Programming and Optimization of Big-Data Applications
on Heterogeneous Computing Systems
PROGRAMMING AND OPTIMIZATION
OF BIG-DATA APPLICATIONS
ON HETEROGENEOUS COMPUTING
SYSTEMS

SUEJB MEMETI

LINNAEUS UNIVERSITY PRESS
Abstract

The next-generation sequencing instruments enable biological researchers to generate voluminous amounts of data. In the near future, it is projected that genomics will be the largest source of big-data. A major challenge of big data is the efficient analysis of very large data sets. Modern heterogeneous parallel computing systems, which comprise multiple CPUs, GPUs, and Intel Xeon Phis, can cope with the requirements of big-data analysis applications. However, utilizing these resources to their highest possible extent demands advanced knowledge of various hardware architectures and programming frameworks. Furthermore, optimized software execution on such systems demands consideration of many compile-time and run-time system parameters.

In this thesis, we study and develop parallel pattern matching algorithms for heterogeneous computing systems. We apply our pattern matching algorithm for DNA sequence analysis. Experimental evaluation results show that our parallel algorithm can achieve more than 50x speedup when executed on host CPUs and more than 30x when executed on Intel Xeon Phi compared to the sequential version executed on the CPU.

Thereafter, we combine machine learning and search-based meta-heuristics to determine near-optimal parameter configurations of parallel matching algorithms for efficient execution on heterogeneous computing systems. We use our approach to distribute the workload of the DNA sequence analysis application across the available host CPUs and accelerating devices and to determine the system configuration parameters of a heterogeneous system that comprise Intel Xeon CPUs and Xeon Phi accelerator. Experimental results show that the execution that uses the resources of both host CPUs and accelerating device outperforms the host-only and the device-only executions.

Furthermore, we propose programming abstractions, a source-to-source compiler, and a run-time system for heterogeneous stream computing. Given a source code annotated with compiler directives, the source-to-source compiler can generate device-specific code. The run-time system can automatically distribute the workload across the available host CPUs and accelerating devices. Experimental results show that our solution significantly reduces the programming effort and the generated code delivers better performance than the CPUs-only or GPUs-only executions.

Keywords: Big Data, Heterogeneous Parallel Computing, Software Optimization, Source-to-source Compilation
To my family and friends
Abstract

The next-generation sequencing instruments enable biological researchers to generate voluminous amounts of data. In the near future, it is projected that genomics will be the largest source of big-data. A major challenge of big data is the efficient analysis of very large data sets. Modern heterogeneous parallel computing systems, which comprise multiple CPUs, GPUs, and Intel Xeon Phis, can cope with the requirements of big-data analysis applications. However, utilizing these resources to their highest possible extent demands advanced knowledge of various hardware architectures and programming frameworks. Furthermore, optimized software execution on such systems demands consideration of many compile-time and run-time system parameters.

In this thesis, we study and develop parallel pattern matching algorithms for heterogeneous computing systems. We apply our pattern matching algorithm for DNA sequence analysis. Experimental evaluation results show that our parallel algorithm can achieve more than 50× speedup when executed on host CPUs and more than 30× when executed on Intel Xeon Phi compared to the sequential version executed on the CPU.

Thereafter, we combine machine learning and search-based meta-heuristics to determine near-optimal parameter configurations of parallel matching algorithms for efficient execution on heterogeneous computing systems. We use our approach to distribute the workload of the DNA sequence analysis application across the available host CPUs and accelerating devices and to determine the system configuration parameters of a heterogeneous system that comprise Intel Xeon CPUs and Xeon Phi accelerator. Experimental results show that the execution that uses the resources of both host CPUs and accelerating device outperforms the host-only and the device-only executions.

Furthermore, we propose programming abstractions, a source-to-source compiler, and a run-time system for heterogeneous stream computing. Given a source code annotated with compiler directives, the source-to-source compiler can generate device-specific code. The run-time system can automatically distribute the workload across the available host CPUs and accelerating devices. Experimental results show that our solution significantly reduces the programming effort and the generated code delivers better performance than the CPUs-only or GPUs-only executions.

Keywords: Big Data, Heterogeneous Parallel Computing, Software Optimization, Source-to-source Compilation
Acknowledgments

The years I have spent doing my Ph.D. have been remarkable. I broadened my knowledge, improved my research skills, published scientific articles, met and worked with scientists from all over the world, and made new friends and colleagues. All this, thanks to my supervisor prof. Sabri Pllana for the opportunity to study at Linnaeus University. I would like to express my sincerest gratitude for his guidance, advice, and invaluable insights. I would like to extend my gratitude to my co-supervisor prof. Jesper Andersson for providing excellent working and learning environment as the head of department of computer science. I am grateful to prof. Welf Löve for serving as examiner during my Ph.D. studies.

I would like to thank my fantastic colleagues at the Department of Computer Science and Media Technology for the time we worked together and their friendship.

I wish to thank prof. Pawel Czarnul for accepting to act as opponent of this thesis, as well as prof. Erwin Laure, prof. Kristian Sandahl, prof. Miquel Pericas, and prof. Koraljka Golub for taking part in the evaluation committee. Additionally, I would like to thank prof. Arianit Kurti for serving as moderator during the Ph.D. defense process.

I am grateful to my wife for her love and support through good and challenging times. I express my sincerest gratitude for my parents, brother, and sisters for their love, support, and encouragement along the way. You have always been there for me, making my journey easier and more joyful.

Besides the Ph.D. title, I am honored and grateful to my little princess for crowning me with the title of father.

Växjö, Sweden
October 29, 2018
## Contents

1 Introduction .......................... 1
   1.1 Parallel Computing Systems .......... 3
   1.2 Big-data .......................... 4
   1.3 Problem Formulation ................. 6
   1.4 Scope of the Thesis .................. 7
   1.5 Methods ........................... 9
   1.6 Overview of the Thesis ............... 10
   1.7 Contributions of the Thesis .......... 10
   1.8 Conclusions and Future Work ......... 17

   2.1 Introduction ........................ 29
   2.2 Research Methodology ................. 32
   2.3 Taxonomy and Terminology ............. 35
   2.4 Compile-Time ........................ 42
   2.5 Run-Time ........................... 50
   2.6 Conclusion .......................... 64

3 PAPER II: PaREM: A Novel Approach for Parallel Regular Expression Matching 69
   3.1 Introduction ........................ 69
   3.2 Methodology ........................ 71
   3.3 Implementation ....................... 75
   3.4 Experimental Evaluation .............. 78
   3.5 Related Work ........................ 80
   3.6 Summary and Future Work ............. 82

4 PAPER III: A Machine Learning Approach for Accelerating DNA Sequence Analysis 85
   4.1 Introduction ........................ 85
   4.2 Background .......................... 88
   4.3 Design and Implementation Aspects of our Algorithm for DNA Sequence Analysis .......... 90
   4.4 A Machine Learning Approach for Accelerating DNA Sequence Analysis .................. 93
   4.5 Evaluation ........................... 96
   4.6 Comparison to Related Work .......... 103
   4.7 Summary and Future Work ............. 108

5 PAPER IV: Combinatorial Optimization of DNA Sequence Analysis on Heterogeneous Systems 113
   5.1 Introduction ........................ 113
   5.2 Background .......................... 116
   5.3 Methodology ........................ 118
   5.4 Evaluation ........................... 126
   5.5 Related Work ........................ 133
List of Figures

1.1 Data and accelerator performance growth .............................................. 2
1.2 Finite automata based DNA sequence analysis on heterogeneous parallel computing systems ...................................................... 5
1.3 A high-level overview of the thesis ......................................................... 10
2.1 Our interactive browser of the results of literature review ......................... 32
2.2 Research methodology ..................................................................... 32
2.3 The process of searching and selecting the relevant literature ................. 33
2.4 Classification of the software optimization approaches .......................... 38
2.5 Software life-cycle activities ................................................................. 40
2.6 Classification of state-of-the-art work based on the target architecture .... 41
3.1 Automaton $A$ for matching the pattern $\textit{parallel}$ .............................. 72
3.2 The PaREM toolchain ................................................................... 75
3.3 Abstract syntax tree ........................................................................ 77
3.4 Transformation of RE operators into NFA ........................................... 77
3.5 Performance results ......................................................................... 79
3.6 Comparison between PaREM algorithm and general enumeration approach ........................................................................... 81
4.1 An example of a DNA sequence analysis application running on the host and device ................................................................. 87
4.2 The heterogeneous platform that comprise two CPUs and one Intel Xeon Phi .............................................................. 88
4.3 Thread-level and SIMD parallelism ...................................................... 91
4.4 The process of converting an Aho-Corasick NFA into a DFA ............. 92
4.5 The work-flow of our solution .............................................................. 93
4.6 Parallel DNA sequence analysis using resources of the host and the device ................................................................. 95
4.7 Performance prediction accuracy for the host ...................................... 99
4.8 Performance prediction accuracy for the device ................................. 100
4.9 The frequency of performance prediction error for the host ............... 101
4.10 The frequency of performance prediction error for the device .......... 101
4.11 Model-based partitioning of different DNA sequences and the corresponding execution times for various numbers of threads on the host and device ..................................................... 102
4.12 Speedup of our algorithm compared to PFAC .................................. 104
4.13 The target heterogeneous platform ..................................................... 117
4.14 Load balancing strategy among the available threads ....................... 119
4.15 The process of converting an Aho-Corasick NFA into a DFA ........... 122
4.16 Using resources of the host and device for DNA sequence analysis 124
5.1 Performance of our parallel DNA sequence analysis algorithm on host and device .................................................................... 128

PAPER V: Combinatorial Optimization of Work Distribution on Heterogeneous Systems 139
6.1 Introduction ...................................................................................... 139
6.2 Background and Motivation .............................................................. 141
6.3 Design and Implementation ............................................................... 144
6.4 Evaluation ...................................................................................... 149
6.5 Related Work .................................................................................. 156
6.6 Summary and Future Work .............................................................. 157

PAPER VI: Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming Productivity, Performance, and Energy Consumption 161
7.1 Introduction ...................................................................................... 161
7.2 Our Methodology and Tools .............................................................. 163
7.3 Evaluation ....................................................................................... 165
7.4 Related Work .................................................................................. 172
7.5 Summary ......................................................................................... 173

8.1 Introduction ...................................................................................... 177
8.2 HSTREAM: Language Extension to Support Heterogeneous Stream Computing ............................................................................ 180
8.3 Evaluation ....................................................................................... 187
8.4 Related Work .................................................................................. 191
8.5 Conclusion and Future Work ............................................................ 192

PAPER VIII: Combinatorial Optimization of Work Distribution on Heterogeneous Systems 139
6.1 Introduction ...................................................................................... 139
6.2 Background and Motivation .............................................................. 141
6.3 Design and Implementation ............................................................... 144
6.4 Evaluation ...................................................................................... 149
6.5 Related Work .................................................................................. 156
6.6 Summary and Future Work .............................................................. 157

8.1 Introduction ...................................................................................... 177
8.2 HSTREAM: Language Extension to Support Heterogeneous Stream Computing ............................................................................ 180
8.3 Evaluation ....................................................................................... 187
8.4 Related Work .................................................................................. 191
8.5 Conclusion and Future Work ............................................................ 192
Listings

4.1 Vectorization report of the SIMD delta function ........ 91
4.2 System environment variables to manage the thread affinity, number of cores, threads per core, and number of threads .... 98
5.1 Vectorization report for the SIMD delta function .......... 120
8.1 An example of the HSTREAM compiler directive ............ 180
8.2 An excerpt of the HSTREAM grammar ..................... 184
8.3 An example of the string template library that is used to generate the CUDA memory transfer statements .... 184
8.4 HSTREAM TRIAD function of the STREAM benchmark ...... 185
8.5 TRIAD function of the STREAM benchmark designed for execution on multi-core CPUs .......................... 185
8.6 TRIAD function of the STREAM benchmark designed for execution on GPUs ..................................... 186
8.7 Triad function of the STREAM benchmark designed for execution on Intel Xeon Phi ................................. 186
Glossary

Effort$_{par}$ Parallelization effort. 167

$I'$ Input fraction to be processed by host CPUs. 94

$I''$ Input fraction to be processed by accelerating device. 94

LOC$_{par}$ The fraction of lines of code used to parallelize a given application. 167

LOC$_{total}$ Total lines of code. 167

$\delta$ The transition function - $Q \times \Sigma \rightarrow Q$. 5, 71

$\mu$OS Micro Operating System - lightweight Linux Operating System. 142

$\Sigma$ Finite input alphabet. 4, 71

absolute difference The absolute difference of application’s performance between two system configurations (for instance, between EM and SAML). 131, 153

absolute error The absolute error between the predicted and measured performance of a given application. 99, 150

percent difference The difference of application’s performance expressed in percentage between two system configurations (for instance, between EM and SAML). 131, 153

percent error The error between the predicted and measured performance of a given application expressed in percentage. 99, 150

$q_0$ Start state. 5, 71

A Adenine. 5, 85

AC Aho-Corasick. 4, 117

ANN Artificial Neural Networks. 54

ANTLR Another Tool for Language Recognition. 183

AST Abstract Syntax Tree. 76

BDTR Boosted Decision Trees Regression. 54

C Cytosine. 5, 85
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeStat</td>
<td>A Tool for Quantifying the Parallelization Effort. 164</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit. 1</td>
</tr>
<tr>
<td>CS</td>
<td>Chunk Size. 190</td>
</tr>
<tr>
<td>CSDT</td>
<td>Cost Sensitive Decision Table. 48</td>
</tr>
<tr>
<td>DFA</td>
<td>Deterministic Finite Automata. 70, 71</td>
</tr>
<tr>
<td>DNA</td>
<td>Deoxyribonucleic Acid. 5, 85</td>
</tr>
<tr>
<td>DT</td>
<td>Decision Tree. 44</td>
</tr>
<tr>
<td>E</td>
<td>Energy. 125, 146, 167</td>
</tr>
<tr>
<td>EM</td>
<td>Enumeration and Measurements. 114, 131, 144, 154</td>
</tr>
<tr>
<td>EML</td>
<td>Enumeration and Machine Learning. 145</td>
</tr>
<tr>
<td>Enum</td>
<td>Enumeration. 80</td>
</tr>
<tr>
<td>EST</td>
<td>Expressed-Sequence-Tags. 116</td>
</tr>
<tr>
<td>F</td>
<td>A distinguished set of accepting states. 5, 71</td>
</tr>
<tr>
<td>FA</td>
<td>Finite Automata. 4, 69, 89, 116</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array. 30</td>
</tr>
<tr>
<td>G</td>
<td>Guanine. 5, 85</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm. 44</td>
</tr>
<tr>
<td>GPGPU</td>
<td>General Purpose Graphical Processing Unit. 86</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit. 1</td>
</tr>
<tr>
<td>GrA</td>
<td>Greedy Algorithm. 44</td>
</tr>
<tr>
<td>HC</td>
<td>Hill Climbing. 44</td>
</tr>
<tr>
<td>HSTREAM</td>
<td>Heterogeneous Stream. 178</td>
</tr>
<tr>
<td>I</td>
<td>Input string. 94</td>
</tr>
<tr>
<td>IR</td>
<td>Intermediate Representation. 184</td>
</tr>
<tr>
<td>kNN</td>
<td>k-Nearest Neighbor. 48</td>
</tr>
<tr>
<td>LEO</td>
<td>Language Extensions for Offloading. 16, 179</td>
</tr>
<tr>
<td>LOC</td>
<td>Lines of Code. 164, 189</td>
</tr>
<tr>
<td>LR</td>
<td>Linear Regression. 48</td>
</tr>
<tr>
<td>LRPR</td>
<td>Logistic Regression. 48</td>
</tr>
<tr>
<td>MC</td>
<td>Memory Controller. 89</td>
</tr>
<tr>
<td>MIC</td>
<td>Many Integrated Core Architecture. 16, 186</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-layer Perceptron. 48</td>
</tr>
<tr>
<td>MPSS</td>
<td>Intel Manycore Platform Software Stack. 97</td>
</tr>
<tr>
<td>NB</td>
<td>Naive Bayes. 48</td>
</tr>
<tr>
<td>NCBI</td>
<td>National Center for Biotechnology Information. 114</td>
</tr>
<tr>
<td>NFA</td>
<td>Nondeterministic Finite Automata. 70, 71</td>
</tr>
<tr>
<td>NN</td>
<td>Nearest Neighbor. 44</td>
</tr>
<tr>
<td>OCL</td>
<td>OpenCL. 171</td>
</tr>
<tr>
<td>ODA</td>
<td>Observe- Decide-Act. 59</td>
</tr>
<tr>
<td>OMP</td>
<td>OpenMP. 171</td>
</tr>
<tr>
<td>P</td>
<td>Pattern. 4</td>
</tr>
<tr>
<td>p</td>
<td>The Boltzmann probability distribution. 126</td>
</tr>
<tr>
<td>Par</td>
<td>Parallel Execution. 127, 129</td>
</tr>
<tr>
<td>PaREM</td>
<td>Parallel Regular Expression Matching. 11, 70, 73</td>
</tr>
<tr>
<td>PDL</td>
<td>Platform Description Language. 181</td>
</tr>
<tr>
<td>PFAC</td>
<td>Parallel Failureless Aho-Corasick. 102, 108, 129</td>
</tr>
<tr>
<td>PIS</td>
<td>Possible Initial States. 120, 122</td>
</tr>
<tr>
<td>PRO</td>
<td>Parallel Rank Ordering. 44</td>
</tr>
<tr>
<td>PSD</td>
<td>Predictive Search Distribution. 44</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization. 58</td>
</tr>
<tr>
<td>PU</td>
<td>Processing Unit. 178</td>
</tr>
<tr>
<td>Q</td>
<td>Finite set of states. 4, 71</td>
</tr>
<tr>
<td>RE</td>
<td>Regular Expression. 70</td>
</tr>
<tr>
<td>REM</td>
<td>Regular Expression Matching. 69, 123</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning. 61</td>
</tr>
<tr>
<td>RSI</td>
<td>Ruled Set Induction. 44</td>
</tr>
<tr>
<td>S</td>
<td>S factor used for partitioning of the input. 94</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing. 13, 115</td>
</tr>
<tr>
<td>SAM</td>
<td>Simulated Annealing and Measurements. 145, 154</td>
</tr>
<tr>
<td>SAML</td>
<td>Simulated Annealing and Machine Learning. 145, 154</td>
</tr>
<tr>
<td>Seq</td>
<td>Sequential Execution. 101, 127, 129</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data. 115</td>
</tr>
<tr>
<td>SM</td>
<td>Streaming Multiprocessors. 166</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

With Moore’s law [39] reaching its end, the computing industry changed course by switching from an established paradigm (that was sequential processing) to parallel processing, which is a milestone in the history of computing [50]. This paradigm-shift was inevitable because the clock rates of single-core processors could no longer increase due to the physical limitations of the Central Processing Unit (CPU), such as heat dissipation and power consumption.

Multi-core CPUs aim to maintain the speed of the sequential applications while providing more cores, which usually run at higher clock frequencies and are optimized for both sequential and parallel processing. Many-core processing units (also known as accelerating devices), such as the Graphical Processing Unit (GPU) [41] and Intel Xeon Phi [10], focus on the execution throughput of highly parallel applications. They usually comprise a larger number of cores, which run at lower clock frequencies and are optimized for parallel execution. Emerging parallel computing systems often combine multiple multi-core CPUs and accelerating devices [48]. Combination of such heterogeneous processing devices is driven by increased performance, scalability, and energy efficiency. The most powerful supercomputing centers in the world comprise multiple heterogeneous nodes. For instance, the nodes of the current fastest and the fifth greenest supercomputer in the world, named Summit, comprise two IBM POWER9 CPUs and six NVIDIA Volta V100 GPUs [54].

However, parallel programming, especially for heterogeneous computing systems is significantly more complex than sequential programming. Furthermore, programmers are traditionally trained to develop applications for single-core sequential processors, and only few programmers have parallel programming skills [44]. There are various parallel programming languages (often implemented as extensions of general-purpose programming languages such as C and C++), including, OpenMP [43], MPI [16], Pthreads [40], OpenCL [53], NVIDIA CUDA [42], OpenACC [58], and Intel TBB [56].

While these programming models ease to some extent the required program development effort, performance optimization remains a challenging task. Despite having problem domain expertise, utilizing the possibly heterogeneous parallel computing resources to the highest extent de-
1.1 Parallel Computing Systems

Parallel computing systems comprise multiple interconnected processing units and memory modules. Based on how the memory is used by the processing units, such systems may be categorized into shared and distributed memory parallel computing systems. While shared memory systems (also known as tightly-coupled or symmetric multiprocessor systems) have a global shared memory address space that can be accessed by all processing units, the processing units of the distributed memory systems (also known as loosely coupled or massive parallel computing systems) have their own local memory, and the address space of that memory can be accessed only by the corresponding processing unit.

Most of the current multi-node parallel computing systems use shared memory within a node and distributed memory between nodes [4]. The main computer architectures within the distributed parallel computing systems include cluster [13, 52], grid [5, 15, 49], and cloud computing [14, 22, 46]. While the software development and optimization of multi-node systems include cluster [13, 52], grid [5, 15, 49], and cloud computing [14, 22, 46].
computing architectures is of great importance, in this thesis, we focus on the single-node software optimization. The nodes of the mainstream platforms for parallel computing comprise multi-core and/or many-core processors. These nodes may be homogeneous or heterogeneous. Homogeneous systems comprise multiple identical cores (or multiple identical multi-core processors). Heterogeneous systems comprise non-identical cores or processors. Such systems may comprise multiple general-purpose multi-core CPUs on the host, accelerated with one or more types of accelerators. These processors may be different in performance capability, energy efficiency, and architecture.

1.2 Big-data

Big data describes a vast amount of structured and unstructured data that is collected through various mechanisms. Briefly, big-data is characterized with the three Vs, volume (that is the size of data), velocity (that is the data movement and processing rate), and variety (that is the types of data) [1]. Other characteristics of big data include veracity (that is data quality and correctness) and value (that is the data outcome), and hence in literature, researchers often characterize big data through the five Vs [8, 18].

In this thesis, we focus on analyzing big-data generated through next-generation genomics sequencing machines [60]. Please note that in the near future genomics is expected to be the largest source of data-generation [51]. In particular, we focus on using pattern matching for analysis of large-scale DNA sequences. Furthermore, we aim to use stream computing for analysis of big-data on heterogeneous parallel computing systems.

Pattern Matching Using Finite Automata

In computer science, finding occurrences of a pattern in the text is called pattern matching. Pattern matching is an important component of many applications, including Internet search engines, text-editing programs, lexical analyzers, DNA sequence analysis, data-mining, and other big-data applications. We distinguish two types of pattern or text matching: (1) exact pattern matching, where the exact sequence of tokens is searched for; and (2) approximate pattern matching, also known as fuzzy pattern matching, where the pattern does not necessarily need to be exact.

In this thesis, we focus on using exact pattern matching techniques for DNA sequence analysis. Aho-Corasick (AC), as one of the well-known algorithms for detecting any exact occurrences (including the overlapping ones) of multiple patterns, is used as a basis for our work. The AC algorithm generates a finite automaton from a given set of patterns, which is later used for matching the patterns against a given text. Finite automata (FA) is an efficient technique for pattern matching, because it examines each character from T exactly once in order to find occurrences of pattern P. Formally, FA is a quintuple of \((Q, \Sigma, \delta, q_0, F)\), where \(Q\) is a finite set of states, \(\Sigma\) is a finite input alphabet, \(\delta\) (known as the transition function) is the function \(Q \times \Sigma \rightarrow Q\), \(q_0\) is the start state, and \(F\) is a distinguished set of accepting states [17].

DNA Sequence Analysis

According to the National Academy of Engineering [3], engineering better medicines, as one of the grand challenges, aims at using genetic science to personalize the medicines. The Deoxyribonucleic Acid (DNA) of the human body comprises about three billion chemical subunits called nucleotides. The DNA sequence contains specific genetic information that is used in the development, functioning, and reproduction of living organisms. The DNA sequence comprises two biopolymer strands that form the so-called double helix, which is composed of simpler units called nucleotides. The four possible nucleotides of a DNA sequence are Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). While the overall DNA sequence of each person is basically the same, it is enough to have one single different nucleotide to give humans different physical properties. Unfortunately, such small differences sometimes may have a more dramatic effect, such as predisposing people to certain diseases. Personalized medicine may help to prevent, diagnose, treat, and monitor particular diseases, which may come as a result of gene anomalies [3, 11, 24].

![Figure 1.2: An example of using finite automata for DNA sequence analysis on heterogeneous parallel computing systems.](image)
Stream Computing

Nowadays, there exist several data warehouses that store large volumes of data [19]. The current challenges of big-data include data-storage and data-analysis. While data warehouses can handle the data-storage, there is still a need for fast data analysis, which can be performed through stream computing [19].

Stream computing deals with cases when the data is not entirely available for processing, which means that data is streamed (i.e. flowing in real-time). Internet of Things is a typical example where data is streamed and not entirely available for processing. In such cases, the data need to be processed in chunks and continuously [38].

In this thesis, we focus on using stream computing for continuous analysis of large amounts of data using heterogeneous parallel computing systems.

1.3 Problem Formulation

Unlike from the single-core era when the compiler engineers could exploit the underlying architecture of new CPUs by manually implementing several code-transformation techniques, software development and optimal execution of applications on heterogeneous parallel computing systems is much more complex. The programmers and tools are exposed to many system parameters [48], including device-specific hardware parameters (such as, number of cores, core architecture, core speed, cache capacity, register capacity, and memory hierarchy of various processing units), application parameters (such as, static and dynamic code analysis features, data-set size), and run-time system parameters (such as, scheduling policy, available resources, workload status). The disparity of various architecture characteristics dramatically increases the compiler and run-time optimization search space. Finding the optimal set of parameters for a specific context is a non-trivial task and manually tuning these parameters is impractical [37].

Furthermore, efficient utilization of the heterogeneous parallel computing resources demands advanced knowledge of various parallel computing architectures and programming frameworks.

In this thesis, we address primarily the following problems:

- fast analysis of large-scale DNA sequences on heterogeneous parallel computing systems;
- software optimization of data-parallel applications on heterogeneous parallel computing systems;
- programming stream computing applications for heterogeneous parallel computing systems.

To address the first problem, we study and develop parallel algorithms and tools for programming pattern matching applications for heterogeneous parallel computing systems. Furthermore, we aim at applying the parallel pattern matching algorithm for analysis of large-scale DNA sequences using modern heterogeneous parallel computing systems that comprise multiple multi-core CPUs on the host, and many-core accelerating devices such as Intel Xeon Phi and/or GPUs.

To address the second problem, we study and develop techniques that combine machine learning and meta-heuristics to determine the near-optimal system configurations that result with the highest application performance when executed on heterogeneous parallel computing systems. While search-based heuristics are used to explore the large space of system configurations, machine learning is used to evaluate (that is, predict) the application performance of a given system configuration.

Regarding the third problem, we study and develop programming abstractions, a source-to-source compiler, and a run-time system for stream computing on heterogeneous parallel computing systems.

Consequently, we have defined the following objectives of the thesis:

O1. To study and develop algorithms for parallel pattern matching.
O2. To study and develop intelligent techniques for optimization of data-parallel applications on heterogeneous computing systems.
O3. To study and develop programming abstractions for stream computing on heterogeneous parallel computing systems.

1.4 Scope of the Thesis

No single research study can address all the problems; therefore, we limit our study to the following aspects, (1) domain application; (2) target architecture; (3) software life-cycle activities and optimization methods; (4) programming frameworks; and (5) assumptions. In what follows, for each of these aspects, we provide more information.

Domain Application

While the proposed solutions in this thesis for software development and optimization of heterogeneous parallel computing systems may be applied to other domains, in this thesis we focus on the application of our algorithms, programming, and optimization approaches to genomics. In particular, we focus on the analysis of real-world DNA sequences using pattern matching algorithms and finite-automata on heterogeneous parallel computing systems.
Target Architecture

The main platforms for parallel computing, such as cluster, grid, and cloud computing systems, comprise multiple homogeneous and/or heterogeneous nodes. The multi-node software development and optimization are of great importance, however, in this thesis, we focus on the optimization of the resources within a single node. Particularly, we consider the software development and optimization on heterogeneous parallel computing systems that comprise multiple multi-core host CPUs and many-core accelerating devices such as Intel Xeon Phi and NVIDIA GPUs.

Software Life-cycle Activities and Optimization Methods

Software optimization can happen during different activities of the software life-cycle, including design and implementation-time, compile-time, and run-time [37]. Regarding the software life-cycle, we limit our scope to compile- and run-time optimizations. While there are other software optimization methods, such as linear programming, dynamic programming, and control theory, we limit our study to the use of machine learning and meta-heuristics.

Programming Frameworks

There are several programming models and frameworks for heterogeneous parallel computing systems, such as OpenMP, OpenACC, OpenCL, and CUDA. In this thesis, we focus on using OpenMP for programming multi-core host CPUs, Intel Language Extensions for Offloading for programming Intel Xeon Phi coprocessors, and CUDA for NVIDIA GPU programming. Please note that our solutions (for code generation and optimization) are generic regarding programming frameworks. With a certain amount of time investment, support for other programming frameworks could be added.

Assumptions

The results of this thesis rely on the following assumptions:

- We assume that the software is running in a dedicated system and all the resources are available for use. Considering the external workload (that is other applications running on the same system) can complement this thesis, but is out of the scope.
- We assume that the monitoring tools for various processing devices do not significantly impact the performance of the application being executed. Considering the optimization of the monitoring tools (such as adjusting the sample interval) can complement this thesis, but is out of the scope.
- Regarding our parallel pattern matching algorithm, we assume that there is a small set of characters in the alphabet, which results in a memory-tolerable finite state machine. We also assume that the size of the input is much larger compared to the set of patterns. Considering cases where the set of patterns is larger may complement this thesis, but are out of scope.
- We assume that there is a strong relationship between performance and energy consumption, which means that the shorter the makespan less energy is consumed [36]. There is evidence that for some applications and types of architectures, higher performance does not necessarily mean lower overall energy consumption. Considering the multi-objective optimization is out of the scope of this thesis and is left for future work.

1.5 Methods

In this thesis, we have used the following methods:

- **Systematic literature review** is used to identify the challenges and opportunities of using machine learning and meta-heuristics for software optimization of parallel computing systems. Kitchenham and Charters [21] define the systematic literature review as: “a study that uses a well-defined methodology to identify, analyze, and interpret all available evidence related to a specific research question in a way that is unbiased and (to a degree) repeatable”. The main stages of systematic literature reviews include planning, conducting, and reporting [21].
- **Algorithm engineering** is used to develop our parallel pattern matching algorithm that is designed for optimal execution on a heterogeneous parallel computing system that comprises host CPUs and Intel Xeon Phi accelerating devices. Sanders [47] defines algorithm engineering as “a general methodology for algorithmic research”. The main activities associated with this method include algorithm design, analysis, implementation and experimental evaluation [47].
- **Empirical evaluation** is used to evaluate and validate the proposed approaches and compare them with other state-of-the-art approaches. Empirical measurements provide means for researchers to manipulate one or more variables, such that the effect is measured and used for further statistical analysis [59], which may help scientists and engineers to understand, control, and estimate the software. In this thesis, we follow the three-stage model for empirical evaluation in software engineering suggested by Wohlin, Höst, and Henningsson [59], which include, design, operation, and analysis & interpretation.
- **Language and compiler design** techniques [2] are used to define our parallel programming language abstractions for stream computing, implement the source-to-source compiler for device-specific code generation from a high-level code, and implement the run-time system for...
1 Introduction

automatic workload distribution on the resources of heterogeneous systems.

1.6 Overview of the Thesis

Figure 1.3 depicts a high-level overview of the thesis. The round-corner outlined boxes indicate the objectives (defined in Section 1.3) of the thesis, whereas the sharp-corner outlined boxes indicate the papers (that is, chapters) of the thesis that contribute towards achieving one or more objectives of the thesis. The solid-line arrows indicate a strong relationship between an objective and a paper, which means that the paper contributes directly towards achieving an objective. The dashed-line arrows indicate that a paper partially contributes towards achieving a certain goal of the thesis.

The first objective, that is to study and develop algorithms for parallel pattern matching (O1), is achieved by Paper II (Chapter 3). Papers III (Chapter 4), IV (Chapter 5), and V (Chapter 6) propose intelligent techniques that use machine learning and/or meta-heuristics for optimization of big-data applications, hence contributing to achieving the second objective of the study (O2). Partially, Papers III, IV, and V contribute to achieving O1 by tuning parameters of the parallel pattern matching algorithm. The third objective, that is to study and develop programming abstractions for stream computing on heterogeneous parallel computing systems, is achieved by Paper VII (Chapter 8). Paper VI (Chapter 7) partially contributes to the third objective by studying the performance, energy consumption, and programming effort of different programming frameworks.

1.7 Contributions of the Thesis

In this section, we first list the scientific publications that are included in this thesis and thereafter we list other scientific publications that were written and published during the period of my Ph.D. studies, but are not part of this thesis.

Scientific Publications Included in this Thesis

In what follows, we list all the scientific publications included in this thesis. For each of them, we provide a summary that briefly describes the scientific challenges it addresses, the proposed solution, and a list of the major contributions.

PAPER I - Chapter 2

“Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review” [37]

Summary: While modern parallel computing systems offer high performance, utilizing these powerful computing resources to the highest extent demands advanced knowledge of various hardware architectures and parallel programming models. Furthermore, optimized software execution on parallel computing systems demands consideration of many parameters at compile-time and run-time. Determining the optimal set of parameters in a given execution context is a complex task, and therefore to address this issue researchers have proposed different approaches that use heuristic search or machine learning.

In this paper, we conduct a systematic literature review of approaches for software optimization of parallel computing systems. We focus on approaches that use machine learning or meta-heuristics published since the year 2000. We classify the selected review papers based on the software life-cycle activities, target computing systems, optimization methods, and period of publication. Furthermore, we discuss existing challenges and future research directions.

The major contributions of this paper include:

- a systematic study of the state-of-the-art software optimization methods for parallel computing systems that use machine learning or meta-heuristics;
- a classification of the existing studies based on the software life-cycle activities, target computing systems, optimization methods, and period of publication;
- a discussion of the existing challenges and future research directions.

PAPER II - Chapter 3

“PaREM: A Novel Approach for Parallel Regular Expression Matching” [32]

Summary: Regular expression matching is essential for many applications, such as finding patterns in text, exploring sub-strings in large DNA sequences, and lexical analysis. However, sequential regular expression matching may be time-prohibitive for large problem sizes.

In this paper, we describe a novel algorithm for Parallel Regular Expression Matching (PaREM) that scales gracefully for various problem sizes and numbers of threads. We devised our algorithm to be efficient for general automata independently from the number of states, and for
a large spectrum of input text-sizes. We optimized our algorithm to
do very accurate speculations on the possible initial states for each of
the sub inputs (that are split among the processing units), instead of
calculating the routes considering each state of the automaton as an
initial state. This method is more effective when the adjacency matrix
(that is used for the graph representation of the automaton) is sparse,
although it shows major improvements in dense matrices. To ease the
access to the proposed parallel algorithm for a broad spectrum of users
(including the users without a background in parallel programming),
we have developed our tool PaREM that can transform automatically
a given regular expression or finite automaton into the corresponding
code (C++ and OpenMP) for our algorithm amenable for parallel exec-
ution on shared-memory systems. Experimental results on a dual-socket
shared-memory system with 24 physical cores show a close to linear
speedup compared to the sequential implementation for problem sizes
comparable to the cache size and significant speedup for larger problem
sizes that use further levels of memory hierarchy.

The major contributions of this paper include:
- a scalable algorithm for parallel regular expression matching;
- PaREM tool that automatically generates parallel code from a given
  regular expression or finite automata;
- empirical evaluation of the proposed parallel algorithm and the
  PaREM tool using a modern dual-socket shared-memory system
  with 24 physical cores.

PAPER III - Chapter 4

“A machine learning approach for accelerating DNA sequence analysis” [25]

Summary: The DNA sequence analysis is a data and computationally
intensive problem and therefore demands suitable parallel computing
resources and algorithms.

In this paper, we present a parallel algorithm for DNA sequence anal-
ysis on heterogeneous parallel computing systems accelerated with the
Intel Xeon Phi coprocessor. The parallelization approach is based on
domain decomposition, which means that the DNA sequence is split
evenly among the available threads. To process the patterns occurring
in the cross-border, our algorithm uses $k - 1$ overlapping characters,
where $k$ is the length of the largest pattern. Furthermore, our algorithm
exploits the wide vector registers of the Intel Xeon Phi architecture and
is optimized using different algorithmic strategies. We use a machine
learning approach for optimal work-sharing of DNA sequence across
the host CPUs and accelerating devices.

We split the DNA sequence between the host and the device based on
the performance prediction such that the load is balanced between the host
and the device and the overall execution time is reduced. We developed
the performance prediction model using the Boosted Decision Tree Re-
gression that is a supervised learning algorithm. The model training is
performed using a set of 11 DNA sequences of different organisms (such
as, alpaca, armadillo, chimpanzee, coelacanth, duck, ferret, guinea pig,
molly, elephant, turtle, and zebrafish). We evaluate our approach ex-
perimentally with real-world DNA sequences (including the sequence
of human, mouse, cat, and dog) using a heterogeneous platform that
comprises two 12-core Intel Xeon E5 CPUs and an Intel Xeon Phi 7120P
accelerator with 61 cores.

The major contributions of this paper include:
- a machine learning approach for automatic work-sharing between
  the host and the device,
- an algorithm for parallel DNA sequence analysis that enables the
  efficient exploitation of resources on the host CPUs and accelerating
devices,
- an experimental evaluation of our machine learning approach for
  DNA analysis using real-world DNA sequences of human (3.2GB),
mouse (2.7GB), cat (2.4GB), and dog (2.4GB).

PAPER IV - Chapter 5

“Combinatorial optimization of DNA sequence analysis on heterogeneous sys-
tems” [28]

Summary: In this paper, we use our algorithm for DNA sequence analy-
is on heterogeneous systems accelerated with Intel Xeon Phi coproces-
sor. To determine a near-optimal work-sharing across host CPUs and
accelerating devices, and to determine the near-optimal system param-
eter configurations, including the number of threads, thread allocation
policy, we use a combinatorial optimization approach. For combinatori-
orial optimization, we use the Simulated Annealing (SA) [45] method,
which is suitable for global optimization in the context of a large dis-
crete configuration space. The objective function we aim to minimize is
the execution time of the DNA sequence analysis. We evaluate our ap-
proach experimentally with real-world DNA sequences (including the
sequence of human, mouse, cat, dog) using a heterogeneous platform
that comprises two 12-core Intel Xeon E5 CPUs and an Intel Xeon Phi
7120P coprocessor with 61 cores. We use SA to explore the system con-
figuration space and determine the near-optimal system configuration
for execution of the DNA sequence analysis.

The major contributions of this paper include:
- an algorithm for parallel DNA sequence analysis that allows ef-
cient use of the resources on host and device of heterogeneous
systems,
- a combinatorial optimization approach that determines the near
optimal work-sharing between host and device and system param-
eters (such as, number of threads, thread allocations),
- an empirical evaluation of our approach for DNA sequence analysis using real-world DNA sequences of human (3.2GB), mouse (2.7GB), cat (2.4GB), and dog (2.4GB).

PAPER V - Chapter 6
“Combinatorial Optimization of Work Distribution on Heterogeneous Systems” [29]
Summary: In this paper we propose an optimization approach that combines the Combinatorial Optimization and Machine Learning to determine near-optimal system configuration parameters of a heterogeneous system. We use Simulated Annealing as a combinatorial optimization approach to search for the optimal system configuration in the given parameter space, whereas for performance evaluation of the proposed system configurations during space exploration we use the Boosted Decision Tree Regression. The objective function we aim to minimize is the application’s execution time. To evaluate our approach, we use a parallel application for DNA Sequence Analysis on a platform that comprises two 12-core Intel Xeon E5 CPUs and an Intel Xeon Phi 7120P coprocessor with 61 cores. Using our optimization approach to determine the near-optimal system configuration we achieve a speedup of 1.74× compared to the case when only the available resources of the host are used, and up to 2.18× speedup compared to the case when all the resources of the accelerating device are used.

The major contributions of this paper include:
- a combinatorial optimization approach to explore the large system configuration space;
- a supervised machine learning approach to evaluate the performance of parallel applications;
- an approach that combines the combinatorial optimization heuristic with machine learning to determine a near-optimal system configuration, such that the execution time is decreased;
- experimental evaluation of our approach;
- performance comparison of our approach that utilizes both CPUs and accelerators, compared to CPU-only and accelerator-only approaches.

PAPER VI - Chapter 7
“Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming Productivity, Performance, and Energy Consumption” [36]
Summary: Many modern parallel computing systems are heterogeneous at their node level. Such nodes may comprise general purpose CPUs and accelerators (such as, GPU, or Intel Xeon Phi) that provide high performance with suitable energy-consumption characteristics. However, exploiting the available performance of heterogeneous architectures may be challenging. There are various parallel programming frameworks (such as, OpenMP, OpenCL, OpenACC, CUDA) and selecting the one suitable for a target context is not straightforward.

In this paper, we benchmark four well-known programming frameworks for heterogeneous systems: OpenMP, OpenACC, Open-CL, and CUDA. Besides the industry-standard benchmark suite SPEC Accel [20], we use the popular Rodinia [7] benchmark suite to evaluate programming productivity, energy efficiency, and performance. We use our tool developed for this study CodeStat to quantify the programming effort for parallelizing benchmark suites under study. Furthermore, we developed x-MeterPU, which enables us to measure the performance and energy consumption on systems accelerated with the Intel Xeon Phi and GPU. We present and discuss results obtained on two heterogeneous computing systems: *Emil* that comprises two Intel Xeon E5 processors and one Intel Xeon Phi coprocessor, and *Ida* that has two Intel Xeon E5 processors and one GTX Titan X GPU.

The major contributions of this paper include:
- development of measurement tools CodeStat and x-MeterPU for accelerated systems with Intel Xeon Phi and GPU;
- an empirical study of four widely used frameworks for programming heterogeneous systems OpenCL, OpenACC, OpenMP, and CUDA; and
- joint consideration of programming productivity, performance, and energy consumption.

PAPER VII - Chapter 8
“HSTREAM: A directive-based language extension for heterogeneous stream computing” [30]
Summary: Big data streaming applications require utilization of heterogeneous parallel computing systems, which may comprise multiple multi-core CPUs and many-core accelerating devices such as NVIDIA GPUs and Intel Xeon Phis. Programming such systems require advanced knowledge of several hardware architectures and device-specific programming models, including OpenMP and CUDA.

In this paper, we present HSTREAM, a compiler directive-based language extension that supports heterogeneous stream computing. HSTREAM aims to keep the same simplicity as programming with OpenMP and to enable programmers to easily utilize the available heterogeneous parallel computing resources on the host (that is CPU threads) and device (including GPUs, or Intel Xeon Phis). The main components of the HSTREAM solution are the compiler and the runtime system. The HSTREAM source-to-source compiler performs several analysis steps (including lexical, syntactical, and semantical) and generates target specific code from a given source code annotated with HSTREAM compiler directives and a PDL file that describes the hardware architecture. HSTREAM supports code generation for multi-core
CPUs using OpenMP, GPUs using CUDA, and Intel Xeon Phis (also known as MIC) using Intel Language Extensions for Offloading (LEO). The HSTREAM run-time distributes the workload across the heterogeneous processing units.

We use the HSTREAM source-to-source compiler to generate the heterogeneous version of the STREAM benchmark [23]. We evaluate the generated heterogeneous STREAM benchmark with respect to programming productivity and performance. The experimental results show that HSTREAM keeps the same simplicity as OpenMP, and the code generated for execution on heterogeneous systems delivers higher performance compared to CPUs-only and GPUs-only execution.

The major contributions of this paper include:
- HSTREAM compiler - a source-to-source compiler for generating target specific code from high-level directive-based annotated source code.
- HSTREAM run-time - a run-time system for scheduling the workload across various non-identical processing units.
- evaluation of the usefulness of HSTREAM using applications from the STREAM and STREAM2 benchmarks.

Scientific Publications not Included in this Thesis
In this section, we list the scientific publications that are not included in this thesis. While some of these publications present early results used to achieve the goals of this thesis, the rest complement other aspects of this dissertation.

1. “PAPA: A parallel programming assistant powered by IBM Watson cognitive computing technology” [31]
5. “Work Distribution of Data-Parallel Applications on Heterogeneous Systems” [33]
6. “Analyzing large-scale DNA Sequences on Multi-core Architectures” [27]
7. “Accelerating DNA Sequence Analysis using Intel Xeon Phi” [26]
8. “Optimal Worksharing of DNA Sequence Analysis on Accelerated Platforms” [34]

1.8 Conclusions and Future Work
In this section, we first provide some concluding remarks, which briefly describe the addressed problems, the proposed solutions, and the observations and lessons learned. Thereafter, we list the future work directions.

Conclusions
Scientific and engineering advancements of data-generation and management tools enable domain scientists to collect large amounts of data quickly and cheaply. This vast amount of data poses several challenges related to storing and fast analysis. While there are several data-warehouses that focus on storing this vast amount of data, in this thesis we presented some ideas on fast analysis of such data using modern heterogeneous parallel computing systems. Such systems combine various heterogeneous processing units, which may have different characteristics and performance capabilities. While heterogeneous parallel computing systems may provide the ability to cope with complex engineering and scientific big-data problems, utilization of the heterogeneous resources demands advanced knowledge of various parallel programming models and languages, and considering multiple compile-time and run-time, software, and hardware related configuration parameters. In such a heterogeneous context, the configuration space is large, and finding the optimal set of configuration parameters is non-trivial.

Regarding the first objective of the thesis (O1), we have presented a parallel algorithm for fast analysis of genomic data that can efficiently utilize the available resources of the heterogeneous parallel computing system. Experimental results of our parallel pattern matching algorithm show performance speedup of up to 17.6x when executed on multi-core Intel Xeon CPUs and up to 10x when executed on many-core Intel Xeon Phi coprocessor. We have applied our parallel pattern matching algorithm to DNA sequence analysis to find occurrences of DNA subsequences in a large DNA sequence. The improved performance may help domain scientists to quickly identify gene anomalies that may predispose certain people to certain diseases, and eventually engineer personal medicine to prevent them.

To achieve the second objective of the thesis (O2), we first systematically studied the literature to identify the challenges and opportunities of software optimization approaches that use machine learning and meta-heuristics for parallel computing systems. From the analysis of the reviewed scientific articles, we have observed that the use of machine learning and meta-heuristics-based techniques for optimization of parallel computing systems can deliver optimized software execution comparable to the manual code tuning or task scheduling in specific cases. Thereafter, we proposed an approach that combines machine learning and meta-heuristics to determine near-optimal system configuration parameters, in-
cluding workload fraction ratio, number of threads, number of cores, and thread scheduling policy, such that the total execution time of the application is minimized. The empirical evaluation results show that the use of meta-heuristics can efficiently explore the large configuration space and machine learning can be used to predictively infer about the performance of applications in a given context. We used the suggested system parameter values proposed by our solution to execute the DNA sequence analysis application. The results show that the resources of the host and device are efficiently utilized, and that the collaborative utilization of the resources on host and device performs 1.74x and 2.18x better compared to the execution that uses the resource on the host only and device only, respectively.

To study and develop programming abstractions for big-data stream applications on heterogeneous parallel computing systems (that is the third objective of the thesis (O3)) we have proposed high-level programming abstractions, which enable programmers to exploit the available application parallelism. The source-to-source compiler generates device-specific ready-to-execute code from the given high-level code, whereas the run-time system evenly distributes the workload across the available resources of the heterogeneous system, including host CPUs and accelerating devices such as GPU and Intel Xeon Phi. The results show that our solution maintains the same programming effort as OpenMP; the source-to-source compiler can generate functionally equivalent target specific code, including OpenMP for host CPUs, Intel LEO for Intel Xeon Phi, and CUDA for GPU accelerators; the run-time system can distribute the workload across the available resources (two host CPUs and four GPUs); and the overall speedup of the applications is improved by up to 7.18x compared to the host-only execution and up to 1.96x compared to the execution that uses the four G54PUs (that is device-only execution).

Future Work

Besides the contributions in programming and optimization of big-data applications on heterogeneous parallel computing systems, this dissertation has also paved the way for future research. We have identified the following future research directions:

- applying the parallel pattern matching algorithm for analysis of large data generated in other application domains (such as intrusion detection systems);
- tailoring our parallel pattern matching algorithm for execution on other heterogeneous systems, such as GPU and FPGA accelerated systems;
- using machine learning and meta-heuristics for multi-objective software optimization, including performance, energy consumption, and the impact on the external workload;
- investigating the use of our approach for determining near-optimal system configuration in a dynamic environment context (that is run-time);
- extend our source-to-source compiler for stream computing by adding support for other types of heterogeneous systems, for instance, the ones accelerated with FPGAs;
- extend our source-to-source compiler for stream computing by adding support for other programming models such as OpenCL and OpenACC;
- extend our run-time system with dynamic and adaptive work-sharing capabilities.
Bibliography


