complex patterns that are functionally significant. Our machine learning examples will describe methods for identification of functionally significant patterns in RNA sequences, but the methods are applicable to DNA and amino acid sequence mining as well.

2 Methods

2.1 Genetic programming with string queries

The inspiration for our genetic programming (GP) system came from our ability to quickly evaluate an expression's fitness with the special-purpose search processor [11] that will be described in Sect. 2.3.

Our algorithm is designed to solve two-class classification problems and operate directly on positive and negative string samples without additional encoding steps. The population of solutions consist of syntax trees that represent queries in a formal query language [14]; we use strong typing to ensure that all individuals are legal expressions in the language [18]. Although we base our solution language on a query language that can be evaluated on our search hardware (`http://www.interagon.com/pub/whitepapers/IQL.`

`reference-latest.pdf`), we may restrict expressions to a subset of the full functionality, depending on the application (see Fig. 2 for an example).

In all our experiments, we use strongly typed crossover and mutation [18], and standard reproduction as genetic operators (89%, 1%, and 10%); ramped half-n-half with a maximum tree depth of 7 to initialize populations; and a modified tournament selection (Ø. Grotmol, unpublished) that treats individuals with identical phenotypes as a single individual for the purpose of selecting individuals to a tournament (tournament size 5). The fitness function is [23]

$$\varepsilon(h, S, D) = \sum_{i=1}^{|S|} d_i \cdot |h(x_i) - y_i|, \tag{1}$$

where h is the individual, S is the training set consisting of sequences x and binary labels y, and D is a weight vector that gives the relative importance d of each sequence in the training set. The weights are initialized to $1/(2 \cdot p)$ and $1/(2 \cdot n)$ for the positive and negative sequences, where p and n are the number of positive and negative sequences. This ensures that the naive solutions that predict all positives or all negatives have a fitness of 0.5.

2.2 Boosted genetic programming

Our classifiers are queries that are either present or not in an input string. In our experience, a single binary classifier can not satisfactorily capture the complexity of the applications we have addressed. A way of mending this problem is to build a soft classifier $f = \sum_{t=1}^{T} \alpha_t h_t$, by assigning weights α_t to our so-called hard classifiers h_t. There are several ways to assign the weights to each classifier. In the simplest case, each weight is set to $1/T$, which means that the model corresponds to the average of the hard classifiers. Boosting algorithms attempt to assign the weights iteratively. Our implementation is based on AdaBoost, which puts more effort into learning the difficult parts of the dataset [9]. Since GP is stochastic, two independent runs will generally not produce identical results. To reduce the variance, we construct a model that corresponds to the average of several boosted classifiers, which in addition to the reduced variance can also give higher accuracy [12]. In this work, averaged classifiers consist of ten single classifiers.

Note that AdaBoost works similar to maximum margin classifiers known from statistical machine learning [17]. One example of a maximum margin classifier is the popular support vector machine, which attempts to construct a classifier that maximizes the distance to any example in the training set [5]. As a result, however, our algorithm would be expected to have problems with noise and outliers. We therefore implemented a regularized boosted genetic programming algorithm along the lines of Rätsch [21] and have successfully used it in genetic sequence classifications [23, 24]. As the regularized algorithm has an extra parameter that should be optimized, estimating the regularized algorithm's performance on a particular problem is more complex

and time-consuming than estimating the performance of the standard or averaged AdaBoost. In addition, simple averaging of boosted classifiers often has performance similar to the regularized algorithm [23]. We therefore do not use the regularized algorithm in this work.

2.3 The pattern matching chip

Interagon's pattern matching chip (PMC) is a custom processor for finding patterns in data [11]. Instead of building data structures, the PMC makes a pass through the entire data set on every search. Each chip has a dedicated local memory bank, with a 100 MB/s search bandwidth. Multiple patterns can be evaluated towards the data in parallel due to the MISD architecture.

Each chip has 1024 parallel processing elements (PEs). 16 chips are integrated into a PCI accelerator card along with 2 GB of DRAM. One PC can typically hold up to 6 of these cards, rendering a total of $6 \cdot 16 \cdot 1024 \approx 100000$ parallel PEs in one PC. Operating at 100 MHz, this corresponds to 10^{13} symbol comparisons per second. The aggregated processing bandwidth of one such system is 9.6 GB/s.

The individual symbol comparisons are combined in a programmable binary tree, allowing for any Boolean operators, counting, ordering or adjacency between symbols or subexpressions. The throughput of the PMC is thus not limited by the complexity of the operators within the query, but rather the number of symbols used.

This capability is important when screening short nucleotide patterns built from a grammar rich in operators. As a bonus, the tree structure of PMC query processing is easily manipulated during mutation and crossover steps in GP.

Table 1. Run time of sequence similarity searches on different algorithms and platforms. Run time is measured for screening 3519 individual 25 nucleotide sequences versus a 3 billion nucleotide data base

Algorithm	Platform	Run time
Proprietary	PMC system with 6 cards	15 minutes
Smith-Waterman	PC	44 days
Smith-Waterman	GeneMatcher2 (Paracel)	255 minutes
BLAST	BlastMachine2 (Paracel)	15 hours
Smith-Waterman	DeCypher (TimeLogic)	14 hours
BLAST	Tera-BLAST ($n = 7$, TimeLogic)	3 hours

Other sequence similarity screening methods exists, most notably the BLAST [1] and Smith-Waterman [27] algorithms. The Smith-Waterman algorithm is considered the gold standard for such searches. It uses dynamic programming to build a cost table for the potential alignments of a query

versus the reference data. The different types of individual nucleotide alignments – that is, match, mismatch, insertion or deletion – have different costs. Smith-Waterman requires extensive updating of the cost table for each query symbol processed, and is thus inherently slow (see Table 1). In our experience, insertions and deletions are less relevant when working with short nucleotides, rendering a considerable fraction of the processing in the Smith-Waterman algorithm as unnecessary.

BLAST uses heuristics to speed up the search, by indexing all consecutive n nucleotides in the reference data, with n typically being 11. This index is used as seed points, from which longer alignments are found through similar cost functions as in Smith-Waterman. Consequently, BLAST can not search for patterns unless there are n consecutive exact matches in the query. For queries in the same order of magnitude as n, this implies that BLAST can only find exact matches, without any operators on the character level (see Table 2). BLAST becomes more sensitive for smaller values of n, but also slows down a factor of 4 for each reduction in n. $\text{BLAST}_{n=7}$ is thus 256 times slower than $\text{BLAST}_{n=11}$, and is considered to be the practical lower limit.

Table 2. Capability comparison for short sequence similarity searches using the pattern matching chip (PMC), BLAST and Smith-Waterman. Entries marked with a dash indicates a search not feasible with the specified algorithm

Query	PMC	BLAST ($n = 11$)	BLAST ($n = 7$)	Smith-Waterman
`GGGAAACCCTTTGGGAAACCCT`	•	•	•	•
`GGGAAACCC...GGGAAACCCT`	•	–	•	•
`GGGAAA...TTTGGG...CCCT`	•	–	–	•
{`GGGAAACCCTTTGGGAAACCCT` : p $\geq$ 21}	•	–	•	•
{`GGGAAACCC...GGGAAACCCT` : p $\geq$ 20}	•	–	–	•

3 Results

3.1 Predicting microRNA targets

Most genes – that is, functionally important stretches of DNA – code for proteins. But during the last decade, biologists have become aware that some genes encode RNA that is not translated into a protein. MicroRNAs (miRNAs) – so named because they are only about 22 nucleotides long – constitute one such class of non-protein-coding RNAs. MicroRNAs base-pair with the RNA intermediate of genes before protein translation occurs, and thereby marks them for destruction by designated protein complexes. This provides a way for miRNAs to decrease the protein output from regular genes.

There are two ways that miRNAs can decrease the expression of their targets. First, their sequence may be an almost perfect complement of their target's RNA intermediate, as shown in Fig. 1a. Second, the complementarity may be more fuzzy, but still prevalent in one end of the miRNA, as shown in Fig. 1b. Near-perfect complementarity results in target cleavage [16], whereas partial complementarity prevents the ribosomes from completing protein translation [19]. The latter binding form is more challenging from a computational point of view, as we only have a few examples of valid target sites and have to deduct the rules from these.

Fig. 1a. The near-perfect complementarity between the human mir-196a miRNA and Hoxb8 mRNA results in mRNA degradation [30]

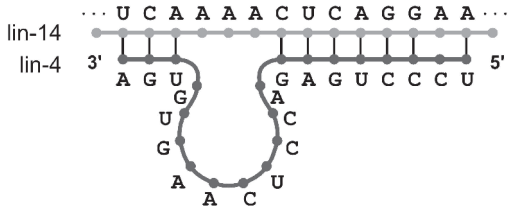

Fig. 1b. The imperfect complementarity between the C. elegans lin-4 miRNA and lin-14 mRNA results in translational repression [29]

Currently, several hundred human miRNAs are known [10] (332 in Release 8.0), but only a limited number of mRNA targets have been verified [26]. Several tools for predicting miRNA target sites have been published (see [26] for an overview), but these tools were developed based on the current imperfect understanding of miRNA target site recognition. Many of the tools may therefore be overly biased towards the authors' preconceptions of an ideal miRNA target site; especially, the importance of overall binding energy between the miRNA and mRNA target site.

Our approach is to use machine learning to create a miRNA target site predictor [22]. Although this approach is not without its biases – especially regarding the data set used to train the predictor – the approach should at least create a predictor that is consistent with currently available data. In addition, we can easily create new predictors as new data becomes available.

We have previously shown that our boosted GP solution creates miRNA target site predictors that are at least as good and better than human created predictors [22]. In the following sections we will study the robustness of our algorithm in terms of the impact of the choice of GP parameters and of the boosting step. We start, however, by defining the query language we will use

to predict miRNA target sites. The genetic programming step will search for candidate solutions in this language.

3.2 A query language for recognizing microRNA target sites

A microRNA target site is characterized by the base-pair interactions between the miRNA and the mRNA; that is, certain nucleotides in the miRNA binds to nucleotides in the mRNA target site. These base-pairing characteristics can be used to search for other similar target sites, by using search queries that matches the base-pair interactions of the target site. To illustrate, the query `CCCAACAACAUGAAACUGC` will find target sites identical to the mir-196a site in Hoxb8 (Fig. 1a) and the query `UCA...CUCAGG.A`, where the dot (`.`) matches any character, will find target sites similar to the lin-4 site in lin-14 (Fig. 1b).

To make this basic idea applicable for general miRNA target site prediction, we extend the query language as follows. First, as we want our queries to represent the characteristics of miRNA target sites in general and be independent of the particular miRNAs, we use the positions of the miRNA nucleotides instead of the nucleotides themselves. Thus, we write the general query for the lin-4 site in Fig. 1b as $p_{21}p_{20}p_{19}\ldots p_8p_7p_6p_5p_4p_3\,.\,p_1$, where p_i represents the complement of the miRNA nucleotide at position i counted from the miRNA's 5' end. We can then search for the target sites of a particular miRNA by replacing the p_i's with the miRNA's corresponding nucleotide.

Second, to introduce more flexibility when searching for potential sites, we allow a certain number of mismatches between our query Q and potential target sites. We write this as $\{Q : \mathrm{p} \geq x\}$, where x is the number of nucleotides that have to match in the original query. Note that this is identical to requiring a Hamming distance of at most $|Q|-x$ between the query and target sequences, where $|Q|$ is the number of nucleotides in the query.

Third, we require that the positions in the query are continual and in sequence. Thus, $p_6p_5p_4p_3$ is a valid query, but $p_6p_4p_2$ is not. Figure 2 formally defines our query language. The grammar shows the legal production rules in the language with alternatives represented as separate productions and nonterminals represented by uppercase letters.

The terminal p_i represents a position in the miRNA sequence, and, as the semantic productions in Fig. 2 show, the exact position depends on the terminal's position in the parse tree. More specifically, the first terminal encountered in a walk of the parse tree defines the query's start position in the miRNA sequence; the rest of the terminals have their positions derived from the initial terminal to ensure that the final query defines a continual sequence of miRNA positions. The *in* and *index* node attributes in the semantic rules handle this.

A query matches a sequence if $Q.hit$ is true. $match(p_k)$ returns 1 if the character in the position indicated by p_k is identical to the character it is compared with. Note that we use two sets of terminals A and N to define the cutoff value on the number of mismatches. The cutoff is initially zero and each

Production	Semantic Rule
(1) $Q \rightarrow \{C : \mathrm{p} \geq x\}$	$C.in := \text{-}1$
	$x := C.cut$
	$Q.hit := C.count \geq x$
(2) $C \rightarrow C_1 C_2$	$C_1.in := C.in$
	$C_2.in := C_1.index$
	$C.index := C_2.index$
	$C.count := C_1.count + C_2.count$
	$C.cut := C_1.cut + C_2.cut$
(3) $C \rightarrow A$	$A.in := C.in$
	$C.index := A.index$
	$C.count := A.count$
	$C.cut := A.cut$
(4) $C \rightarrow N$	$N.in := C.in$
	$C.index := N.index$
	$C.count := N.count$
	$C.cut := N.cut$
(5) $A \rightarrow p_i, i \in \{1, \ldots, 21\}$	$k := A.in = -1\,?\,i : A.in - 1$
	$A.cut := 1$
	$A.count := match(p_k)$
	$A.index := k$
(6) $N \rightarrow p_i, i \in \{1, \ldots, 21\}$	$k := N.in = -1\,?\,i : N.in - 1$
	$N.cut := 0$
	$N.count := match(p_k)$
	$N.index := k$

Fig. 2. The grammar and semantics of the pattern language used in our microRNA target prediction experiments. The grammar and semantics are explained in the main text

terminal from A increases the cutoff by one. Consequently, queries where all terminals are from N have a cutoff of zero, and queries where all terminals are from A have a cutoff equal to the number of terminals. This ensures that the cutoff always has a meaningful value.

3.3 Genetic programming produces good but unstable classifiers

The choice of parameters in a GP run often has a major impact on the quality of the results [7, 8]. We therefore wanted to study how different input parameters affected the performance of the target site queries created by the GP step in our machine learning system. To do this, we ran several experiments where we varied the population size and the number of generations in the GP run. We used the same dataset as in [22], which consisted of 36 experimentally verified miRNA target sites for 4 miRNAs and 3000 random negative sites, and used leave-one-miRNA-out cross-validation to train and test classifiers.

As Fig. 3a shows, increasing the number of generations and the population size resulted in GP finding queries that had an increasingly higher per-

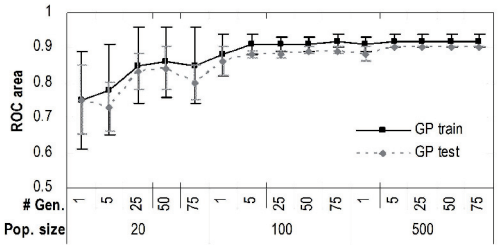

Fig. 3a. The ROC-score of the best-of-run individual increases with increasing population size and number of generations in both the training and test sets. The graph shows the ROC-score average and standard deviation of 10 independent GP runs on the query language in Fig. 2

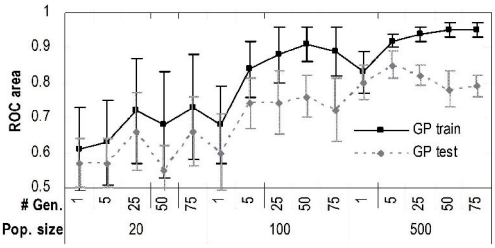

Fig. 3b. GP overfits a more complex query language. The graph shows the ROC-score average and standard deviation of 10 independent GP runs on the query language used in [22]

formance in the training set. This was expected, as increasing the population size and the number of generations increases GP's ability to find better candidate solutions [14, 8]. Good performance in the training set does, however, not guarantee good performance on unseen data [28]; this is especially the case when the training set is small or contains noise. In this case it is both, as the positive set only consists of 36 experimentally verified target sites and the random negatives likely contain sites that closely resemble the verified sites. Nevertheless, the target site queries produced by GP show no sign of overfitting, as their performance in the test set closely resemble their performance in the training set.

What is more, the graph indicates that the solutions produced with the largest population size run for more than five generations are the best possible with our solution language. This is because neither the performance in the training set nor in the test set improves with increasing number of generations, and because there is no variance in the test scores. Manual inspection of the results confirmed the convergence: all the 30 runs on the 3 largest training sets produced the query $\{p_8p_7p_6p_5p_4p_3p_2 : \mathrm{p} \geq 6\}$; the 10 runs on the smallest training set with the let-7 sites excluded produced the query $\{p_8p_7p_6p_5p_4p_3p_2p_1 : \mathrm{p} \geq 6\}$ or equivalents. These results indicate that base-pairs between nucleotides 2-8 are most important for miRNA target site recognition, but also that some mismatches are tolerated.

Sætrom et al. defined a query language that used variable length wildcards to join the sequence motif of Fig. 2 and an unordered motif [22]. This is a query language that can model more complex relationships in the data, and we wanted to determine if GP could evolve even better solutions than the ones based on our initial query language. As Fig. 3b shows, however, genetic programming could not. Even though the best solutions have a higher performance in the training set, their performance in the test set is much

worse. Thus, even though GP now finds solutions that better describe the training data, these solutions are overfitted and do not generalize to unseen data.

From statistical learning theory, we know that overfitting is closely related to the capacity of the set of functions from which we draw our solutions [28]. The above results illustrate this relationship. The above analysis also illustrates the impact that parameter choices have on the results produced by genetic programming. First, the average ROC-score on the test set varies greatly depending on the parameter choice (standard deviations of 0.055 and 0.099 for the ROC-score averages in Figs. 3a and 3b). Second, even though genetic programming can in some cases produce optimal results simply by choosing a large population size and running for many generations, this is a sure recipe for overfitting in the general case. Consequently, finding good solutions requires parameter optimization, but on problems where slight changes in parameters give large changes in results, this is inherently difficult [20]. Furthermore, parameter optimization is an extra training step that risks overfitting, which means that the parameter optimization must be kept completely independent of the test set. Otherwise one will get biased estimates of the classifier's performance in unseen data [25].

3.4 Averaging significantly improves predictor accuracy and stability

The previous section illustrated a major problem with using genetic programming for pattern mining: the solutions produced by genetic programming are brittle, as slight changes in input parameters can give large changes in solution quality. This is illustrated by the large variances in the average ROC-scores for the different parameter settings and in the ROC-scores for some of the individual parameter settings. This latter observation does, however, hint at a possible way to improve the GP results, as diverse and accurate classifiers can be combined to produce more accurate and less variable classifiers [12].

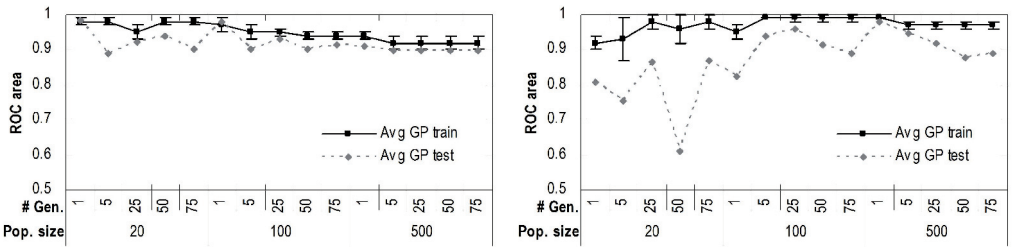

Fig. 4. Averaging GP classifiers improves performance and reduces parameter variance. The graphs show the ROC-scores in the training and test sets for the classifiers created by averaging the expressions from Fig. 3a (left) and Fig. 3b (right)

Figure 4 shows the result of averaging the solutions from the GP runs in Sect. 3.3. A comparison with Figs. 3a and 3b shows the improved performance of the average classifiers compared to the individual classifiers. For all parameter settings where GP produce varying solutions, the average classifiers have a higher performance than the individual classifiers. The increased performance was significant ($p = 4 \cdot 10^{-3}$ and $p = 3 \cdot 10^{-8}$ with paired Student's t-tests on the simple and complex grammar) and the ROC-scores also varied less across the different parameter settings for the simple grammar ($p = 0.04$ with an F-test), which indicates that the average classifiers are more robust than the single classifiers. In fact, the average classifiers use the stochastic nature of GP to improve the classifier performance.

3.5 Boosting further improves predictor accuracy and stability

To try to improve our classifiers further, we combined our genetic programming system with the AdaBoost algorithm [9]. Fig 5 shows the performance of the resulting weighted sequence motif classifiers for different parameter settings of GP.

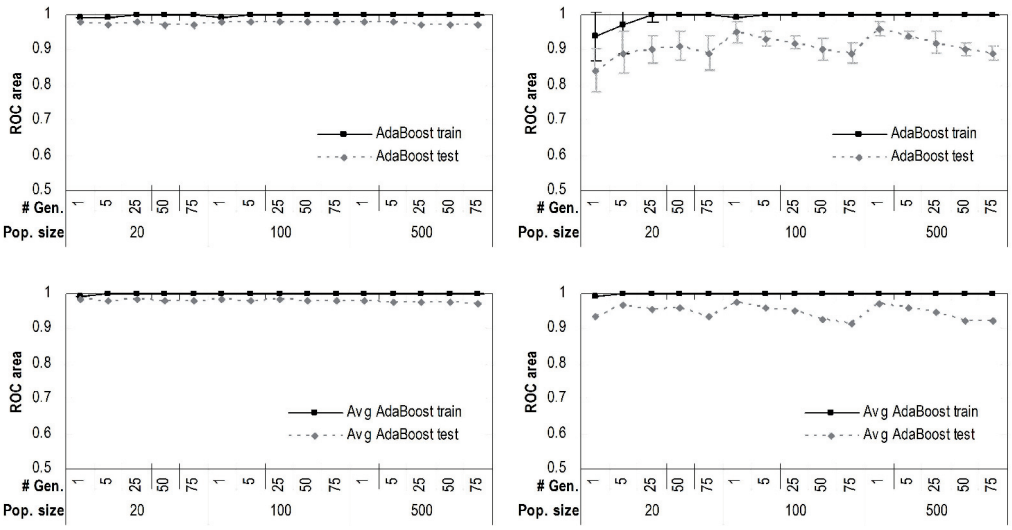

Fig. 5. Boosting further improves performance and reduces parameter variance. The graphs show the ROC-scores in the training and test sets for the classifiers created by boosting the expressions from Fig. 3a (top left) and Fig. 3b (top right) for 25 iterations, and by averaging the boosted classifiers (bottom left) and (bottom right)

Comparing with the single and averaged GP results (Figs. 3a and 4), the boosted expressions from Fig. 2 have both increased performance and parameter stability ($p = 7 \cdot 10^{-7}$ and $p = 9 \cdot 10^{-7}$ with paired Student's t-tests

and $p = 9 \cdot 10^{-12}$ and $p = 6 \cdot 10^{-8}$ with F-tests). The boosted versions of the more complex expressions from [22] also had increased performance and parameter stability, but the differences were not as large as for the simpler grammar ($p = 6 \cdot 10^{-7}$ and $p = 0.1$ with paired Student's t-tests and $p = 6 \cdot 10^{-5}$ and $p = 1 \cdot 10^{-4}$ with F-tests). Averaging the boosted classifiers further improved the performance and parameter stability for both grammars (Figs. 5 bottom left and right; $p = 0.004$ and $p = 1 \cdot 10^{-5}$ with paired Student's t-tests and $p = 0.1$ and $p = 0.1$ with F-tests).

Running the boosting algorithm for additional iterations had a minor impact on the classifiers' performance. We repeated the above experiment with both classifier languages for the different GP parameters, but increased the number of boosting iterations to 75. The only difference between these results and the results from the 25 iteration boosting was a small, but significant decrease in the average performance across the different parameter settings for the averaged boosted classifiers based on the complex expressions (from 0.947 to 0.924; p-value $= 6 \cdot 10^{-6}$ with a paired Student's t-test).

The AdaBoost algorithm creates a classifier that maximizes the minimum margin in the training set [2], and theoretically, AdaBoost converges towards this maximum margin classifier exponentially with the number of boosting iterations [17]. Thus, the first boosting steps have the highest impact on the final boosted classifier's performance, and increasing the number of boosting iterations have less and less impact on the performance of the final classifier.

In summary, boosting the classifiers created by GP significantly improved the classifiers' performance and significantly improved the stability of the algorithm with respect to the choice of GP parameters. In particular, the average performance of the boosted classifiers was, for all the parameter settings, higher than the best performance of any of the single classifiers created by GP. Thus, we do not get the best performance by ensuring that GP finds the optimal solution in the training set; we get better results by running many GP runs on small populations for few generations and combining the suboptimal solutions found in these runs into a single robust classifier.

3.6 The solution language has a major impact on classifier performance

As the previous sections showed, both the solution languages we investigated could accurately separate real miRNA target sites from random sequences. These solution languages were based on the hamming distance filtering function, which our special purpose hardware evaluates fast and efficiently. We used this hardware in our experiments to accelerate the fitness evaluation of the candidate expressions and thereby reduced the total runtime of the experiments. Simpler solution languages that use exact matching can, however, be evaluated much faster in software than can the approximate Hamming distance-based expressions. We therefore wanted to determine whether we

could create accurate miRNA target predictors based on a solution language that only allowed exact matches between miRNAs and potential target sites.

To create a solution language that only allowed exact matches, we removed productions (4) and (6) from the grammar in Fig. 2. Consequently, a query now only returns a hit when all the positions in the query report a match. Using this solution language, we repeated the analyzes of the miRNA target site data. Figure 6 summarizes the results.

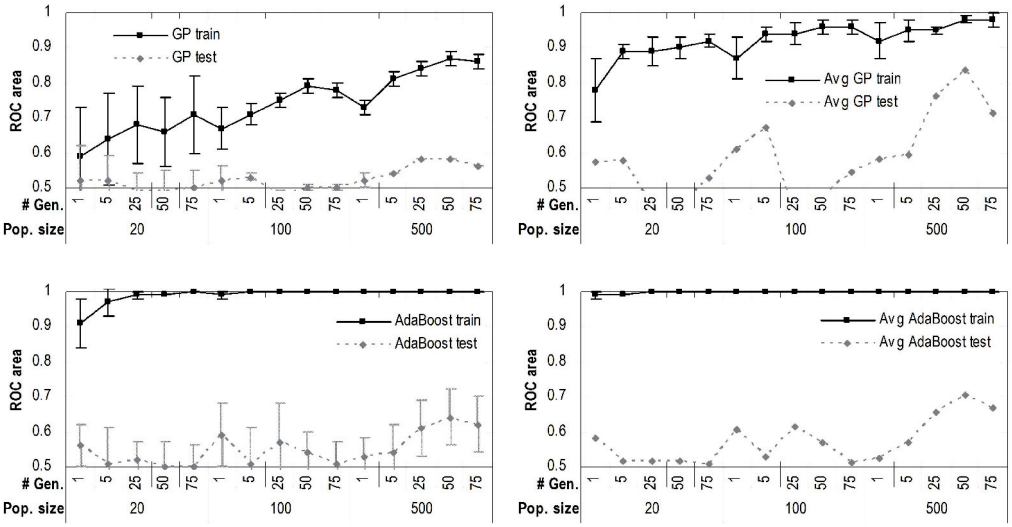

Fig. 6. Classifiers that only allow exact matches between miRNAs and target sites have a much lower accuracy than the classifiers based on approximate matches (Figs. 3a, 3b, 4, and 5). The graphs show the ROC-scores in the training and test sets for single GP (top left), averaged GP (top right), GP boosted 25 iterations (bottom left), and averaged boosted GP (bottom right) classifiers based on a solution language that only allowed exact matches

A comparison with the previous results showed that all the "approximate match" classifiers were significantly more accurate than the "exact match" classifiers (all p-values $< 1 \cdot 10^{-6}$ with paired Student's t-tests). The "exact match" classifiers did, however, have relatively high accuracies in their training sets; in other words, these classifiers were severely overfitted to their training sets. And even though some of the classifiers had a relatively high performance in the test set – especially, the averaged GP classifiers – this best accuracy is still much lower than the best accuracy for the "approximate match" classifiers. This high accuracy may also be an artefact of our multiple testing.

These results show that the solution language that only allows exact matches is inappropriate for predicting miRNA target sites. This was expected as miRNA binding is inexact in itself. The results also illustrate the impact the solution language has on the classifier performance.

3.7 The superior performance of ensemble classifiers generalize to other sequence classification problems

The previous sections have shown that when predicting microRNA target sites, combining several single GP classifiers into an ensemble classifier gives classifiers that are significantly more accurate than the single GP classifiers. To show that this property also generalizes to other problems, we compared the performance of the single, averaged, boosted, and averaged boosted GP classifiers on two other sequence classification problems. These were (i) predicting whether 50 nucleotide long sequences are parts of non-coding RNA genes [24], and (ii) predicting the knockdown efficacy of 19 nucleotide long short interfering RNAs [23].

We used the same test setup as in the previous experiments, except that we excluded 50 generations from the set of parameters tested. We used the solution language from [23] in both experiments. This language resembles the one in Fig. 2, except that the terminals are the four nucleotides and not a template position in the sequence. The language also allows alternatives at each position; see [23] for complete details.

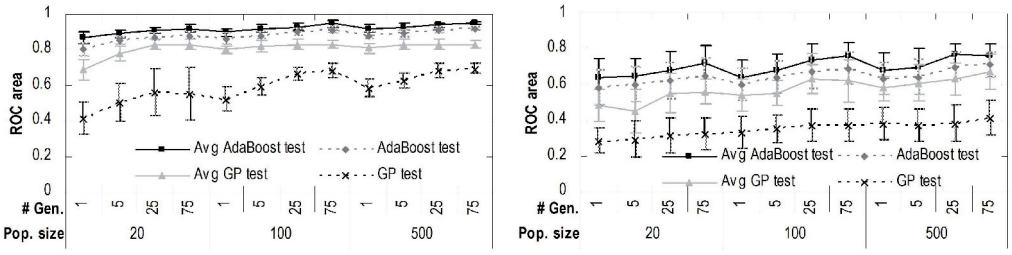

Fig. 7. Ensemble classifiers are more accurate than single genetic programming classifiers. The graphs show the test set ROC-scores for the non-coding RNA gene prediction (left) and short interfering RNA efficacy prediction (right) problems

As Fig. 7 shows, the ensemble classifiers are better than the single classifiers for all parameter settings, and the averaged boosted classifiers have the highest overall performance. The differences are significant for both problems. More specifically, all p-values in the paired Student's t-tests that compare the performance of the single GP classifiers to the ensemble classifiers are $< 3 \cdot 10^{-8}$. Similarly, all p-values for the averaged boosted classifiers are $< 4 \cdot 10^{-8}$.

4 Discussion

We have presented our boosted genetic programming (GP) system, and shown how we can use a special-purpose search processor for fitness evaluation by

operating on parse trees that represent search queries. Using a motivating example from molecular biology – recognition of viable target sites for small RNA molecules called microRNAs – we have demonstrated how the system's performance varies with different setups.

Importantly, GP can produce classifiers with satisfactory performance for target recognition. Populations with more than 100 individuals that are run for 50 or more generations seem to be adequate for good performance. The solution language is very important, and increasing the algorithm's capacity by allowing more complex queries is a recipe for overfitting.

Unfortunately, the stochastic property of GP results in a great difference between best-of-run classifier performances. Using the average of several classifiers significantly improves the stability of our algorithm, but accuracy is also improved in the process. When we used a boosted GP algorithm on our problem, we saw an impressive stability with negligible variance between runs. Also, the boosted classifiers have higher accuracy and increased generalization performance, as shown by their high and stable performance on the test set.

References

1. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
2. P. Bartlett, Y. Freund, W. S. Lee, and R. E. Schapire. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.
3. D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. GenBank. *Nucleic Acids Research*, 33(DB):D34–D38, 2005.
4. S. Brenner, F. Jacob, and M. Meselson. An unstable intermediate carrying information from genes to ribosomes for protein synthesis. *Nature*, 190:576–581, 1961.
5. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.
6. F. H. C. Crick. The biological replication of macromolecules. *Symposia of the Society for Experimental Biology*, 12:138–163, 1958.
7. Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transations on Evolutionary Computation*, 3(2):124–141, July 1999.
8. Robert Feldt and Peter Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 271–282, Edinburgh, 15-16 April 2000. Springer-Verlag.
9. Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, Aug 1997.
10. S. Griffiths-Jones. The microRNA registry. *Nucleic Acids Research*, 32(90001):D109–111, 2004.

11. A. Halaas, B. Svingen, M. Nedland, P. Sætrom, O. Snøve Jr., and O. R. Birkeland. A recursive MISD architecture for pattern matching. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(7):727–734, 2004.
12. L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.
13. D. E. Knuth. All questions answered. *Notices of the AMS*, 49(3):318–324, 2002.
14. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
15. B. Lewin. *Genes VII*. Oxford University Press, Oxford, UK, 2000.
16. J. Martinez and T. Tuschl. RISC is a 5' phosphomonoester-producing RNA endonuclease. *Genes & development*, 18(9):975–980, 2004.
17. R. Meir and G. Rätsch. An introduction to boosting and leveraging. In S. Mendelson and A. Smola, editors, *Advanced Lectures on Machine Learning*, volume 2600, pages 118–183. Springer-Verlag, 2003.
18. David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
19. C. P. Petersen, M.-E. Bordeleau, J. Pelletier, and P. A. Sharp. Short RNAs repress translation after initiation in mammalian cells. *Molecular cell*, 21(4):533–542, 2006.
20. L. Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.
21. G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, Mar 2001.
22. O. Sætrom, O. Snøve Jr., and P. Sætrom. Weighted sequence motifs as an improved seeding step in microRNA target prediction algorithms. *RNA*, 11(7):995–1003, 2005.
23. P. Sætrom. Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063, 2004.
24. P. Sætrom, R. Sneve, K. I. Kristiansen, O. Snøve Jr., T. Grünfeld, T. Rognes, and E. Seeberg. Predicting non-coding RNA genes in *Escherichia coli* with boosted genetic programming. *Nucleic Acids Research*, 33(10):3263–3270, 2005.
25. S. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.
26. P. Sethupathy, B. Corda, and A. G. Hatzigeorgiou. TarBase: a comprehensive database of experimentally supported anima l microRNA targets. *RNA*, 12(2):192–197, 2006.
27. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):403–410, 1981.
28. V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, NY, USA, 1998.
29. B. Wightman, I. Ha, and G. Ruvkun. Posttranscriptional regulation of the heterochronic gene *lin-14* by *lin-4* mediates temporal pattern formation in *C. elegans*. *Cell*, 75(5):855–862, 1993.
30. S. Yekta, I. Shih, and D. P. Bartel. MicroRNA-directed cleavage of *HOXB8* mRNA. *Science*, 304(5670):594–596, 2004.

Paper VII

The Pattern Matching Chip

The Pattern Matching Chip

Olaf René Birkeland and Ola Snøve Jr*

3 September, 2002

Contents

1 Introduction **2**

2 Simple Pattern Matching **2**
2.1 The Basic Idea 2
2.2 The Data Distribution Tree 3
2.3 The Processing Elements 4
2.4 The Result Gathering Tree 5
2.5 Example Queries with Configurations 6

3 Advanced Pattern Matching **9**
3.1 Forward Sequence Control 9
3.2 Backward Sequence Control 10
3.3 Repeating Patterns 10
3.4 Skipping Patterns 13
3.5 Sequence Control Limitations 14

4 Managing Documents and Hits **15**
4.1 The Document Manager 15
4.2 The Hit Managers 16

A Result Gathering Tree Details **17**
A.1 Alphabetical and Numerical Comparisons 17
A.2 Implementing Boolean Functions 18

References **20**

*Interagon AS, Medisinsk teknisk senter, NO-7489 Trondheim, Norway

1 Introduction

The Pattern Matching Chip (PMC) is an Application Specific Integrated Circuit (ASIC), capable of searching for advanced patterns in arbitrary data at a constant high speed. The PMC is based on breakthroughs made by researchers at the Norwegian University of Science and Technology (NTNU), who have devoted more than 15 years into developing ASICs for approximate searching. With a clock frequency of 100 MHz, the PMC is able to search with up to 64 distinct queries on 100 MB of data per second.

This document serves as an introduction to the PMC. The reader familiar with the principles introduced here, should cf. [1, 4, 5] for details on the PMC design and architecture.

2 Simple Pattern Matching

This section starts with an introduction to the main ideas behind the PMC. Furthermore, a description of the major parts of the circuit is presented. To enhance the readers understanding, a number of example queries with corresponding PMC configurations are included towards the end of this section.

2.1 The Basic Idea

We should think of the overall operation of the PMC as a stream of data flowing through the chip. The data is distributed to 1024 *processing elements* (PEs) via a binary *data distribution tree*. The results from the comparisons in the PEs are then processed in a *result gathering tree*.

We illustrate the basic workings of the PMC with two simple queries, namely `x|y` and `xy`. That is, either an `x` or a `y` in the former, and an `x` directly followed by a `y` in the latter. Consider figure 1 where the data flows and the PE configurations are illustrated for these queries. If one asks for an `x` or a `y` in a data stream, a natural approach would be to configure one PE to report a match if an `x` occurs, and the other to match a `y`. Then, have them receive data in parallel and the result gathering tree node collecting the results to report a match if either the first, the second or both PEs have a match. That is, report a match if the sum of the results from the PEs

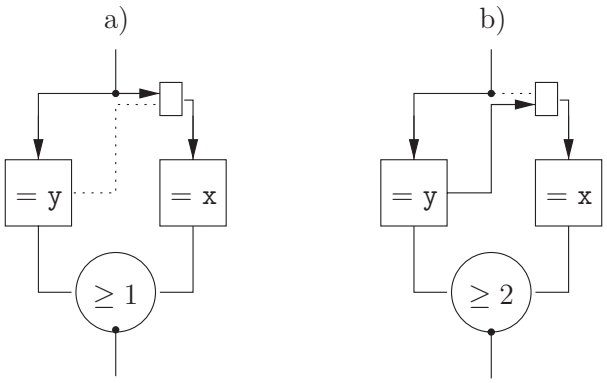

Figure 1: Data flow, pattern matching and result gathering for queries a) `x|y` and b) `xy`.

is greater or equal to one. This is illustrated in figure 1 a). Note that the implementation of parallel or sequential data distribution is simply a $2:1$ multiplexer deciding whether to pass data received from above or from the left neighbor PE, illustrated in the figure by dotting the non-preferred line in the respective cases. Figure 1 b) shows how sequential data distribution is used to match an `x` directly followed by a `y`. If the pattern `xy` were to occur in the data stream, both PEs would report a match, and the result gathering tree node should report a match only if this is the case, i.e. if the sum of the results from the PEs equals two. Remark that matching the latter pattern requires two clock cycles as the `x` first has to be shifted into the left and then to the right PE. As the `x` is directly followed by a `y` in the data stream, the situation will become as shown in the figure after the second shift.

These two queries illustrate how the PMC can utilize parallel and sequential data distribution, and use the boolean `OR` and `AND` operators in the result gathering tree to obtain the desired result. As the basic idea should be clear to the reader, we proceed with a more detailed presentation of the processing elements and the trees.

2.2 The Data Distribution Tree

The processing elements are leaf nodes in a binary data distribution tree, where each internal node can decide whether to pass data to its two chil-

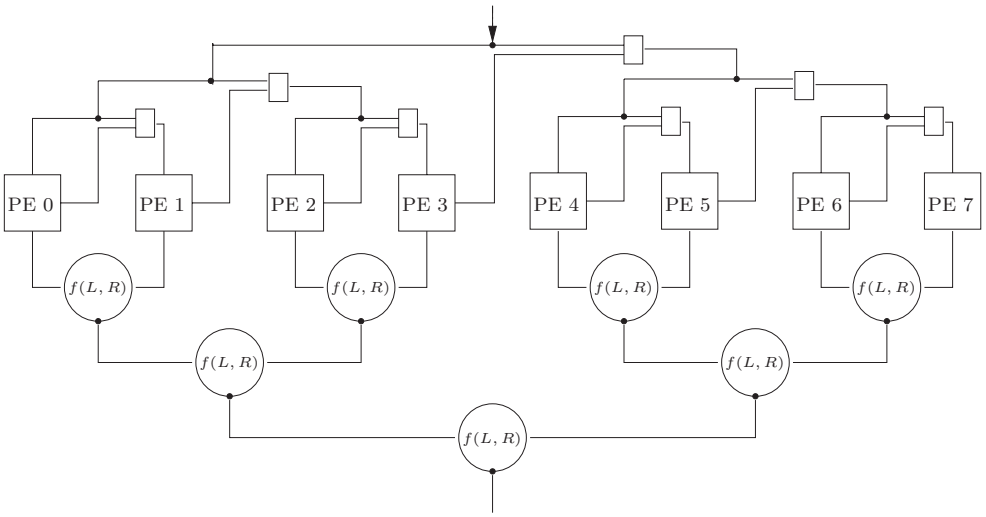

Figure 2: A data distribution tree with eight leaf nodes (PEs), and the corresponding result gathering tree with function $f(L, R)$ calculated from the above left (L) and right (R) results.

dren in parallel or make the right child receive the data sequentially from the rightmost leaf node (PE) in the left sibling tree. Figure 2 shows an implementation of such a data distribution tree with eight PEs (the result gathering tree will be described in section 2.4). This tree is just a direct expansion of the single node tree with two PEs shown in section 2.1. In the actual ASIC implementation there are, as mentioned before, 1024 PEs.

2.3 The Processing Elements

The line of 1024 PEs represent the pattern matching part of the chip, and will sometimes be referred to as level zero of both the data distribution and result gathering tree. Each PE is configured with a reference value, i.e. a byte which is assumed to be a character. It can be configured to perform one out of four possible comparisons:

(i) Check if the reference value, `x`, is less than or equal to the character, `a`, it is compared to, i.e. indicate match if `x` $\leq$ `a`.

(ii) Check if the reference value, `x`, is greater than or equal to the character, `a`, it is compared to, i.e. indicate match if $\texttt{x} \geq \texttt{a}$.

(iii) Check if the reference value, `x`, is not equal to the character, `a`, it is compared to, i.e. indicate match if $\texttt{x} \neq \texttt{a}$.

(iv) Check if the reference value, `x`, is equal to the character, `a`, it is compared to, i.e. indicate match if $\texttt{x} = \texttt{a}$.

Every PE has the possibility of reporting hits one, two, three or infinitely many clock cycles for each time it has an actual match, i.e. the user can specify a latency of $0, 1, 2$ or ∞. Moreover, each PE can be disabled, meaning that it does not perform any operation on its received data.

2.4 The Result Gathering Tree

Each node in the result gathering tree uses the result from the two nodes directly above it to perform its destined task. Nodes at level one, i.e. directly below the line of PEs, uses the output from the two PEs directly above it. We will refer to result from the node above and to the left as L, and the result from its right neighbor as R. Figure 2 shows the result gathering tree below a line of eight PEs, and the operation is denoted $f(L, R)$. The results of the operation can be either boolean or integers, and each node can be configured to perform one out of eight possible operations:

(i) Alphabetical/numerical comparison (cf. appendix A.1).

(ii) Check if the result from the left branch is equal to the result from the right branch, i.e. indicate a match if $L = R$.

(iii) Check if the result from the left branch is greater than the result from the right branch, i.e. indicate a match if $L > R$.

(iv) Check if the result from the left branch is greater than or equal to the result from the right branch, i.e. indicate a match if $L \geq R$.

(v) Summarize the result from the left and right branches, i.e. pass the value $L + R$ to the next level.

(vi) Check if the sum of the results from the left and right branches is greater than or equal to some predefined value, $c \in \{0, 1, \ldots, N\}$, i.e. indicate a match if $L + R \geq c$.

(vii) Check if the sum of the results from the left and right branches is less than or equal to some predefined value, $c \in \{0, 1, \ldots, N\}$, i.e. indicate a match if $L + R \leq c$.

(viii) Check if the sum of the results from the left and right branches is equal to some predefined value, $c \in \{0, 1, \ldots, N\}$, i.e. indicate a match if $L + R = c$.

Note that the maximum accumulated sum for a node in the result gathering tree is $N = 2^{\text{level}}$. Hence specifying a positive integer $c > N$ makes no sense, and is not permitted in the PMC. By using the above operations it is possible to perform all boolean operations (cf. appendix A.2).

As for the PEs, the result gathering nodes can be given a latency value to maintain hits for a certain number of clock cycles. There is a limited number of bits available for specifying the latency value for each node. The maximum number of bits is four on level one, eight on levels two and three, and sixteen on higher levels. If the four, eight or sixteen respective bits are all set, the node is given infinite latency.

2.5 Example Queries with Configurations

The PMC comes with two query languages: The PMC Specification Language (PSL) [3], which is a low level chip specific query language, and the Interagon Query Language (IQL) [2], which is a high level query language similar to regular expressions. This section contains a few example queries with matching PMC configurations. Although the queries will be expressed in IQL, they will be explained in plain English as well, so the reader does not have to be familiar with IQL beforehand.

Example 1 (Range of characters). By combining the functionality of the PEs, we can easily configure the PMC to match any character within a specific range. For consider the query `[b-f]`, that is a `b`, an `f` or any character between them. Figure 3 a) shows an illustration on how to obtain the desired result: One PE matches every character less than or equal to `f`, while the other one matches every character greater than or equal to `b`. With parallel data distribution, and a result gathering node that reports a hit if and only if both PEs report a hit, we realize that the PMC will produce the right output.

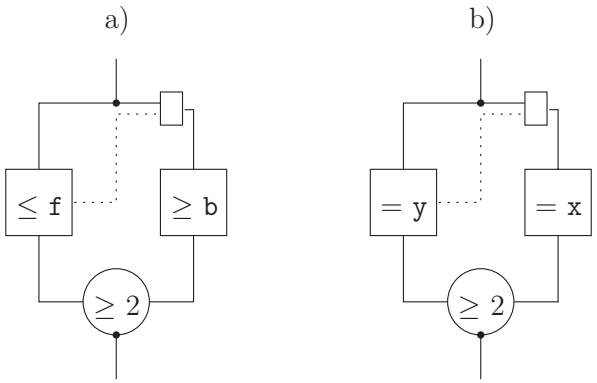

Figure 3: Illustration of PMC configuration matching the queries a) `[b-f]`, i.e. either a `b`, an `f` or any character between them, and b) `x NEAR y`. Remark that latency is required for the PEs in b).

Example 2 (The `NEAR` operator). The IQL has a `NEAR` operator, which makes it possible to inquire the existence of a sequence in the data stream where one expression occurs in the neighborhood of another. Figure 3 b) shows how we can make the PMC match the query `x NEAR y`, that is an `x` appearing close to a `y` in the data stream. By having one PE match an `x` and the other one a `y`, and use latency for both PEs we obtain the desired result if we have parallel data distribution and an `AND` configuration for the result node (sum of left and right branch greater than or equal to 2, as indicated by ≥ 2 in the figure). The statement "close" is a matter of definition. In the IQL, the user may specify what this should mean in terms of number of bytes between the expressions. In our example "close" have to mean one, two or infinitely many (!) clock cycles, because that is the choices we have with respect to latency at the PE level (cf. section 2.3). However, we could easily extend the height of our tree to propagate the latency to higher levels where we could configure a larger latency value.

Example 3 (The `n-of-m` modifier). A useful feature of the IQL is the *pattern n-of-m* modifier. This modifier can be used to search for patterns in the data stream which are, if not entirely correct, at least accurate to some extent. A simple example is the query {`mRNA:p>=3`}, which is an IQL

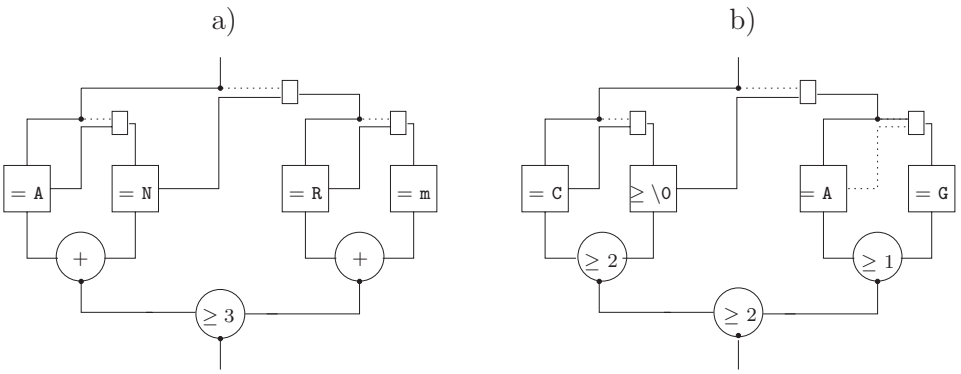

Figure 4: Illustration of PMC configuration for the queries a) {mRNA:p>=3}, that is match all strings of length four that match at least three out of four characters in mRNA, and b) (G|A).C, i.e. a G or an A followed by a wild card and a C

expression demanding that at least three out of the four characters in the string mRNA have to match. Figure 4 shows an appropriate PMC configuration. Since we are to match a string, we want every node in the distribution tree to have sequential data distribution. The PE configuration then follows intuitively, and the desired result is obtained if we summarize the results from the PEs at level one, and at level two checks if the sum of the results from level one is greater than or equal to three. The chip will now match the string mRNA, as well as tRNA, mDNA, mRSA etc.

Example 4 (Use of wild cards). Probably indispensable when performing any type of search is the ability to put wild cards into the query, i.e. we allow any character to match. To illustrate this, consider the query (G|A).C, that is either a G or an A, followed by any character, directly followed by a C. Figure 4 b) illustrates a tree configuration for this query. The second leftmost PE is configured to match any character greater than or equal to the "empty" character (ASCII 0), which means that it will act as a wild card. We have serial data distribution at level two, and in the left subtree of level one, whereas the right subtree has parallel data distribution to enable matching of either A or G in the same position. We use boolean AND to account for the .C part of the query, and the OR operator to ensure we have either of the

two specified characters, `G` or `A`, in place. Finally, we use the `AND` operator to establish if the entire query match. Such a configuration will match strings like `GGC`, `ACC`, `ABC`, `GCC`, and so on.

3 Advanced Pattern Matching

With the reader being familiar with the main ideas behind the PMC, i.e. the binary distribution and result gathering trees, and the processing elements, we move on to introduce the concept of *sequence control*. Sequence control is used, among other things to enable the PMC to match repeating or skipping patterns.

3.1 Forward Sequence Control

With *forward sequence control* we can specify that a PE or a node in the result gathering tree should not report a match unless the node in front of it (in the data flow direction) also reports a match. This is best illustrated with an example.

Example 5 (The `BEFORE` statement). With the use of forward sequence control one is able to specify that a pattern should appear before another pattern in the data stream. The IQL has its own `BEFORE` operator, thus permitting queries like `REM BEFORE (cd|lp)`. In other words, the string `REM` has to appear before either `cd` or `lp` in the data stream. Figure 5 shows the applicable tree configuration. The distribution tree configuration should be clear from previous examples. This is the first time we encounter a pattern that does not completely fill all PEs in a subtree. The simple solution is to disable the unused PEs. In this case we need to disable a single PE, as illustrated in figure 5 by a "×" in this PE. Another feature, not previously mentioned, is the possibility of specifying that a result should be blocked and not passed on to the rest of the result tree. The right result node at level two will report a match if `REM` exist in the data stream. This node should be configured with infinite latency to ensure that the result node on its left can report a hit whenever either `cd` or `lp` is found afterwards. Due to the forward sequence control of this node (indicated by a broken arrow in the figure) a hit will not be reported on the occurance of either `cd` or `lp` if `REM`

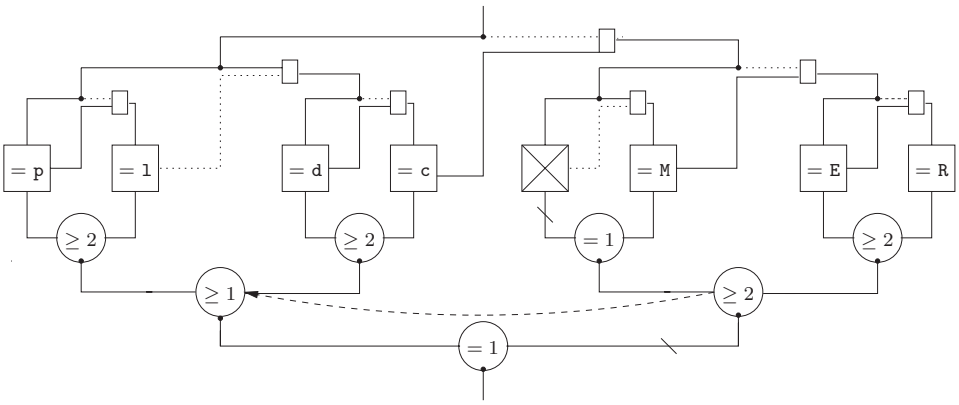

Figure 5: Illustration of complete distribution tree, PE line and result gathering tree for the query `REM BEFORE (cd|lp)`. Remark that the right result node at level two has to be configured with infinite latency.

has yet to appear. Note that the result signal from the right node on level two has to be blocked for this configuration to work.

3.2 Backward Sequence Control

Backward sequence control works like forward sequence control, only in the opposite direction, i.e. a node cannot report a hit unless the node behind it also reports a match. Backward sequence control will be essential when we explain the matching of skipping patterns in section 3.4.

3.3 Repeating Patterns

It may be of great interest to match a query where parts of the expression is repeated one or several times. We start off with a very simple example query, and describe a way to implement matching of repeating patterns in a tree structure like in the PMC.

Example 6 (Repeating patterns I). Consider the query `co+p`, i.e. a `c`, directly followed by one or more `o`'s, followed by a `p`. Figure 6 shows the appropriate configuration. Note that there is only one PE, namely the one

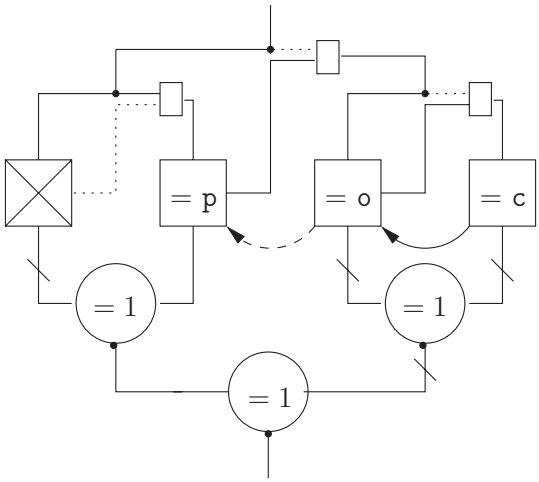

Figure 6: The appropriate configuration for the query `co+p` is shown above. The broken arrow means forward sequence control, while the solid represent forward sequence control with repeat.

responsible for matching `p`'s, that is allowed to report its match to the result tree. All other PEs have their result signal blocked. However, because this PE is configured with forward sequence control, it cannot report a hit if a `p` occurs unless there is also a match on the PE responsible for matching `o`'s. This particular PE is configured with a special sequence control called *forward sequence control with repeat*, which means that either of the following conditions must be satisfied for this PE to indicate a match:

(i) An `o` is matched by the PE, and the PE in front of it, the one matching `c`'s, also reports a match. This is exactly the same as with plain forward sequence control.

(ii) An `o` is matched by the PE, and there was also a match reported by this PE one clock cycle ago. This means that the two rightmost PEs in the line has recognized the pattern `co+`, that is a `c` followed by one or more `o`'s.

Naturally, the second condition cannot be satisfied unless the first one has hit to allow matching of repeated `o`'s. As long as the pattern `co+` is recognized, that is the second rightmost PE reports a match, the second leftmost PE is

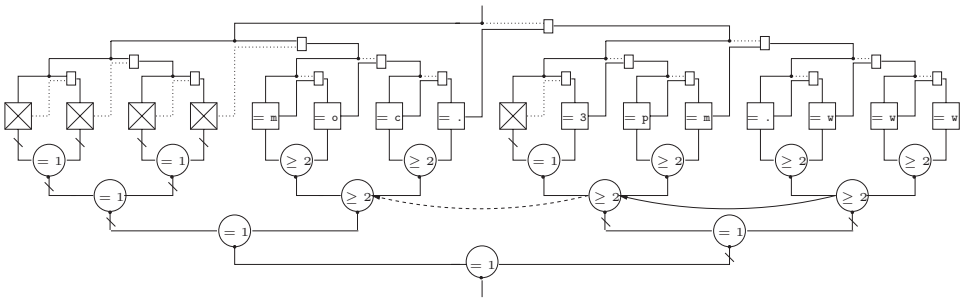

Figure 7: Illustration of a tree configuration for the query `www\.(mp3)+\.com`. Note that to complete the tree it is necessary to introduce a "dummy" subtree where all data is distributed in parallel and nothing is matched, whence it has no effect.

allowed to report a match if a `p` is shifted in. This means that the result node at level two will report a match, i.e. the entire pattern `co+p` is recognized. Valid matches are for instance `cop`, `coop`, `cooop` and so on. However, if a `p` does not occur, and any character besides `o` is shifted into the second rightmost PE, the pattern `co+` is no longer recognized and the match is cleared, meaning that a full match is no longer possible until another `co+` pattern arrives in the data stream.

Obviously, the repeating patterns can be longer than a single character. We illustrate this with another example.

Example 7 (Repeating patterns II). There is no principal difference between matching a single repeating character and a repeating pattern of arbitrary length. However, as we have seen before, the matching of a pattern require that an appropriate subtree is configured for this purpose, whence sequence control must be applied at a higher level in the tree. Consider the query `www\.(mp3)+\.com`. Note that `\.` is the IQL punctuation mark notation (the single character `.` is reserved for wild cards). Figure 7 shows the appropriate configuration for this particular query. We see that the situation is no different from example 6 and figure 6, except for the "dummy" subtree with four disabled PEs, that is included to complete the tree. The conditions for a full match remain the same, i.e. the rightmost subtree with four PEs

has to match `www.` for the second rightmost subtree to report a hit on an occurence of `mp3`. If this is the case, the second leftmost subtree is allowed to match `.com`, thus setting up the result node at level four to match the string `www.mp3.com`. Another possibility is that the second rightmost subtree can continue to report a hit three clock cycles later if and only if there is a repeated occurence of the string `mp3`. Any match of the pattern `www\.(mp3)+` will be cleared if not followed by either of the strings `.com` or `mp3`.

Clearly, if a node is to match repeating patterns it has to hold information about the past, e.g. the second rightmost PE in example 6 has to know whether or not it reported a match one clock cycle ago. Equivalently, the subtree reporting an occurence of the pattern `www\.(mp3)+` in example 7 need information on its own state three clock cycles ago. Generally, a subtree which is supposed to match a repeating string needs to hold all its reported results for as many clock cycles as equals the length of the data path in its own data distribution subtree. This is done by linking memory bits in a flip-flop chain to hold historical information. The bits flow through the chain so that the nth bit tells if there was a match for this node n clock cycles earlier.

3.4 Skipping Patterns

Skipping patterns, i.e. where parts of the query may or may not occur in the data stream, are important search features. They are also easily implemented on the PMC, and utilize *backward sequence control with skip*. We illustrate with an example.

Example 8 (Skipping patterns). Consider the query `Joh?n`, that is `Jo`, with an optional `h`, followed by an `n`, i.e. `John` and `Jon` will be matched. Figure 8 shows an appropriate configuration for this query. The broken arrows illustrate backward sequence control, while the solid arrow represents backward sequence control with skip. The backward sequence control with skip configuration means that the second leftmost node has two possibilities for indicating a match:

(i) The leftmost PE reports a matching `n` character, and the second leftmost PE can at the same time match the `h`. This is exactly the same as with plain backward sequence control.

(ii) There was a match on the leftmost PE one character ago. Since the `h` is optional the result remains positive.

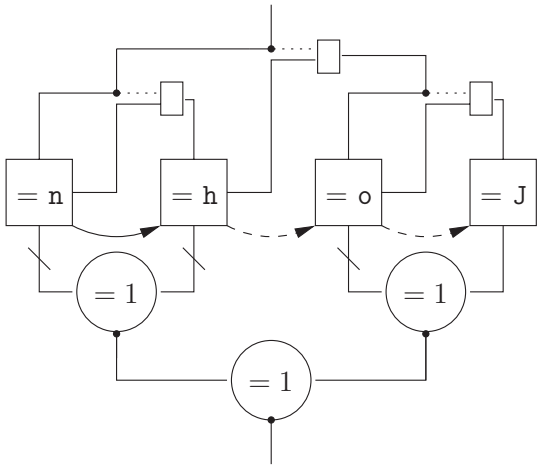

Figure 8: Skipping patterns tree configuration illustrated for query `Joh?n`, i.e. occurences of either of the strings `John` or `Jon` will be matched.

As the PE responsible for matching `o`'s has backward sequence control, it cannot report a hit unless the pattern `h?n` has been matched. Likewise, the PE matching `J`'s cannot report a match unless `oh?n` is already matched. Hence, the desired result can be obtained by blocking all results to the result tree except from the PE matching `J`'s. This PE alone reports a full match of the query.

As for forward sequence control with repeat, a node utilizing backward sequence control with skip needs to know whether there was a hit on its left sibling node n clock cycles ago, where n is the length of the data path of the skipped pattern subtree. In example 8 this subtree held only one character, i.e. the length of the data path was one. However, when having arbitrary optional parts in a query, the situation changes as in example 7 with repeating patterns.

3.5 Sequence Control Limitations

Sequence control is available at all levels in the result tree, including level zero (the PE level). However, design constraints and practical considerations are the reasons why the PMC has the possibility of using sequence control with

skipping or repeating patterns only at levels 0, 2, 4, 6 and 8. Also, the length of the flip-flop chains are 1, 4, 16, 64 and 256 at these levels respectively. Note that this does not restrain functionality since the numbers are identical to the maximum length of the data path for these levels. Also, sequence control chains cannot span more than sixteen nodes at the same level.

As we have seen, the length of the expressions to be repeated or skipped has to be predefined, which is why the PMC cannot handle nested repeats or skips. Furthermore, because repeating and skipping patterns are fundamentally different in that the former use forward sequence control and the latter backward sequence control, one cannot use both in the same sequence control chain.

4 Managing Documents and Hits

The aim of this chapter is to introduce the *document manager* and the *hit managers* of the PMC, and how they are used to search and manage hits from multiple queries in multiple consequtive documents. We start off with the document manager, before moving on to the hit managers of the result gathering tree.

4.1 The Document Manager

A common situation when searching large data sets, is that there exist a natural way to divide them into non-overlapping subsets. For instance, books are easily divided into chapters and genomes into chromosomes. In the following we will refer to these subsets as documents.

Occasionally, hits spanning several documents are undesireable, so it is necessary to have some way of knowing which document a hit belongs to. For this reason, the PMC is equipped with a document manager, which is capable of matching a predefined document separator tag of length one, two, three or four bytes. This is done simply by chaining three extra processing elements. Hence, the document separator tag to be matched can be represented with the same functionality as in the pattern matching part of the PMC, i.e. each PE can match characters that are either less than or equal to, greater than or equal to, not equal to or equal to a predefined character. When a document separator tag is encountered in the data stream, the document

manager feeds a signal to the data distribution tree which follows the same path as the data stream. Hence, this signal will reach all nodes at the same time as the characters defining the document separator tag, thus being able to prevent all matches and reset all values such as latencies due to earlier matches, so that the PMC can start afresh on the new document.

Example 9 (Tag of length two). Needless to say, an expression which is supposed to be a document separator tag cannot occur in any other place in the data set except between documents. Hence, some knowledge of the data is necessary to choose an appropriate document separator tag. Assume that the complete works of William Shakespeare - thirtyseven plays and five poems - are concatenated into one large data set. A sensible document separator tag of length two could for instance be `##`, as two consecutive `#`'s are very unlikely to appear in written English.

4.2 The Hit Managers

Many of the queries previously presented as examples in this text use very few, some no more than two, PEs. In theory, all nodes in the result tree could report hits. To limit the number of hits to an amount small enough for the receiving system to handle, and at the same time ensure that a reasonably high number of queries can be handled simultaneously, the PMC has hit managers from level four and higher in the result gathering tree. Hence, a query can use no less than sixteen (2^4) PEs, and the maximum number of simultaneous queries per PMC become 64 if all queries use the minimum amount of PEs [1]. Altogether, there are 127 hit managers in the result tree, each of which has five different modes of operation:

(i) The hit manager can be switched off, i.e. no hits are reported to host system memory. This is the default configuration.

(ii) The hit manager can report all hits to host system memory.

(iii) To limit the amount of hits, the hit manager can report only the first hit during a predefined range of bytes to host system memory.

[1] Note that a query can be arbitrary complex, as several PMCs can be used to search for patterns whose queries does not map on a single chip.

(iv) Like the previous mode of operation, the main purpose of this mode is to limit the amount of reported hits, but instead of reporting the first hit within a range of bytes, the PMC is to report only the first hit within a document to host system memory.

(v) To handle queries asking if a pattern is `NOT` present in the data stream, there is a mode for reporting a hit to host system memory if there is a hit at the last byte of a document.

The hit managers can report either the address or the document and hit number of the hits. The hit counter is reset between documents, meaning that the hit number represent hit count number within the current document.

A Result Gathering Tree Details

As described in section 2.4, the nodes in the result gathering tree can perform one out of eight possible operations on the results from their left and right children. Thus, they can be viewed as binary operators. This appendix contains details with respect to the functionality of the result gathering tree. We start off by explaining how alphabetical and numerical comparisons are performed, before moving on to an overview of how to implement all boolean operators with the functionality present in the tree.

A.1 Alphabetical and Numerical Comparisons

By exploiting the recursive nature of the alphabetical and numerical string comparison problem (denote the operator `A/N`), one can perform alphabetical and numerical comparisons in a tree structure like that of the PMC. Alphabetical and numerical comparisons are best illustrated with an example.

Example 10 (Alphanumeric on the PMC). Consider the query (≥`1990`), i.e. match all expressions that are alphabetically/numerically larger than `1990`. Figure 9 illustrates the PMC configuration. The extra lines in the result gathering tree are drawn to illustrate that every node in the result tree in the PMC has knowledge of whether or not there was an equality in the nodes above them, e.g. the right node at level one knows both if the rightmost PE matched a character greater than or equal to `1` and, in addition, if it was

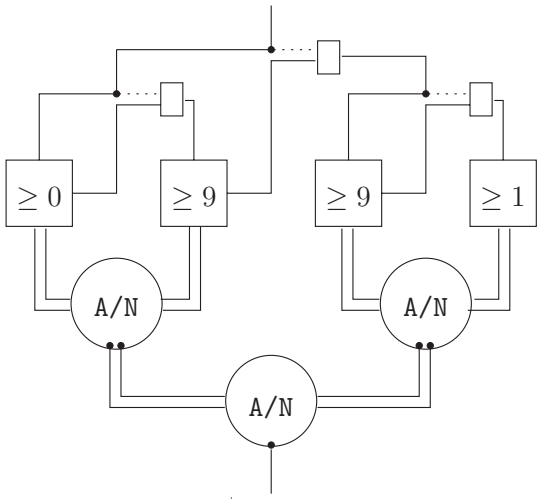

Figure 9: Illustration of the configuration for the query `(>=1990)`, i.e. match all strings alphabetically/numerically larger than 1990.

identical to `1`. Obviously, if the rightmost PE does not match, the string is alphabetically/numerically smaller than `1990`. If it matches, and in addition there is no equality, the string is greater. If the match is an equality, one has to gain information from the second rightmost PE. If there is also an equality here, the right node at level one can conclude with an equality, and one has to move on to the left node at level one. Denoting the equlity and result signals from the upper right and left branches eq_{right}, res_{right}, eq_{left} and res_{left} respectively, the `A/N` operation becomes

$$res = \{res_{\text{right}} \texttt{ AND } (\texttt{ NOT } eq_{\text{right}})\} \texttt{ OR } \{res_{\text{left}} \texttt{ AND } eq_{\text{right}}\}$$

and

$$eq = eq_{\text{right}} \texttt{ AND } eq_{\text{left}}.$$

A.2 Implementing Boolean Functions

With the operations of the result tree introduced in section 2.4, all boolean operations can be performed. Again, denoting the result from the left upper branch L, the result from the right upper branch R, and assuming that these

Function	Implementation	Commment
0	$L + R \geq 3$	Null
(L `AND` R)	$L + R \geq 2$	
(L `AND NOT` R)	$L > R$	
(Transfer L)	$L + R$	Subtree R assumed to always generate zero
(L `NOR NOT` R)	$R > L$	Subtrees swapped
(Transfer R)	$L + R$	Subtree L assumed to always generate zero
(L `XOR` R)	$L + R = 1$	
(L `OR` R)	$L + R \geq 1$	
(L `NOR` R)	$L + R \leq 0$	
(L `XNOR` R)	$L = R$	Equivalence
(`NOT` R)	$L + R \leq 0$	Subtree L assumed to always generate zero
(L `OR NOT` R)	$L \geq R$	Implication, if R then L else true
(`NOT` L)	$L + R \leq 0$	Subtree R assumed to always generate zero
(`NOT` L `OR` R)	$R \geq L$	Implication, if L then R else true
(L `NAND` R)	$L + R \leq 1$	
1	$L + R \geq 0$	Identity

Table 1: Implementation of all boolean functions using the functionality present in the nodes of the result gathering tree.

results are always zero or one when performing boolean functions, table 1 shows an overview on how to implement them.

References

[1] Fast Search & Transfer ASA. The FAST pattern matching chip, rev. c1. More detailed description of sequence control.

[2] Fast Search & Transfer ASA. The FAST query language (FQL). Reference guide for a high level query language.

[3] Fast Search & Transfer ASA. The PMC specification language (PSL). Reference guide for chip specific query language.

[4] Synopsis Finland Oy. Architectural plan, PMC. Design document, version 0.3, November 25, 1999.

[5] Synopsis Finland Oy. Technical specification, PMC. Design document, version 0.21, Mai 08, 2001.