Design and Optimization of Architectures for Data Intensive Computing

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical and Computer Engineering

By

Jayaprakash Pisharath

EVANSTON, ILLINOIS

December 2005
ABSTRACT

Design and Optimization of Architectures for Data Intensive Computing

Jayaprakash Pisharath

Computer technology in recent years is propelled by new hardware designs, advanced software features and multitudinous user demands. Due to this, the data collected and managed by applications is also abundant. Today’s “connect anytime and anywhere” society based on the use of advanced digital technologies has increased the expectations of users. Data access is expected to be quick, highly reliable and fast. Future systems are expected to even more data centric and compute intensive. This fact is used as a motivation in this research work to propose new techniques and optimizations that enable high speed access to data. A three-tier data driven approach is taken to propose new architectural designs and techniques. The three perspectives of data that is used are streaming data, structured databases and new-age massive datasets (could be structured or unstructured).
Streaming data is widely popular and is still emerging. In this work, new scheduling and resource allocation strategies are proposed for such stream data systems. Both performance and energy improve dramatically when the proposed schemes are deployed in existing heterogeneous systems. Also, a new framework of analysis is proposed to comprehensively study the performance and energy consumption of such high performance systems.

The next focus area of this work is modern database systems. Storage technology has evolved significantly in the recent years. In this study, modern memory technology is used as a motivation to tune and adapt modern DBMS to modern storage technology. First, dynamic hardware management schemes are proposed. Secondly, the query optimizer is also modified to reflect the change in the storage architectures. Other emerging storage paradigms are also considered in this work.

With data growing at alarming proportions, data mining is emerging to be an excellent tool to automatically extract useful information from such large datasets. The growth of data, the advancements in data mining tools, and the user expectations are totally incongruent to the improvements in the performance of general purpose computing systems. To alleviate this, a new data mining system is designed that achieves massive speeds that can never be achieved using traditional high performance techniques.
Acknowledgements

I express my sincere gratitude to everyone that helped me in accomplishing my doctoral research work. I am grateful to my research advisor, Alok Choudhary, for his constant guidance and extensive feedback. This work was supported in part by DOE under contract 28264, DOE’s SCiDAC program (Scientific Data Management Center) - award number DE-FC02-01ER25485, NSF’s NGS program under grant CNS-0406341, NSF/DARPA ST-HEC program under grant CCF-0444405, and in part by Intel Corporation. I would like to thank my committee members Lawrence Henschen and Peter Scheuermann for their valuable feedback. I am indebted to Mahmut Kandemir at the Pennsylvania State University for providing me priceless help and advice with the database research work.

I am grateful to my dad Jayarajan Pisharath and mom Bhagavathi Pisharath for their encouragement and moral support through all these years in my life, and also for providing me the best possible level of education and knowledge. Also, I would like to thank my friends, especially the gutter gang, for those lighter moments during my doctoral study at Northwestern. Thanks are due to my fellow researchers at the Center for Ultra Scale Computing and Information Security at Northwestern University for teaming with me in realizing my research goals.
## Contents

ABSTRACT iii

Acknowledgements v

List of Tables x

List of Figures xii

Chapter 1. Introduction 1

Chapter 2. Background and Related Work 10

2.1. Architecture Optimizations 10

2.2. Performance-Energy Tradeoff Frameworks and Tools 14

2.3. Modern Databases 17

2.4. Data Layouts for Emerging Storage Devices 22

2.5. Data Mining Application Analysis and System Design 23


3.1. Multiprocessor and Heterogeneous Streaming Systems 26

3.2. Approach 28
3.3. Heterogeneous Platform 29
3.4. Methodology 34
3.5. Performance-Energy Analysis 36
3.6. Overall Observations and Shortcomings of Traditional Metrics 40

Chapter 4. Energy-Resource Efficiency Framework 42
4.1. ERE Framework 43
4.2. Analysis Setup 47
4.3. Evaluation of ERE Framework 48
4.4. Extension of ERE Framework 56
4.5. Summary 57

Chapter 5. Energy Management Techniques for Memory Resident Databases 59
5.1. Memory Resident Databases 59
5.2. System Architecture 63
5.3. Power Management Schemes 71
5.4. Experimental Evaluation of Memory Management Schemes 78
5.5. Summary 93

Chapter 6. Data-Driven Query Optimizations for Memory Resident Databases 94
6.1. Data Windowing 95
6.2. Query Restructuring 99
6.3. Experimental Evaluation 105
6.4. Summary 112
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Database Layouts for MEMS-Based Storage Systems</td>
<td>114</td>
</tr>
<tr>
<td>7.1</td>
<td>MEMS Device and Characteristics</td>
<td>116</td>
</tr>
<tr>
<td>7.2</td>
<td>Data Layouts for a Single Table</td>
<td>118</td>
</tr>
<tr>
<td>7.3</td>
<td>Data Layouts for Multiple Tables</td>
<td>124</td>
</tr>
<tr>
<td>7.4</td>
<td>I/O Performance Evaluation of Layouts</td>
<td>129</td>
</tr>
<tr>
<td>7.5</td>
<td>Discussion</td>
<td>140</td>
</tr>
<tr>
<td>7.6</td>
<td>Summary</td>
<td>141</td>
</tr>
<tr>
<td>8</td>
<td>Workload Characterization Framework for Data Mining Applications</td>
<td>144</td>
</tr>
<tr>
<td>8.1</td>
<td>Methodology</td>
<td>145</td>
</tr>
<tr>
<td>8.2</td>
<td>Data Mining Workload</td>
<td>148</td>
</tr>
<tr>
<td>8.3</td>
<td>Evaluation and Analysis Framework</td>
<td>156</td>
</tr>
<tr>
<td>8.4</td>
<td>Characteristics of Data Mining Applications</td>
<td>165</td>
</tr>
<tr>
<td>8.5</td>
<td>Summary</td>
<td>181</td>
</tr>
<tr>
<td>9</td>
<td>Architectural Acceleration of Data Mining Kernels</td>
<td>184</td>
</tr>
<tr>
<td>9.1</td>
<td>Data Mining Applications Mix</td>
<td>186</td>
</tr>
<tr>
<td>9.2</td>
<td>Workload Characteristics</td>
<td>187</td>
</tr>
<tr>
<td>9.3</td>
<td>Architectural Exploration</td>
<td>192</td>
</tr>
<tr>
<td>9.4</td>
<td>Evaluation of Acceleration</td>
<td>200</td>
</tr>
<tr>
<td>9.5</td>
<td>Summary</td>
<td>207</td>
</tr>
<tr>
<td>10</td>
<td>Conclusion</td>
<td>209</td>
</tr>
</tbody>
</table>
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Typical power consumption of MPC7410</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Typical power consumption of MSC8101</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>Software setup</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Benchmarks used in evaluating the heterogeneous setup</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Interpretation of ERE framework values</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>The two classes of queries considered for the experiments</td>
<td>81</td>
</tr>
<tr>
<td>5.2</td>
<td>SQL for organizer queries</td>
<td>83</td>
</tr>
<tr>
<td>6.1</td>
<td>Scenarios for organizer queries</td>
<td>106</td>
</tr>
<tr>
<td>6.2</td>
<td>Scenarios for TPC-H queries</td>
<td>106</td>
</tr>
<tr>
<td>7.1</td>
<td>Device parameters used in experiments</td>
<td>130</td>
</tr>
<tr>
<td>7.2</td>
<td>Summary of layout performance</td>
<td>142</td>
</tr>
<tr>
<td>8.1</td>
<td>Algorithms used in the study and their descriptions</td>
<td>151</td>
</tr>
<tr>
<td>8.2</td>
<td>Dataset for classification and ARM applications</td>
<td>161</td>
</tr>
<tr>
<td>8.3</td>
<td>SEMPHY datasets and their characteristics</td>
<td>162</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>8.4 Comparison of data mining application with other benchmark applications</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>8.5 Top kernels of data mining applications</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>9.1 Algorithms used in this study and their descriptions</td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>9.2 Kernels of the applications</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>9.3 Architectural performance of the kernels</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>9.4 Datasets used in the experiments</td>
<td>203</td>
<td></td>
</tr>
</tbody>
</table>
List of Figures

3.1 Heterogeneous architecture 29
3.2 Base power consumption of the board 34
3.3 Power, execution time, energy and energy-delay product for art 38
3.4 Power, execution time, energy and energy-delay product for g721 38
3.5 Power, execution time, energy and energy-delay product for bzip2 38
3.6 Power, execution time, energy and energy-delay product for jpeg 39
3.7 Power, execution time, energy and energy-delay product for pegwit 39
4.1 Design of ERE framework 44
4.2 ERE framework for scalable applications 50
4.3 ERE framework for pegwit 53
4.4 ERE framework for multi-stream applications 55
5.1 Energy consumption of a memory resident database 63
5.2 DBMS architecture 64
5.3 Banked memory architecture 66
5.4 Available operating modes and their resynchronization costs 68
7.4 Column layout for MEMS 121
7.5 Object layout for MEMS 123
7.6 Row-interleaved layout for MEMS 125
7.7 Column-interleaved layout for MEMS 126
7.8 Row-fill layout for MEMS 127
7.9 Column-fill layout for MEMS 128
7.10 Object-fill layout for MEMS 129
7.11 Experimental MEMS framework 131
7.12 I/O service times 132
7.13 Cache performance 138

8.1 Evaluation perspectives 148
8.2 Data mining taxonomy 149
8.3 Evaluation and analysis framework 157
8.4 Analysis flow 164
8.5 Measures of interest for evaluation and analysis 165
8.6 Automated classification of data mining, SPEC INT, SPEC FP, MediaBench and TPC-H benchmark applications 168
8.7 Multi-phased operations seen in data mining 171
8.8 Processor scalability of data mining applications 179
8.9 Data scalability of data mining applications 180

9.1 Design goals 185

9.2 Data mining system 194

9.3 Hardware logics for distance and min kernels 199

9.4 Hardware logic for the density calculation kernel 200

9.5 Applications for acceleration and their logics 201

9.6 Performance of accelerated kernels 204

9.7 Hardware sensitivity analysis of kernels 205

9.8 Application speedups 206
CHAPTER 1

Introduction

New technology has also unleashed its own flood of data. With the establishment of an extensive network infrastructure, including the Internet, corporate Intranets and virtual private networks, a reliance on electronic data to conduct both front and back office business is mounting. For instance, one of the leading business chains, Walmart, gathers information in the order of about 20 million customer transactions per day from all its establishments. Such high volume data collection is enabled through sophisticated data collection infrastructure, which includes both software tools and human resources. The gathered information is in turn used by Walmart to provide customized services (e.g., in Customer Relationship Management). But this piling up of information also brings forth brand new challenges in the computing world; in Walmart’s case, massive data processing capabilities are needed in systems in order to extract meaningful information (like customer purchase patterns) from such a flood of transactions in a timely manner.

In the mass storage industry, technology advancements have consistently delivered reduced cost per megabyte, increased storage capacity, and improved storage mechanisms. Thanks to these developments in storage technologies, today, there are gigabytes of photos, music, video and other personal data stored on systems. Personal computing paradigm has been steadily changing. A digital transformation is underway; anything
that can be digitally recorded is being recorded. Researchers assert that for each person on the planet roughly 800 megabytes of data are recorded annually. And the amount of that data is doubling every 18-24 months [Cor05]. Between 1999 and 2002 new data stored on electronic media grew by 30 percent, according to a recent report by the University of California, Berkeley. The same study says that print, film, magnetic, and optical storage media produced about 5 exabytes of new information in 2002 – that’s equal to nearly half a million new Libraries of Congress. In the near future, there will be terabytes of digital data that will require teraflops of processing power to manage, analyze, and synthesize meaningful information.

Improvements in storage and network technologies have also impacted the scientific community. Until a few decades ago, investigative methods in most scientific and engineering disciplines were either empirical or theoretical. Recently, computational science has emerged as a major branch in these disciplines. It is not uncommon to see collecting vast amounts of data to make forecasts and intelligent decisions about future directions. The National Science Foundation (NSF) spends more than 170 million dollars every project term to enable the collection and analysis of data pertaining to weather prediction.

The world’s largest commercial databases are approaching the 50TB mark, whereas the database sizes on hybrid systems are approaching the PB mark. Some of these large databases are growing by a factor of 20 every year. This does not even include the amount of data made available by millions of users on the Internet. In addition to the increasing amount of available data, there are two other major factors that make the
problem of information extraction particularly complex. The first challenge is the need for increasingly complicated algorithms to analyze and extract meaningful information from the large data sets. Second, in many cases, such analysis needs to be done in real time to reap the actual benefits. For instance, a security expert would strive for real-time analysis of the streaming video and audio data in conjunction. Managing and performing run-time analysis on such data sets is appearing to be the next big challenge in computing.

As the data trends suggest, there is an increasing need for automated data analysis tools to extract the required information and to locate the necessary data objects. There are numerous commercial, research and scientific tools that have been proposed, developed and currently being used. Researchers at the Jet Propulsion Laboratory apply neural-network based learning techniques to the images collected from Venus in order to identify volcanoes. Every computing community faces the need to develop new tools every day. Recently, the Department of Energy’s Oak Ridge and Pacific Northwest national laboratories have teamed up to learn about new aspects of biological research through the use of data-intensive computing capabilities. They are planning to develop tools that enable such capabilities. Such data intensive applications are characteristically different from traditional IT applications, which are targeted by current hardware and software systems. Just as graphics and multimedia applications have had tremendous impact on processors and systems, these data intensive data mining applications will have tremendous impact on future systems. Besides this fact, more sophisticated and
data-intensive application systems have emerged in the past from significant advances in processing power and performance.

As sheer volume of data is rising rapidly, the level of complexity in data type and in data analysis tools also arise. The deluge of data must not only be stored somewhere, but also be managed and processed efficiently. On one hand, it requires that algorithms and software is efficiently optimized. While on the other hand, even computing systems have to be redesigned and customized to efficiently host these data intensive applications. This requires innovative and “smarter” computing architectures. Such architectural innovations should strive to achieve new performance levels in data intensive computing on a scale not previously seen. Such innovations would enable high performance computing in the new era of digital transformations. This forms the core goal of this thesis work.

Future systems are bound to be data-intensive, and require significant design enhancements to alleviate several drawbacks. This research work addresses a section of these design challenges. It focuses on diverse topics in system design. The designs and optimizations proposed in this thesis take a three tier approach. This is basically a data-driven approach rather than control driven. There are three different kinds of data that are processed in current systems and which is bound to be seen in future systems. This includes streaming data, resident databases and massive datasets.

One of the diverse data types existing today is streaming data. It seen in a wide range of systems, ranging from hand held devices to high performance equipment. Features like video, text messaging, audio and voice are not uncommon in new cellphones. On
the other hand, streaming data is also seen in multiprocessor heterogeneous systems like communication base stations, switchboards, set-top boxes, sensor network servers and web servers. The first section of the thesis focuses on enabling high performance on such streaming setups. Besides performance, energy consumption is also a crucial issue in the design of such multiprocessor streaming systems. In Chapter 3, an application-driven task scheduling scheme is proposed for such systems. This scheme aims to improve the performance and concurrently reduce the energy consumption. The proposed scheme uses stream data as a driving model to optimally partition processor resources, and schedule application tasks.

In reality, achieving a good tradeoff between performance and energy consumption is a tedious task since it requires a thorough analysis of the system under consideration, which could be time consuming. Existing analysis tools and frameworks do not take into consideration the streaming nature of data or even the presence of constraints in heterogeneous multiprocessor systems. The existing metrics and measures lack the capabilities of highlighting the real performance and energy improvements in these systems. Chapter 4 addresses this fact and proposes a new analysis framework called Energy-Resource Efficiency (ERE). This framework systematically analyzes the performance, energy consumption and resource-efficiency of a system to highlight the various performance-energy tradeoffs. With a pre-defined performance and energy requirement, the ERE can also be used to find the minimal number of resources needed to achieve that desired configuration.
Another prominent data type is databases. Database is typically an organized collection of records having a standardized format and content. Computerized indexes and catalogs are some of the most common types of databases. Simple access schemes are seen in structured address books on hand held devices, whereas online library catalogs carry a complex indexing structure to access the database. Databases have evolved through ages. Access schemes are smarter and well optimized. A lot of optimizations have been done from the algorithmic and query perspectives. As databases are evolving, so is the data storage technology. Modern database techniques do not take into consideration the underlying storage technologies. This work uses this as a motivation to propose database access techniques that take into consideration all of the design factors, namely the databases, the query structures, the access patterns and the underlying storage system. Memory, a core storage component, has evolved in recent years to become more structured and partitioned, providing ample optimization opportunities. Chapter 5 uses this as a motivation to propose memory hardware schemes that smartly take into consideration the database system. That is, the memory tunes itself to the needs of the DBMS, thus enhancing the performance. This is achieved by taking advantage of the multiple modes of operation that are available in current memory systems. Chapter 6 goes one step further and optimizes the DBMS to take into consideration the smart allocation schemes in new storage architectures. Intelligent intra-query and inter-query restructuring schemes are proposed, which optimize queries to adapt to the changing
storage technologies of the system. Chapter 7 sets out to find answers to various questions that arise while storing a database in a new storage technology based on Micro-ElectroMechanical Systems (MEMS). MEMS based storage devices do not use rotating mechanical components like the ones seen in hard disks, and are known for lower cost, higher performance and increased reliability. This thesis proposes a mapping between application (high-level) database layouts and device (low-level) data layouts by taking advantage of the features associated with the underlying MEMS storage device.

As already mentioned, data is growing at an exponential rate. Databases are becoming larger by the day, and the paradigm of having structured records and structured query-based access to the database is no longer valid. These days, large datasets are common, data structures are complex, and access schemes are intricate. Sophisticated analysis tools are already in place. Especially, as the data sizes are exponentially increasing, the need to use complicated tools to extract information from the collected data becomes clear. Data mining, an automated method to extract useful information from large datasets, is becoming widely popular. Data mining programs have become essential tools in many domains including marketing, customer relationship management, scoring and risk management, recommendation systems, and fraud detection. In addition, data mining techniques have been adopted by various scientists to analyze the vast amounts of data representing real-world systems. Data mining techniques have been utilized in cosmology simulation, climate modeling, bioinformatics, drug discovery, and intrusion detection.
As the amount of data collected increases, there arises a need to utilize even more complicated data mining applications. However, one important obstacle that has to be addressed is the fact that the performance of computer systems is improving at a slower rate compared to the increase in demand for data mining applications. Recent trends suggest that the system performance (data based on memory and I/O bound workloads like TPC-H) has been improving at a rate of 10-15% per year, whereas, the volume of data that is collected doubles every year. Having observed this trend, researchers have focused on efficient implementations of different data mining algorithms. Another approach to address this problem is to develop an understanding of the characteristics of these applications. This information in turn can be utilized during the implementation of the algorithms and the design/setup of the computing systems. Understanding the architectural bottlenecks is essential not only for processor designers to adapt their architectures to data mining applications, but also for programmers to adapt their algorithms to the revised requirements of applications and architectures. In Chapter 8, several widely-used data mining algorithms from many domains are extensively studied by means of a new automated analysis framework. The framework is used to study data mining applications as a workload for future platforms. The workload characterization results are then used to design NU-MineBench, a benchmarking suite containing representative data mining applications. This is a major contribution of this work.

The characterization studies proved that data mining applications are very different from traditional applications (including DBMS). Data mining applications have multiple phases of compute and data intensive computations, which are called *kernels*. These
kernels are identified for every data mining application. Some of the kernels are common across data mining applications. Once these kernels are extracted, designing high performance schemes for data mining applications becomes an easy task. This is because kernels contribute to majority of the execution times. Hence, when the kernels are accelerated, the entire application also speeds up. This thesis also extends scalable high performance techniques to these kernels of data mining applications to achieve application speedups. Chapter 9 goes one step further and presents a new data mining system. This new data mining system has a reconfigurable hardware accelerator in addition to the traditional processor. The task of this accelerator is to accelerate the kernels of data mining applications. Kernels of applications are identified using automated analysis tools after extensive workload characterization. The results show that remarkable speedups are achieved when an accelerator based data mining system is used to execute the applications. Such speedups are not seen in traditional systems.

The rest of this document is organized as follows. The summary of all related work is presented in Chapter 2. Chapter 3 and Chapter 4 discuss the optimizations for streaming architectures. Chapter 5 discusses the new memory energy management schemes designed for databases. In Chapter 6, new data-driven multiquery optimizations are proposed. These optimizations take into the consideration the database and the storage technology underneath. Chapter 7 proposes new database access layouts for MEMS based storage systems. Workload characterization of data mining applications are covered in Chapter 8. Chapter 9 uses these characterization results to propose a new data mining systems architecture. The thesis work is concluded in Chapter 10.
CHAPTER 2

Background and Related Work

This chapter introduces the related work and background for the work presented in this document. This section is organized based on the chapter layout. First, it discusses architecture based techniques to enhance the performance and reduce the energy of data intensive systems. Several metrics and measures for evaluating and assessing system performance and power consumption are also discussed. Following this, the related work in the field of modern databases is then covered. The topics discussed include both hardware and query driven schemes that are proposed by researchers and widely used both in the industry and academia. Well known data layouts for existing and emerging storage architectures are also discussed. The last section covers high performance data mining systems. This includes even the limited data mining application characterization work that has been done in the past.

2.1. Architecture Optimizations

2.1.1. Memory Hierarchy

Memory is a very common candidate for low power design. Lebeck et al. proposed a power-aware page allocation scheme for the main memory \cite{LFZE00}. In this work, the different operating modes of RDRAM are exploited to improve the energy-delay
product. This dynamic scheme forms the basis for the proposed power-aware memory hierarchy. Another approach by Delaluz et al. introduced novel techniques to exploit the low-power operating modes of DRAMs [DKV+01]. These techniques include heuristics that use fixed thresholds for detecting idleness and an adaptive threshold that attempts to adjust itself with the dynamics of a program.

Significant work has been done on designing low-power caches. An integrated architectural and circuit-level approach by Se-Hyun et al. aims at reducing leakage energy in instruction caches [YPF+01]. They introduced DRI i-cache, an i-cache that can dynamically resize and adapt itself to an application’s requirements. Kamble and Ghose proposed architectural techniques to reduce power consumption in on-chip caches, and also provided analytical models for estimating energy dissipation of conventional caches as well as low-power caches [KG97, GK99]. Compiler and hardware-based approaches to reduce cache misses, by Sherwood et al. [SCE99], suggest dynamic reordering of pages in physically addressed caches. In [Alb00], Albonesi proposed an on-demand cache resource allocation policy called selective cache ways. Selective cache ways technique gives the ability to disable a subset of ‘ways’ in a set associative cache during periods of modest cache activity, while the full cache may remain operational for more cache-intensive periods. A mechanism called Cache Decay, proposed by Kaxiras and Hu [KH01], exploits generational behavior of caches to reduce cache leakage power. Bahar et al. have studied power-performance tradeoffs for several power/performance sensitive cache configurations, which involve techniques like increasing cache size or associativity and including buffers along side L1 caches [BAM98].
Brooks et al. proposed a framework for analysis and optimization of power at the architectural level [BTM00]. Wattch provides a power evaluation methodology within the portable and familiar SimpleScalar [Aus] framework. The proposed model for power estimation is based on the Wattch model.

One of the major challenges that the hardware industry faces today is to cope with the immense power requirements of integrated circuits. Most of the studies and proposals for minimizing the power requirements of computing systems are directed towards processors and peripherals. The power consumed by processor cores are decreasing drastically, leaving the memory hierarchy and other external components as the significant bottleneck. For instance, the memory hierarchy could consume up to 50% of the power for memory intensive applications, and the off-chip data bus could consume up to 20% of the power. Hence, there arises a need for optimizations that reduce the power consumption without affecting the performance of the system. This forms the motivation for the work done in [PC02] and [BCPK02].

In [PC02], a power-aware memory hierarchy is proposed. This power-aware memory hierarchy improves the system energy consumption and performance by 50%. A new layer called Energy-Saver Buffers is also introduced between the cache and memory subsystem. In [BCPK02], a new data bus protocol called Power Protocol is proposed. This protocol is very effective in reducing the traffic over the data bus by proposing a data cache called ”value cache” for the bus. As a result of caching, the energy and the performance improves by average 42%. These two work are mentioned here to emphasize the need for new architectural techniques that keep pace with emerging computing
requirements and design parameters. Thus, these work form the first layer (architecture) of the multi-layered model considered in this document.

2.1.2. External Data Bus

Most of the existing work in bus encoding aims towards improving the performance and effective bandwidth of the bus. For example, Citron and Rudolph have described a technique to encode data on the bus using a table-based approach [CR95]. In this method, they divide a given data item (to be transferred over the bus) into two parts. The lower part is sent without being encoded, while the upper part is encoded using a look-up table (LUT). Their method aims at reducing the size of the data being sent. The upper part of the data is broken into a key, which is used to search the data in the LUT, and a tag, which is used to match the data in the LUT to find if there is a hit. If there is a hit, the data sent over the bus is compacted to y [y=Comp(x)], where x is the original data and the lower d bits of y is the same as the lower d bits of x. On the receiving end, a component called bus expander expands y to original data x [x=Expn(y)]. It follows that x=Expn(Comp(x)).

In the recent past, the encoding paradigms for reducing the switching activity on the bus lines have been investigated (e.g., [SB97, MOI96]). Most of the common encoding techniques rely on the well-known spatial locality principle. An analysis of several existing low-power bus encoding techniques such as T0, bus inversion, and mixed encoding has been performed by Fronaciari et al. [FSS99]. Among the different techniques that have been studied for reducing the power dissipation of the address bus, Gray code
addressing appears to be a very efficient scheme [MOI96]. Shin, Chac, and Choi have explored a partial bus invert coding technique where they proposed a heuristic to select only portions of the bus to invert for further improving the effectiveness of the bus inversion coding [SCC98]. They have also proposed a combination of bus inversion with transitional signaling for a typical signal processing application.

Techniques using look-up tables to compress data have been explored by Bishop and Bahuman who have proposed an adaptive bus coding to reduce bit transition activity by exploiting the repetition of the data [BB01]. Childers and Nakra have studied the effect of reordering the memory bus traffic to reduce power dissipation [CN00]. Yang et al. [YZG00] have proposed the use of a compression cache to compress the data stored in the cache line to at least half their length so that a single cache line can effectively store two cache-lines of data. However, their main objective is to compress the data in the on-chip cache and increase the cache hit rate. Since their approach sends the data across the bus in a compressed form, it also results in a reduction in bus activity. Also, the compression strategies used in this work and in [YZG00] are entirely different. Farrens and Park [FP91] have presented a strategy that caches the higher order portions of address references in a set of dynamically allocated base registers. The bus energy optimization strategy is also similar to the strategy proposed by Yang and Gupta [YZG00].

2.2. Performance-Energy Tradeoff Frameworks and Tools

Power optimizations have been proposed for all the kind of systems. A survey of some of the prominent power optimization techniques for generic systems was done by
Benini and Micheli [BM00]. An energy-conscious reconfigurable system with DSPs has been explored by Wan et al. [WILR98]. The focus of this work is on the flexibility and the adaptability of reconfigurable processors in a heterogeneous environment. Tien-Chien-Lee et al. developed an instruction-level power model for DSPs and proposed some energy minimization techniques based on this model [TCLTMF97].

Due importance is given to power-performance tradeoffs by all power-aware system architects. In [BMWB00], Brooks et al. used a fast cycle accurate simulator to model power and to derive energy characterizations. Using the simulated data, they perform a detailed power-performance tradeoff study. Yang et al. performed a quantitative study on energy-efficient architectures and developed an integer linear program to model performance-power tradeoffs in such architectures [YGGT02]. By presenting some power evaluation mechanisms in [JM01], Joseph et al. consider a wide range of power-performance tradeoffs in a general purpose CPU. Vijaykrishnan et al. [VK1+00] undertook an integrated hardware-software approach to power reduction by applying power optimizations at the compiler level and studying their impact on some of the existing power-aware hardware technologies. They used a simulation-based tool for power estimation.

2.2.1. Evaluation Metrics and Measures

Estimation and evaluation of power consumption has become a crucial step in designing efficient computing systems. Significant work has been done on power estimation and evaluation for low-power embedded processors. Researchers have proposed
both analytical models and simulation-based methods for estimating power consumption. Kamble and Ghose provided analytical models for estimating energy dissipation of caches [KG97]. In Wattch [BTM00], Brooks et al. propose a framework for analysis and optimization of power at the architectural level. SimplePower [YVKI00] is a cycle-accurate RT level energy estimation tool developed by Ye et al. Cignetti et al. designed a tool for the estimation of energy consumption in PalmOS family of devices [CKE00]. This tool works on a set of empirically derived performance and energy values. These values are measured using a framework consisting of an oscilloscope, a power supply and an IBM workpad. In the work done by Joseph and Martonosi [JM01], the runtime power dissipation of different processor units is estimated. Power is modeled by associating it with a set of performance counters defined in the hardware.

The metrics that are used to measure performance-energy tradeoffs typically target uniprocessor architectures. Energy-delay product is a very well known metric used for such studies [HIG94]. Most of the work mentioned in the previous paragraphs use this metric. For low-end systems, even energy-per-operation [BB95] gives a rough estimate of the power-performance behavior. Some of the researchers also use custom frameworks to quantify performance-energy tradeoffs ( [YGGT02] for instance). In [ZS02], Zyuban and Strenski use a mathematical approach to introduce a new metric for representing speed-power tradeoffs. They consider various circuit power implications to design a new variable called hardware intensity, which is used for evaluating issues that affect both circuits and microarchitecture. The above metrics work well when applied to
uniprocessor architectures. Heterogeneous and parallel systems require special consideration due to the presence of multiple processing elements. Hence these metrics may not quantify the best available tradeoffs. This thesis work uses this fact as the motivation to develop a new metric called energy-resource efficiency ($ERE$).

### 2.3. Modern Databases

#### 2.3.1. Hardware Schemes for Databases

In the past, memory has been redesigned, tuned or optimized to suit emerging fields. Need for customized memory structures and allocation strategies form the foundation for such studies. Copeland et al. proposed SafeRAM [CKKS89], a modified DRAM model for safely supporting memory-resident databases alike disk-based systems, and for achieving good performance. In PicoDBMS [PBVB01], Pucheral et al. present techniques for scaling down a database to a smart card. This work also investigates some of the constraints involved in mapping a database to an embedded system, especially memory constraints and the need for a structured data layout. Anciaux et al. [ABP03] explicitly model the lower bound of the memory space that is needed for query execution. Their work focuses on light weight devices like personal organizers, sensor networks, and mobile computers. Boncz et al. show how memory accesses form a major bottleneck during database accesses [BMK99]. In their work, they also suggest a few remedies to alleviate the memory bottleneck. In [WKKS99], Weikum et al. propose a self-tuning memory management system for databases. They propose memory-based techniques that tune data storing and fetching to improve the performance. Bouganim et
al. develop a memory-adaptive scheduling technique for queries that perform memory-intensive operations [BKV98]. In their scheme, memory is both statically and dynamically managed based on the input query.

2.3.2. Data-Centric Query Optimizations

Memory-resident databases have been studied since a long time back by various researchers. In the past, memory technology bottlenecks have prevented the actual implementation of these databases. Currently, BerkeleyDB [Sle03], TimesTen [Tim03], DataBlitz [BBG+00], and Monet [Bon02] are a few known memory-resident databases.

An exhaustive study of indexing schemes for main memory database systems was done by Lehman and Carey [LC86]. In their work, they compare various index structures and propose T-trees by considering memory access time improvements, a deviation from the traditional way of optimizing for disk accesses. They also indicate how B trees are not suitable for memory databases due to the shallow nature of their tree nodes. An alternative, but complex, cache-conscious indexing called CSB+ trees has been proposed by Rao and Ross [RR00]. These trees have a reduced size (implies a lesser storage requirement) in the main memory due to their offset-based child nodes. Due to the simpler requirements, T-trees [LC86] and extendible hashing [RG02] are used in the databases for this work. In [Bon02], Boncz considered a set of basic operations on memory databases and profiled them based on the main memory access costs. They also propose a new storage scheme that stores entries in a space-saving format called Binary Association Table. A query optimization based on multi-level hashing is also
proposed for queries involving hash-join operation. Ross takes a bottom-up approach to optimize queries for memory-resident databases [Ros02]. In his work, query optimizations are first proposed by extensively studying the system, and then a cost model is used to study the effect of these optimized queries on a memory-resident database. Researchers have exhaustively proposed multi-query optimizations. A survey of some of the multi-query optimizations specific to memory databases has been done by Choenni et al. [CKdAS96].

Query optimizations have been designed specifically targeting modern cache architectures in a move to make database systems cache friendly. A cache-friendly sorting algorithm targeting RISC machines is proposed by Nyberg et al. in [NBC+94]. Shatdal et al. discuss techniques to improve the cache performance by reorganizing the order of execution of operations in a query based on highly-reused data blocks in the query [SKN94]. In [TT99], Trancoso and Torellas combine data blocking and software prefetching to achieve improvements in cache performance. Ailamaki et al. designed a new layout called Partition Attributes Across (PAX) [ADH02]. This scheme targets improvements in cache performance by extracting the best of slotted and decomposed storage schemes.

The data windowing technique proposed in this thesis takes a data-centric approach to improve the performance of cache used in memory-resident databases. The approach is unique in the sense that the proposed technique optimizes operations based on the underlying data. Operations are reordered to suit the data brought from a table, and do not emphasize on the control or the flow of the multiple queries. In other words, the
proposed technique does not base the optimizations purely on data dependencies and control-flow of queries. Nor is the table layout changed. Contrastively, every bit of data read from the underlying tables is maximally reused. The queries are reorganized to maximally reuse every block of the data brought to the cache.

2.3.3. Query Restructuring Techniques

An et al. analyze the energy behavior of mobile devices when spatial access methods are used for retrieving memory-resident data [AGS+02]. They use a cycle accurate simulator to identify the pros and cons of various indexing schemes. In [AG93], Alonso et al. investigate the possibility of increasing the effective battery life of mobile computers by selecting energy efficient query plans through the optimizer. Although the ultimate goal seems the same, their cost plan and the optimization criterion are entirely different from the schemes proposed in this research work. Specifically, their emphasis is on a client-server model optimizing the network throughput and overall energy consumption. Gruenwald et al. propose an energy-efficient transaction management system for real-time mobile databases in ad-hoc networks [GB01]. They consider an environment of mobile hosts. In [MFHH03], Madden et al. propose TinyDB, an acquisitional query processor for sensor networks. They provide SQL-like extensions to sensor networks, and also propose acquisitional techniques that reduce the power consumption of these networks. It should be noted that the queries in such a mobile ad-hoc network or a sensor environment is different from those in a typical DBMS. This has been shown by Imielinksi et al. in [IVB94]. The research work discussed in this document bases its techniques on a
generic banked memory environment and supports complex, memory-intensive typical database operations. There are more opportunities for energy optimizations in generic memory databases, which have not yet been studied completely. The approach proposed in this work is different from prior energy-aware database related studies, as the focus is on a banked memory architecture, and use low-power operating modes to save energy.

Gassner et al. review some of the key query optimization techniques required by industrial-strength commercial query optimizers, using the DB2 family of relational database products as examples [GLSW93]. This chapter provides insight into design of query cost plans and optimization using various approaches. In [Man02], Manegold studies the performance bottlenecks at the memory hierarchy level and proposes a detailed cost plan for memory-resident databases. The cost plan and optimizer used for this thesis research work mimics the PostgreSQL model [Dat01, Fon]. It is chosen due to its simple cost models and open source availability.

A query restructuring algorithm is proposed by Hellerstein in [Hel98]. This algorithm uses predicate migration to optimize expensive data retrievals. In [CS99], Chaudhuri et al. extend this approach to study user-defined predicates and also guarantee an optimal solution for the migration process. Sarawagi et al. present a query restructuring algorithm that reduces the access times of data retrieval from tertiary databases [SS96]. Monma et al. develop the series-parallel algorithm for reordering primitive database operations [MS79]. This algorithm optimizes an arbitrarily constrained stream of primitive operations by isolating independent modules. However, the work proposed in the thesis is different from all of the above work in the sense that queries are reordered for
reducing energy consumption. Moreover, the database considered is memory-resident, with the presence of banked memory, which gives one more freedom for optimizations.

2.4. Data Layouts for Emerging Storage Devices

Modeling of MEMS-based storage was the first step towards understanding the performance characteristics of these devices. Griffin et al. [GSGN00a] and Madhyastha et al. [MY01] use the physical device mechanics of MEMS to study and model the performance of such probe-based devices. They derive equations that represent the time a MEMS device takes to service a basic I/O request. Researchers have built on this basic model to study MEMS from various perspectives.

As mentioned earlier, Griffin et al. study the possibility of operating system management of MEMS devices [GSGN00b]. There, they also provide an analogy between traditional disks and MEMS devices. Scheduling of I/O requests similar to disks was the immediate extension, and scheduling for MEMS devices has been studied in various formats [GSGN00b, UMA03, YAA03b]. Lin et al. study the power consumption of MEMS and propose several power conservation strategies based on the device characteristics [LBLM02]. These strategies cover the scheduling as well as the layout aspects. The layout optimizations that they propose are very dynamic and are applicable even to the proposed layouts. An optimized data placement algorithm based on the seek time of MEMS devices is proposed by Peterson et al. in [PBL02]. Detailed analysis of the seek time for various access patterns is also discussed in this work. However, a database is not considered. Yu et al. present a methodology to map a relational table from a
database onto a MEMS device \cite{YAA03a}. In this work, a data layout is derived based on existing page layouts for disk-based systems \cite{ADH02,CK85,RG02}. An exhaustive study of data page layouts for disk-based systems has been done by Ailamaki et al. in \cite{ADH02}. In their work, the performance is quantified from both the processor and memory hierarchy aspects.

The data layouts for MEMS proposed in this work is different from all the above work. It proposes multiple layouts for a database. A benchmark database with multiple queries is considered. In \cite{YAA03a}, a single layout is proposed for a custom table and a scan operation was illustrated on that table layout. The approach studies many primitive operations (like select, join) and also various query types. The multiple table case is also considered. The approach is very exhaustive in studying the operations. The mapping between database access patterns and the underlying device characteristics forms the key to the proposed approach.

2.5. Data Mining Application Analysis and System Design

2.5.1. Workload Analysis

Characterization of data mining applications is the key to understanding their inherent characteristics. In the past several system level characterizations of data mining algorithms have been performed to reveal the major characteristics and bottlenecks of these applications from a systems architecture perspective \cite{BF98,CC02,LPL+04,TW03}. Only individual applications are studied in these characterization studies. An application is studied in detail with respect to the pure application execution performance.
on the system. In fact, none of the system bottlenecks are actually highlighted in these work. On the other hand, studies such as [BGB98, KQH98] focus on analyzing specific components of the system and their related bottlenecks (in this case, the memory subsystem). In the characterization work attempted in this thesis work, each application is characterized to understand both the functional and architectural bottlenecks in order to design an accelerator to alleviate the bottlenecks. Moreover, all performance characteristics are studied from multiple aspects, not just purely from a system or application perspective. A new benchmark called NU-MineBench is also introduced, which has never been attempted before by other researchers. Characterization of data mining applications is a challenging task given the complex nature of these applications.

2.5.2. Hardware Schemes for Data Mining

Hardware optimizations for data mining applications have been proposed in the past. In [HNO98], Hayashi et al. propose using a PRAM to perform the k-Merge process, a conventional method of performing data mining. A clustering algorithm based on k-Means methodology was mapped on to a reconfigurable hardware logic by Estlick et al. [ELTS01]. The above work require algorithmic modifications to the code, which is not needed in this proposed work. A similar approach to using configurable logic has been applied to apriori algorithm by Baker et al. in [BP05]. In their work, they accelerate the search function of apriori, which involves a specialized case of pattern matching. Another pattern matching (pure string matching) hardware has been proposed by Zhang et al. [ZCI+04]. They apply it to certain string-matching based data mining
applications. The proposed scheme is different from all the above work. This work focuses on proposing logics based on the inherent kernels, which are actually identified after extensive characterization at multiple levels and not based on pure algorithmic analysis. The kernels and logics are generic, and are defined so that they are not specific to just one application, unlike apriori or k-Merge logics. The proposed methodology and design is applicable for any data mining algorithm (in fact, without any modifications to any distance, variance and density based algorithms).
CHAPTER 3

Performance-Energy Optimization Techniques for Multiprocessor Systems

Streaming systems, like media processors and today’s world-wide-web, will be a key component in the future information structure. Developing an understanding of this and the key challenges blocking it, namely reliability and scalability, is essential for the growth of this service and its availability. This chapter considers the challenges arising in such systems and then presents new operating system and compiler techniques for such systems. These techniques emphasize the need for smarter scheduling, resource allocation and driver strategies required in modern compilers and operating systems. This chapter illustrates system software (and hardware) aspects that need not be controlled explicitly by a user and instead allows automated control. It also discusses the need for improved metrics and analysis frameworks to judge the performance, reliability, availability and scalability of such systems.

3.1. Multiprocessor and Heterogeneous Streaming Systems

Embedded computing devices are becoming a part of everyday lifestyle. Smart cards, vending machines, TV set-top boxes, personal data assistants (PDAs), mobile phones, and industrial automation equipment are just some common embedded devices.
Each of these devices is a congregation of various emerging technologies. For instance, devices like set-top boxes and mobile phones are a result of the merger of technologies like broadband communications or 3G wireless networks with interactive multimedia. Such embedded devices support numerous functions like multimedia (MP3, MPEG2 media playback), wireless communication (GSM, digital radio, Bluetooth) and some mandatory functions including user interfaces and file management, to be carried out at the same time. The data handled by such systems are also increasing at a tremendous rate. Even further, newer standards such as MPEG4, WMV and JVT (H.26L) require these devices to have a high performance in order to handle the data flow in real time. As a result, system designers are opting to embed more than one processor or processor core into these devices in order to satisfy both multitasking and performance needs \cite{EE01,HNO97,Mot03,PSH+01}. Traditional parallelization techniques can also be extended to these systems by schematically distributing a given task to multiple processors to achieve performance gains.

Besides performance, system designers also have to focus on the power consumption of embedded systems. Power dissipation has been a crucial issue in the design of embedded systems. The more the number of processors in a device, the higher is the power dissipation of that device. Hence, parallelization techniques need to be modified to take this factor into consideration during the compilation (in case of auto-parallelizing compilers) and scheduling phases. This provides the motivation for the rest of this chapter. In the following sections, a parallelization scheme that aims to improve the performance and concurrently reduce the energy consumption of a system is presented.
3.2. Approach

The proposed scheme targets a multi-resource heterogeneous embedded system consisting of low-power embedded processors that serve as computing resources and a general-purpose conventional processor that acts as the master controller. The performance of the system is enhanced by fully utilizing the available computing resources to execute an application. For this, some of the existing parallelization techniques are extended to the given heterogeneous platform. The master splits a given task among the available resources and these resources do the work in parallel. A mechanism to reduce the energy consumption is introduced by applying low-power optimizations in tandem with the parallel techniques. These low-power optimizations are implemented at the application level, and utilize the low-power modes offered by the underlying processors to reduce the overall energy consumption. Then, the behavior of the system is experimentally studied and characterize when these application-controlled schemes are supported. The proposed schemes are effective in improving both the performance and energy consumption.

The rest of the chapter is organized as follows. In the following section, the experimental setup is discussed. It includes the architecture used, and the power methodologies that were developed. Then, the methodology for parallelization is elaborated followed by discussion of the experimental results.
3.3. Heterogeneous Platform

3.3.1. Host/Slaves Architecture

A heterogeneous computing environment with DSPs and PowerPC forms the backbone for all the experiments. A Motorola PowerPC chip, MPC7410 \[\text{Mot02b}\] is the master controller and a farm of twelve Motorola 16-bit DSP processors (MSC8101 \[\text{Mot01b}\]) are the slaves. Each of the slave processor acts as a processing element. The block diagram for this setup is shown in Figure 3.1.

The MPC7410 is an implementation of the PowerPC (also referred to as PPC) family of RISC microprocessors by Motorola. The MPC7410 has six functional units including the AltiVec technology. MPC7410 operates at 400MHz (1.8V). The MPC7410 has a unified, 32KB, 8-way set-associative, on-chip L1 cache. The L2 cache is implemented with an on-chip, two-way, set-associative tag memory, and with internal, synchronous SRAMs for data storage.
Each processing element, the MSC8101 chip, is an one-chip integration of five major components:

- high-performance and low-power 300MHz DSP SC140 StarCore with 1.5V as core voltage
- on-device SRAM memory (0.5 MB)
- flexible System Interface Unit
- Communications Processor Module (CPM) based on PowerQUICC II (MPC8260)
- 16-channel DMA controller

The setup involves a distributed DSP architecture connected through the 16-bit parallel host interface (shown as HI in Figure 3.1). The host (MPC7410) controls the bank of MSC8101s. The host also coordinates the DSP resource allocation. Each slave DSP completely manages its own subsystem resources.

### 3.3.2. Protocol for Host/Slaves Model

The protocol of interaction between the PPC and DSPs is designed after considering the specific design needs. The simplest form of protocol is as follows:

- If the PPC is in any low-power mode, it is first reactivated.
- The DSPs are in *wait* mode awaiting signals from PPC.
- The PPC sends a request for download signal to the required number of DSPs based on the program. The code is then downloaded to the DSPs when they become ready.
- The DSPs acknowledge the download.
- The program data is now supplied to the DSPs through multiple streams of data and acknowledgements going back and forth between PPC and DSPs. Once the data is downloaded completely on a DSP, a program segment is ready to run on that DSP.
- The participating DSPs run the program segment in parallel and return the resultant data back to the PPC.
- Finally, the program completes when all the expected output data reaches the PPC.
- The PPC and hence the DSPs go to a low-power mode if idle further.

3.3.3. Power Management Strategy

A centralized power-management module is designed to handle the idle states of the system. This centralized module executing on the PPC, manages the power modes of both PPC and DSPs. By existing as an arbiter between the hardware and software layers, this module enables an application to set the entire system to various low-power modes.

The MPC7410 PPC, offers three programmable power modes, namely doze, nap and sleep. These modes are enabled by setting certain registers [Mot02a]. The typical power consumed by various power modes of MPC7410 is shown in Table 3.1.

The proposed centralized power-management module puts the master PPC to doze mode whenever there is no communication with the DSPs, or if the PPC is not controlling any task. Even after assigning a task to the DSPs, the PPC remains in a low-power
Table 3.1. Typical power consumption of MPC7410

<table>
<thead>
<tr>
<th>Mode</th>
<th>Power (watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Power</td>
<td>6.6</td>
</tr>
<tr>
<td>Doze</td>
<td>3.6</td>
</tr>
<tr>
<td>Nap</td>
<td>1.35</td>
</tr>
<tr>
<td>Sleep</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 3.2. Typical power consumption of MSC8101

<table>
<thead>
<tr>
<th>Mode</th>
<th>Power (watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Power</td>
<td>0.5</td>
</tr>
<tr>
<td>Wait</td>
<td>0.25</td>
</tr>
<tr>
<td>Stop</td>
<td>0.17</td>
</tr>
</tbody>
</table>

receive state waiting for communication from DSPs. That is, the proposed module shifts the PPC to the doze mode and then listens for any external event (data from DSP) or interrupt to break the idle state, after which the PPC returns to full-power mode. Doze mode is chosen as the low-power mode for the master controller since this mode enables the functioning of the core units, while still offering a significant reduction in power consumption (and hence return to full-power mode is faster than it is from nap, sleep modes).

The MSC8101 DSP chip supports two low-power standby modes, namely the wait and stop modes [Mot02c]. The typical power consumption of MSC8101 for various modes is shown in Table 3.2. Wait state is the preferred low-power mode for DSPs in
Table 3.3. Software setup

<table>
<thead>
<tr>
<th></th>
<th>PPC</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chip Name</strong></td>
<td>MPC7410</td>
<td>MSC8101</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>VxWorks 5.4</td>
<td>-</td>
</tr>
<tr>
<td><strong>Development Environment</strong></td>
<td>Tornado 2.0 (IDE)</td>
<td>GreenHills Multi 2000 (IDE)</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>Cygnus 2.7.2 (gcc)</td>
<td>Optimizing C Compiler</td>
</tr>
</tbody>
</table>

the experimental analysis. If the DSPs are idle at any point of time, the module running on PPC instantly transfers them to the wait state.

### 3.3.4. Experimental Setup

The PowerPC operates at 1.8V and each of the MSC8101 DSP processors at 1.5V. A HP34401A multimeter [Agi01] connected across a small resistance (0.972 Ohms), is used for measuring the voltage, current and hence, the power consumption of the board. The HP34401A automatically adjusts the range to the characteristic being measured. The default power mode for the MPC7410 PPC, is full-power mode and for the MSC8101 DSP, it is full-power mode (of DSP). The board boots up and stabilizes to consume 15 watts of power (Figure 3.2), which is attributed to 1 PPC, 12 DSPs, system bus and the external memory. Moreover, none of the power optimizations are turned on. Table 3.3 describes the development environment used in experiments.
3.4. Methodology

Typical parallel applications share the given work with multiple processing elements. These processing elements are homogenous or heterogeneous computing resources. The work is distributed by breaking it down into simpler tasks either through pre-defined (static) or dynamic partitioning schemes. The partitioning aims to achieve improvements in performance, in other words, speedup in the application. Power-aware applications aim to control the energy consumption by taking into consideration both the program flow and the underlying hardware constraints. At the application level, it is easier to understand the hardware needs of a program. This is more relevant in the context of heterogeneous computing environment, where applications get partitioned to be run on various platforms and architectures.
A set of embedded benchmark applications are first studied and parallelized targeting the underlying heterogeneous setup. These applications are executed to study the impact of parallelization on the performance. Then, the impact of these parallel techniques on the energy consumption of the heterogeneous setup is studied, and concurrently a power-aware application is designed that reduces the overall energy consumption.

Table 3.4 shows the benchmarks used for the experiments. The categorization of the benchmarks will be evident from the conclusion section. Each benchmark is partitioned in a way that allows the core segments to be run in parallel, to ensure maximum utilization of DSPs. In the considered heterogeneous setup, the master controller partitions the code and assigns the various fragments to available DSPs. These DSPs run the code fragments in parallel. There are various techniques of power reduction that can be applied here. For instance, after assigning the work to DSPs, the master can be manually switched to a low-power mode if it remains idle. The unused DSPs can be put to a low-power mode. Also, the DSPs that execute a job can go to a low-power mode once the task gets completed, provided there are no more pending tasks. These schemes are already installed in the centralized power-management module (Section 3.3.3) that was designed for handling idleness. Each benchmark application utilizes this centralized controller to enable low-power modes. Both high-performance and power-aware techniques are integrated into the same partitioning algorithm. The following part elaborates the partitioning strategy used for implementing each benchmark in the heterogeneous system.
(1) After reading the input data, the PPC splits the raw data into blocks of static size. Each block is assigned a pending status. The PPC assigns one of the pending blocks to each DSP that is participating in the execution.

(2) The DSPs then work on their respective blocks in parallel. Once the computation starts, the PPC switches itself to *doze* mode if idle.

(3) When a DSP finishes working on its block, it replies to the PPC with its output. The PPC converts the status of the received block from pending to completed. If there is more work left to be done (i.e., any more blocks with pending status), the returning DSP takes another block to work on.

(4) If there are no more pending jobs to take, the returning DSP switches to *wait* mode.

(5) The PPC finally assembles the output data when all involving DSPs complete their execution.

In steps 2 and 4 of the above scheme, the low-power modes (*doze* for PPC, *wait* for DSP) are enabled by invoking the centralized power-management controller of Section 3.3.3. To recall, the controller already has schemes defined for remotely enabling low-power modes for the entire system.

### 3.5. Performance-Energy Analysis

The execution time (delay) and the power consumption is measured for each benchmark application. To determine the effectiveness of the new schemes, the impact of parallelization on the energy and energy-delay product (EDP) is also evaluated. For
Table 3.4. Benchmarks used in evaluating the heterogeneous setup

<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmark</th>
<th>Explanation</th>
<th>DSP Code Size (KB)</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Stream</td>
<td>art</td>
<td>Neural network based pattern recognition algorithm</td>
<td>26</td>
<td>10 KB image, 600 KB weight</td>
</tr>
<tr>
<td>Scalable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bzip2</td>
<td>Data compression</td>
<td>36</td>
<td>12 MB data</td>
</tr>
<tr>
<td></td>
<td>jpeg</td>
<td>Image compression</td>
<td>21</td>
<td>10 MB image</td>
</tr>
<tr>
<td>Single Stream</td>
<td>pegwit</td>
<td>Public key encryption</td>
<td>40</td>
<td>220 KB data</td>
</tr>
<tr>
<td>Non Scalable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-stream</td>
<td>g721</td>
<td>Voice compression</td>
<td>23</td>
<td>296 KB voice data</td>
</tr>
<tr>
<td>(12 Streams)</td>
<td>FIR</td>
<td>Complex FIR filter</td>
<td>17</td>
<td>N = 15000, Tap = 1000</td>
</tr>
</tbody>
</table>

Each benchmark application, there is an initial part of hand-shaking between PPC and DSPs, followed by the download of code to the DSPs. This is a prelude to each of the presented algorithms and the results presented in this section take these phases into account as well. Moreover, the presented results are an average over many runs (typically 4) since communication is involved.

Figure 3.3 to Figure 3.7 show the performance for the benchmark applications. These were prominent and representative. For all applications, the power consumption increases as more DSPs are brought into the system. For the art, g721 and bzip2 benchmarks, the execution time scales down linearly on designating the work to increasing number of DSPs [Figure 3.3(ii), Figure 3.4(ii), Figure 3.5(ii)]. Furthermore,
it is evident that the improvements in execution time surpass the moderate increase in power consumption. This implies that the system is very energy efficient for these algorithms. The average energy consumption decreases by 87% when 12 DSPs are used as against 1 DSP [Figure 3.3(iii), Figure 3.4(iii), Figure 3.5(iii)]. For jpeg, as seen in
Figure 3.6. Power (i), execution time (ii), energy (iii) and energy-delay product (iv) for the application jpeg.

Figure 3.7. Power (i), execution time (ii), energy (iii) and energy-delay product (iv) for the application pegwit.

Figure 3.6(ii), the speedups in execution times do not scale very well with the number of DSPs (even though the execution times decrease when multiple DSPs are used). This is due to a tremendous amount of communication between the PPC and DSPs. This, in turn, has a direct impact on the energy consumption. Moreover, the overall gains in energy are much less when compared to the art, g721 and bzip2 benchmarks. The execution times have a similar effect on the energy-delay product as well [Figure 3.6(iv)]. pegwit is an algorithm that is very hard to parallelize and also not computationally intensive. The performance deteriorates when more than 2 DSPs are involved [Figure 3.7(ii)]. This arises due to the increased communication overhead that can be avoided if fewer DSPs are used. Power consumption also increases. The worsened power and execution
times consequently increase the energy consumption too. There are no improvements in execution times, energy and energy-delay product beyond 2 DSPs.

3.6. Overall Observations and Shortcomings of Traditional Metrics

Scalable parallel algorithms have the potential to reduce energy consumption, besides improving performance. Additionally, for an algorithm that is hard to parallelize or that is not scalable (like pegwit, jpeg), it is better to do the computation without much communication, i.e., with fewer DSPs. The presented results show that by combining low-power optimizations with existing parallel techniques, one can achieve sizable improvements in both energy and performance of multi-resource heterogeneous systems.

Typically, one would be interested in studying the effect of performance improvements on the energy consumption and vice versa. Therefore, a metric that highlights performance-energy tradeoffs is essential in an analysis framework. An existing metric to study performance-energy tradeoffs is energy-delay product (EDP), which takes into account the delay and the energy consumption of a system. In the graphs in experimental result section, the EDP follows the trend of delay (execution time). Decreasing trends in EDP indicate good savings in both performance and energy. EDP is good at capturing overall trends. It is difficult to clearly say from EDP the mutual impact of performance and energy variations on one another. Moreover, in a multi-resource environment similar to the considered heterogeneous setup, the number of resources that
are used to achieve any savings also need to be considered during evaluation. This factor is not included in EDP. These shortcomings and special requirements serve as the motivation to introduce a new framework of analysis in Chapter 4.
In Chapter 3 it was shown how scalable parallel algorithms have the potential to reduce energy consumption, besides improving performance. Also, it was shown how existing metrics lack the capabilities of highlighting the real performance and energy improvements in parallel, distributed and heterogeneous systems. In reality, achieving a good tradeoff for heterogeneous, parallel and distributed systems is a tedious task since it requires a thorough analysis of the system. This might be time consuming, as there is no methodical way to co-analyze both performance and energy consumption for such systems. Conclusions derived using existing metrics and frameworks might not be accurate or even representative of the system since these systems use diverse computing resources. Hence, new methods need to be developed to study such systems. This is used as a motivation in this chapter to propose a new framework of analysis for computing systems.

An analysis framework with a new metric named Energy-Resource Efficiency (ERE) is proposed. This framework systematically analyzes the performance, energy consumption and resource-efficiency of a system to highlight the various performance-energy tradeoffs. With a pre-defined performance and energy requirement, the ERE can also
be used to find the minimal number of resources needed to achieve that desired configuration. Through experiments, the effectiveness of the ERE framework is verified. This also illustrates how to apply ERE to analyze various scenarios. This is achieved by incorporating certain optimization techniques meant to improve both performance and energy consumption into a multi-resource heterogeneous system. The effectiveness of these techniques is studied using ERE as the evaluation framework. The traditional evaluation techniques and metrics are also considered to highlight the differences and the advantages of ERE in studying performance-energy tradeoffs.

The rest of the chapter is organized as follows. Section 4.1 derives and explains the ERE framework. Section 4.2 presents the experimental setup. The ERE framework is evaluated in Section 4.3. Section 4.4 provides some intuitive inferences and the advantages of ERE over traditional frameworks, which is followed by conclusion remarks.

4.1. ERE Framework

By widely extending some existing mechanisms and adding a new metric, this section presents a more relevant evaluation framework for heterogeneous systems. The primary entity that drives the framework is a new metric called Energy-Resource Efficiency (ERE). ERE illustrates the performance-energy tradeoffs by concurrently considering the performance improvements, energy savings and resource-efficiency of a system. ERE is defined as the measure of the tradeoff that occurs between the energy

\[ ERE = \{ S_N, \eta, \Delta, ERE \} \]

\(^1\)ERE is a metric whereas ‘ERE framework’ refers to the entire \( S_N, \eta, \Delta, ERE \) framework.
consumed by an application and the amount of resources needed to achieve any reduction in energy consumption.

A heterogeneous computing environment offers a variety of processing elements, each of which can be of a different class (type). With such a mixed configuration, one needs to identify a set of processing elements (referred to as resources) best suited for code execution. ERE identifies the advantages of using a set of resources through relative comparison between configurations. Let us assume 1 resource of type $i$ as the base configuration. The advantage (or disadvantage) of using a new configuration with $N$ resources of type $j$ is easily measured using $ERE$ defined as follows:

$$ERE = \Delta \cdot \eta$$

where, $\Delta$ is the fraction of the energy saved on using the new configuration as compared to the base configuration, and $\eta$ is the corresponding normalized efficiency of utilization.
of resources.

\[ \Delta = \frac{E_{1i} - E_{Nj}}{E_{1i}} \]  

where, \( E_{1i} \) is the energy consumed by the application when run on a single resource of class \( i \) (the base case) and \( E_{Nj} \) is the energy consumed by the same version running on \( N \) resources of class \( j \) (the new configuration).

\[ \eta = \frac{S_{Nj}}{\alpha_j \cdot N} \]

where, \( \alpha_j \) is the scaling factor of class \( j \) over class \( i \) and \( \alpha_j > 0 \). \( S_{Nj} \) is the speedup, defined as follows:

\[ S_{Nj} = \frac{T_{1i}}{T_{Nj}} \]

where, \( T_{Nj} \) is the execution time of the new configuration and \( T_{1i} \) is that of the base configuration.

The scaling factor (\( \alpha_j \)) normalizes the efficiency (\( \eta \)) by annulling the skew between various types of processors. \( \alpha_j \) can be any “skew” based on an analyst’s preference or weightage. \( \alpha_j \) is computed by statistical methods (Section 4.2). One can opt to define \( \alpha_j \) as the ratio between benchmark values of two processor-classes from the same standard benchmark too (may be SPEC \cite{Sta01}, BDTI \cite{Ber94}, Dhrystone \cite{Wei84} mark ratios). In this case, \( \alpha_j \) becomes very sensitive to the architecture, their respective scores, and the deployed benchmarking scheme.
The base case is when $j = i$, $N = 1$ and $\alpha_j = 1$ (no skew). This follows that $S_{1i} = 1$, $\eta = 1$, $\Delta = 0$ and $ERE = 0$ for the base case. Table 4.1 shows how one can judge the performance using the new framework. It is possible that $\eta > 1$, which arises due to factors like differences in architectures (e.g. cache layout, processor speeds) and latency-improvement mechanisms (e.g. cache and memory optimization algorithms).

The ERE framework equations consider 1 resource of type $i$ as the base case. Towards the end of experimental results (Section 4.4), discussions illustrate how ERE can be extended to use any configuration for the base case and also to study any combination of processing elements and configurations.

$ERE$ represents the tradeoff between performance and energy by relating efficiency and energy. The rationale behind choosing efficiency is because of its ability to represent real performance improvements by bringing in resource availability. Efficiency decreases if performance does not scale well. $ERE$ reflects this as a penalty in tradeoff (since in equation 4.1, $ERE$ multiplies $\eta$ and $\Delta$). Intuitively, $ERE$ strives to reflect negative and positive effects of resource-utilization on energy, and viceversa.

The advantage of using the ERE framework lies in an integrated analysis. The analysis spans over all aspects of the system. For improving performance, one can lay

<table>
<thead>
<tr>
<th>Good Result</th>
<th>Bad Result</th>
<th>Impact on</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{Nj} &gt; 1$</td>
<td>$0 &lt; S_{Nj} &lt; 1$</td>
<td>Speedup</td>
</tr>
<tr>
<td>$\eta \to 1$</td>
<td>$\eta \to 0$</td>
<td>Efficiency</td>
</tr>
<tr>
<td>$\Delta &gt; 0$</td>
<td>$\Delta &lt; 0$</td>
<td>Energy Improvement</td>
</tr>
<tr>
<td>$ERE &gt; 0$</td>
<td>$ERE &lt; 0$</td>
<td>Performance-Energy Tradeoff</td>
</tr>
</tbody>
</table>
emphasis on $S_{Nj}$. Whereas, for understanding the benefit of resource addition, $\eta$ proves to be a good estimate. An energy-aware design can be based on $\Delta$ as the deciding factor. $ERE$ can be used to study both energy and performance, and hence its mutual impact. Typically, looking at just $S_{Nj}$ and $\Delta$ would suffice. But there is a limitation to that. $S_{Nj}$ and $\Delta$ cannot capture the real benefit of using a resource, unlike the efficiency ($\eta$). $\eta$ considers both the speedup achieved on that resource and the amount of such resources needed to achieve that speedup. Consequently, $ERE$ is a better measure as it considers both the real performance improvement (from $\eta$) and the energy-savings. By looking at the entire $ERE$ framework ($S_{Nj}, \eta, \Delta, ERE$), one can decide the tradeoff that needs to be set between performance and energy consumption, without sacrificing on either of them. Alternatively, one can derive the minimal amount of resources needed to achieve a desired performance and energy behavior by using $ERE$. A broad design layout and flow of the $ERE$ framework is shown in Figure 4.1. Some schemes to evaluate and quantify the various measures in the $ERE$ framework, and to derive useful conclusions from them, are presented in the succeeding sections.

4.2. Analysis Setup

To have an efficient implementation and relative results for comparison purposes, the heterogeneous setup of Chapter 3 is reused, which includes the experimental setup and the benchmark applications. The only new addition is the definition of $\alpha_j$. $\alpha_j$ is defined as the relative fitness of the performance of one processor type ($i$) over the other ($j$). The considered setup involves two classes of processors, $PPC$ and $DSP$. 1 DSP is
the base case for the experiments (that is, \( i = \text{DSP}, j = \text{PPC} \)). \( \alpha_{\text{PPC}} \) is an estimate of the number of DSPs required to achieve a performance comparable to that of 1 PPC. Non-linear regression analysis (tool from http://www.graphpad.com) is used for estimating this. Applications from MediaBench [LPMS97] and SPEC CPU2000 [Sta01] suites were executed on a varying number of processors of type DSP and one processor of type PPC independently. For each application, a fitting curve is projected over the performance points and an equivalent number of DSPs for 1 PPC is deduced from the curve. The geometric mean of \( \alpha_{\text{PPC}} \) over all MediaBench and SPEC CPU2000 benchmarks is 4.43 (average error of fitness \( \sim 0.05 \)). Also, \( \alpha_{\text{DSP}} = 1 \) always since DSP is the base. To minimize the error in values, \( \alpha_{\text{PPC}} \) is estimated for each application in the experiments.

### 4.3. Evaluation of ERE Framework

#### 4.3.1. Methodology

A heterogeneous architecture would undoubtedly support multi-stream applications since each stream can potentially be handled by a resource (processor). This is particularly the case for media communication, signal processing and other similar applications. For example, in a cellular base station, several streams of voice and data communication will be supported each requiring its own resource. To analyze whether such an architecture is capable of executing single-stream applications without losing much on performance, a set of single-stream applications is considered and executed on the setup. The goal is to fully utilize the system by sharing the given work with idle processors through some
existing parallelization techniques. During analysis of the results, single-stream applications are categorized as scalable and non-scalable applications for ease of presentation. The discussion begins with scalable applications and then moves on to applications that do not scale on parallelization, which can be identified by the proposed ERE. Even in the case of multi-stream applications, there is no systematic way to judge performance-energy tradeoffs. The discussions end with techniques to judge such tradeoffs.

The benchmark applications were parallelized using the parallel techniques defined in Chapter 3. The performance and energy improvements got from these techniques are then judged using the ERE framework. The system is analyzed simultaneously using a traditional evaluation framework with power, execution time, energy and energy-delay product \cite{HIG94} as the metrics, to highlight the differences and advantages of the ERE framework. The aim is to keep the discussion simple and hence, only the most relevant metric is presented in this section on experimental results. While most of the results are applicable even for other metrics, the comparison of \textit{ERE} with other equivalent metrics is deferred till Section 4.4.

**4.3.2. Scalable Applications**

This section presents the results for the set of applications whose performance scale very well on parallelization. In the next section, results are presented for the applications that are hard to parallelize. This section covers art, bzip2 from SPEC CFP2000 benchmark suite \cite{Sta01} and jpeg from MediaBench \cite{LPMS97} suite. Here, the results for traditional metric is not presented since the trends are similar to Chapter 3. Analysis using
ERE is directly presented, and where ever applicable, the differences of the traditional metrics are highlighted.
The ERE framework for the scalable applications is shown in Figure 4.2. In the legend, 1 DSP, 2 DSPs, 4 DSPs, 8 DSPs, and 12 DSPs shows the case when 1, 2, 4, 8, and 12 DSPs are involved in parallel execution of the code. To see how DSPs perform with respect to PPC, the same code (with same input) was executed on just one PPC, which is shows as 1 PPC in the legends. All the conclusions for performance and energy derived from the traditional framework can be derived using speedup and energy-savings graphs of the ERE framework. The definition of ERE and the intuitive interpretations from Table 4.1 guides one to highlight the trends and further explore the results.

A valuable addendum comes from efficiency of resource usage ($\eta$) of the ERE framework. Looking at $\eta$ of Figure 4.2, bzip2 is the most resource-aware algorithm. 1 PPC is the best utilized resource for bzip2, which indicates that the application is very well hosted on PPC. In this case, $\eta > 1$ arises due to subtle differences in hardware (PPC has caches while DSPs don’t). The best $\eta$ for jpeg occurs at 1 DSP. The consistent drop in $\eta$ beyond 1 DSP indicates that the speedups achieved on parallelization to more DSPs are not worth the amount of resources spent.

The “best” points for $S_{nj}$, $\eta$, $\Delta$, and $ERE$ plots are different. $ERE$ accounts for the performance-energy tradeoffs. The “best” performance-energy point of bzip2 ($ERE \Rightarrow 4$ DSPs) is different from those derived through pure performance and energy considerations ($S_{nj}$ and $\Delta \Rightarrow 12$ DSPs). This is attributed to the following fact. The rate at which energy is saved is 40% up to 4 DSPs. But, the improvement rate rapidly drops to mere 8% from 4 DSP to 8 DSPs, and further to 0% (due to saturation). Efficiency is
consistent (∼100%) up to 4 DSPs, but drops beyond that. These two factors determine the best ERE point at 4 DSPs.

Another characteristic example is jpeg, where the tradeoff considered by ERE is evident. ERE shows the best value at 2 DSPs in spite of a 35% drop in η for that case. By giving up a little on the efficiency, one can achieve improvements of 18% in the energy and 28% in speedup than the most resource-efficient configuration. Hence, it makes more sense to go with 2 DSPs, which is shown by ERE in one straight-forward step. 

Such trends and useful information derived from ERE graphs are clearly not observable when traditional evaluation frameworks are used.

For all configurations of art, bzip2 and jpeg, the values $S_{Nj} > 1$, $Δ > 0$, $ERE > 0$. This means that all resources are able to host the algorithms very well despite having varying configurations. In short, these algorithms are friendly to optimizations and mapping using parallel techniques. Thus, the presented schemes are validated to be effective by ERE framework.

4.3.3. Non-Scalable Applications

The previous section covered scalable applications. Now, the scenario changes when non-scalable applications are considered. Serial and parallel versions of pegwit from MediaBench [LPMS97] were executed on the heterogeneous platform. The algorithm used for parallelization is the same as the one presented in the previous section. Figure 4.3 shows the ERE framework.
The ERE framework of Figure 4.3 demonstrates that pegwit is not a scalable algorithm. $S_{Nj}$ hardly increases beyond 2 DSPs (6% improvement over 1 DSP). $\Delta$ increases till 2 DSPs and takes a negative trend beyond that point, which implies that the energy consumption increases on more parallelization. The energy-savings of 1 PPC is comparable to that of 2 DSPs and hence, 1 PPC is the other alternative.

The best performance-energy point with $\text{ERE}=0.02$ (Figure 4.3) is shared by 2 DSPs and 1 PPC. There is just a 2% overall improvement from the 1 DSP case. By merely
looking at the benefits, it is clear that the parallel techniques prove to be futile in improving energy or performance. Since a serial version is comparable in performance, one can choose the 1 DSP configuration too. That is, a serial version running either directly on 1 PPC or on 1 DSP is the best alternative for good performance and energy.

4.3.4. Multi-Stream Applications

This section covers the G.721 voice compression standard from MediaBench suite [LPMS97] and the Complex FIR filter [Mot01a].

Intuitively, multi-stream applications perform well if shifted from one resource to multiple resources due to the speedups achieved by parallel handling of streams. The speedups scale only up to the point where each stream is independently handled by one resource. In this case, 12 DSPs with $S_{Nj} = 11.4$ is the best point for such a scenario [Figure 4.4(i)].

Traditional analysis methodologies are easily extendible to the ERE framework. If speedup and efficiency are important, one can combine $\eta$ and $S_{Nj}$. In the case of FIR, if 4% of efficiency ($\eta$) is given up to move from the best point (4 DSPs) to the next best point (12 DSPs), one can achieve a significant increase in the speedup (4 to 11.47). Thus, deriving a tradeoff from these graphs results in gaining a lot in speedup. Another point to be noted is the systematic methodology behind choosing a tradeoff. For instance, 8 DSPs is not a good point for tradeoff when compared to 12 DSPs. In this case, even though the speedup increases by 6, $\eta$ deteriorates by 26%. Thus 12 DSPs make a better choice.
Figure 4.4. ERE framework: Speedup ($S_{N,j}$), efficiency ($\eta$), energy-savings ($\Delta$) and ERE for multi-stream applications.

for tradeoff. Such beneficial tradeoffs are not apparent from the equivalent traditional framework.
Although both $S_{N_j}$ and $\Delta$ suggest same best points (12 DSPs), it should be noted that their intermediate trends are different. $\eta$ influences the performance-energy tradeoffs and drives $ERE$ to make 12 DSPs as the best performance-energy point.

### 4.4. Extension of ERE Framework

ERE framework analysis extends to any architecture irrespective of the type. The ERE equation can be broadly defined as follows:

\begin{align*}
ERE &= \Delta \cdot \eta \quad (4.5) \\
\Delta &= \frac{E_{base} - E_{new}}{E_{base}} \\
\eta &= \frac{S_{new}}{\alpha_{new} \cdot \frac{N_{new}}{N_{base}}} \quad \text{(4.5)} \\
S_{new} &= \frac{T_{new}}{T_{base}}
\end{align*}

where, $base$ is the base configuration and $new$ is the configuration that needs to be evaluated.

In the case of single resource architectures (homogeneous architectures), $new$ and $base$ are variations in configurations of the same resource (for instance, performance-energy study for various cache sizes). In this case, $\alpha_{new} = 1, N_{new} = N_{base} = 1$ and $\eta$ compares execution times. ERE can also be used to study a parallel setup. The base case need not be 1 processor case. Say, before buying hardware for a parallel setup, $ERE$ can be used to study the improvements one can get by involving 500, or 1000 (values for
new) processors assuming the currently available 200 processors ($N_{\text{base}}$) of the same class as base. Thus, even a complicated parallel setup can be analyzed for performance and energy. The base case can also be a mix of processors, in which case, $\alpha_{\text{new}}$ is the skew of the new configuration with respect to the base configuration and, $N_{\text{base}}$ and $N_{\text{new}}$ are the total number of resources (of all classes) used in the base and new configurations respectively. Alternatively, one can feed in the desired improvements needed (into the ERE framework equations) and settle for cost-effective, high-performance or energy-aware configuration. Hence, the ERE framework is generic (as shown in Figure 4.1) and can be flexed to one’s own discretion.

ERE framework can be used by system architects, designers, analysts, users, and researchers based on their requirements. The ERE formulas and methodology can be easily plugged into any existing evaluation frameworks (synthesis tools, simulators) that study benchmark applications. It can also be extended to any existing setup by profiling a set of applications based on their nature (similar to the multi-stream, parallel, serial application analysis). The trends from such profiled data can be used to decide between configurations or to directly implement a configuration or an application.

4.5. Summary

Energy-Resource Efficiency (ERE) and its framework provide an elegant way to capture performance-energy tradeoffs in a multi-resource heterogeneous system. The experimental results clearly emphasized the need to consider energy consumption besides
performance while assessing a system. ERE considers the speedup and energy variations to model performance-energy tradeoffs in a system. Performance-energy tradeoffs are prominent when the ERE framework is used during evaluation. This framework can also be used to establish a system to a set performance-energy configuration. The ERE framework considers the underlying environment and includes resource usage also as a factor while studying the system. This framework and the modeling techniques can be easily extended to any type of system. Such features are not provided by any existing metric.

This work also demonstrates that by utilizing ERE along with application-aware scheduling and resource allocation strategies, one can achieve up to 80% performance and energy gains, since the tradeoffs are very prominent. Currently, the diagnosed results are utilized to manually partition the code by taking into consideration the desired performance-energy tradeoff. Compilers and operating systems can instead use these results in a feedback mechanism during scheduling, partitioning and resource allocation, to do a better management of the system. These techniques should adapt themselves to consider not just the performance, but the performance-energy tradeoffs, while making scheduling decisions. A framework having speedup, efficiency, energy-savings and ERE is recommended instead of any traditional framework to study and establish these tradeoffs. ERE could be used in tandem with other metrics to fine tune a system.
CHAPTER 5

Energy Management Techniques for Memory Resident Databases

The preceding chapters focused on systems that handle streaming data. This chapter discusses solutions to some of the challenges in modern databases. With the tremendous growth of system memories, memory-resident databases are increasingly becoming important in various domains. Some of these databases serve to enhance the data management capabilities, while others act as the core storage component. Newer memories provide a structured way of storing data in multiple chips, with each chip having a bank of memory modules. A banked storage system offers a lot of room for performance and energy optimizations. The structured organization enables the selective mode of operation in which banks are exclusively accessed. Energy consumption could be reduced if unused banks are put to a lower power mode using access pattern information. This chapter presents the implications of a banked memory environment in supporting memory-resident databases.

5.1. Memory Resident Databases

Memory-resident databases (also called in-memory databases [Blo99]) are emerging to be more significant due to the current era of memory-intensive computing. These
databases are used in a wide range of systems ranging from real-time trading applications to IP routing. With the growing complexities of embedded systems (like real-time constraints), use of a commercially developed structured memory database is becoming very critical [Bir03]. Consequently, device developers are turning to commercial databases, but existing embedded DBMS software has not provided the ideal fit. Embedded databases emerged well over a decade ago to support business systems, with features including complex caching logic and abnormal termination recovery. But on a device, within a set-top box or next-generation fax machine, for example, these abilities are often unnecessary and cause the application to exceed available memory and CPU resources. In addition, current in-memory database support does not consider embedded system specific issues such as energy consumption. Designers often use the term "power" almost interchangeably with "battery life". However, in embedded devices, one needs to focus on energy consumption rather than power, since there is a limited supply of energy in a battery, even though the power a battery is required to supply can vary over a substantial range during the course of its life. The focus of this study is on the energy consumption.

Memory technology has grown tremendously over the years, providing larger data storage space at a cheaper cost. Recent memory designs have more structured and partitioned layouts in the form of multiple chips, each having memory banks [Ram03]. Banked memories are energy efficient by design, as per-access energy consumption decreases with decreasing memory size (and a memory bank is typically much smaller compared to a large monolithic memory). In addition, these memory systems provide
low-power operating modes, which can be used for reducing the energy consumption of a bank when it is not being used. An important question regarding the use of these low-power modes is when to transition to one once an idleness is detected. Another important question is whether the application can be modified to take better advantage of these low-power modes. While these questions are slowly being addressed in architecture, compiler, and OS communities, there has been no prior work that examines the energy and performance behavior of databases under a banked memory architecture. Considering increasingly widespread use of banked memories, such a study can provide valuable information regarding the behavior of databases under these memories and potential modifications to DBMSs for energy efficiency. Since such banked systems are also being employed in high-end server systems, banked memory friendly database strategies can also be useful in high-end environments to help reduce energy consumption.

Detailed energy characterization of a banked memory architecture that runs a memory-resident DBMS showed that nearly 59% of overall energy (excluding input/output devices) in a typical query execution is spent in the main memory, making this component an important target for optimization (see Figure 5.1). Moreover, for any system, memory power and energy consumption have become critical design parameters besides cost and performance. Based on these observations, this chapter evaluates the potential energy benefits that memory-resident database queries can achieve by making use of banked memory architectures supported with low-power operating modes. Since each memory
bank is capable of operating independently, this opens up abundant avenues for energy and performance optimizations.

This chapter focuses on a banked memory architecture and presents potential energy benefits when database queries are executed on such architectures. To see whether query execution can make use of available low-power modes, both hardware and software techniques are studied. The hardware techniques predict the idleness of memory banks and switch the inactive (idle) banks (during query execution) to low-power operating modes. This chapter also presents a query-based memory energy optimization strategy, wherein the query plan is augmented by explicit bank turn-off/on instructions that transition memory banks into appropriate operating modes during the course of execution based on the query access pattern. All the proposed schemes are experimentally evaluated using an energy simulator. Experiments using TPC-H queries \[\text{Tra04}\] and a set of queries suitable for handheld devices clearly indicate that both hardware-based and query-directed strategies save significant memory energy.

Apart from providing useful input for database designers, the derived results can also be used by hardware designers to tune the behavior of low-power modes so that they handle query access patterns better. Similar to the observation that creating a lightweight version of a disk-based database will not serve as a suitable in-memory database, the belief is that taking an in-memory database system and using it on a banked architecture without any modification may not generate the desired results. Therefore, the results presented in this work also shed light on how database design and memory architecture design interact with each other.
Figure 5.1. Breakup of the energy consumption for various system components. The results are based on the average energy consumption of TPC-H benchmarks [Tra04] executed on a memory-resident DBMS.

The remainder of this chapter is organized as follows. Section 5.2 elaborates the memory database setup and also the memory banking scheme used for this work. Section 5.3 presents in detail the proposed hardware and query-directed energy optimization techniques. The results of energy evaluation of these schemes are discussed in Section 5.4. The experiments also account for the performance overhead incurred in supporting the new schemes. Finally, the final section summarizes the results.

5.2. System Architecture

5.2.1. DBMS

For this work, the PostgreSQL DMBS was modified to model a main-memory resident database system. The block diagram for the experimental setup is shown in Figure 5.2. The core components are derived from PostgreSQL. The flow of this model is similar
to PostgreSQL except that the database is memory resident. A query is parsed for syntax and then sent to the rewrite system. The rewrite system uses the system catalog to generate the query tree, which is then sent to the optimizer. The query optimizer derives the cost of the query in multiple ways using the query tree and issues the best suited plan to the query execution engine. The proposed software-based techniques were incorporated at the optimizer stage of the DBMS. These optimizations are based on the cost that is derived for each of the query plans (the discussion pertaining to the modified cost model is deferred till Section 5.3). Based on the final query execution plan, the execution engine executes the query by using the database. The database is entirely memory resident and the memory is organized in a banked format (elaborated in the following section). The executor recursively iterates the query plan and uses a per-tuple based strategy (pipelined execution, and not bulk processing) to project the output results. The proposed hardware optimizations are at the computer architecture level of the system. Since the base DBMS model is similar to PostgreSQL, each component is not elaborated in detail ([Post02] provides an elaborate discussion). Instead, just the specific contributions and modifications to DBMS (shown in blue in Figure 5.2) are discussed.
in the following sections. Overall, the proposed strategies require modification to the
query optimizer, memory hardware, and system software components.

5.2.2. Memory Model

This work focuses on a memory system that contains a memory array organized as banks
(rows) and modules (columns), as is shown pictorially in Figure [5.3] for a $4 \times 4$ memory
module array. Such banked systems are already being used in high-end server systems
[Ram03] as well as low-end embedded systems [Sam]. The proposed optimizations will,
however, apply to most bank-organized memory systems.

Accessing a word of data would require activating the corresponding modules of the
shown architecture. Such an organization allows one to put the unused banks into a low-
power operating mode. To keep the issue tractable, this chapter bases the experimental
results on a sequential database environment and does not consider a multiprocessing
environment (like transaction processing which requires highly complex properties to
be satisfied). The experiments assume that there is just one module in a bank; hence, in
the rest of the discussion, the terms “bank” and “module” are used interchangeably.

5.2.3. Operating Modes

There exists five operating modes for each memory module: active, standby, nap, power-
down, and disabled\(^1\). Each mode is characterized by its energy consumption and the

\(^1\)Current DRAMs [Ram03] support up to six energy modes of operation with a few of them supporting
only two modes. One may choose to vary the number of modes based on the target memory.
time that it takes to transition back to the active mode (termed *resynchronization time* or *resynchronization cost*). Typically, the lower the energy consumption, the higher the resynchronization time \[\text{[Ram03]}\]. These modes are characterized by varying degrees of the module components being active. The details of the power modes are discussed below:

- **Active**: In this mode, the module is always ready to perform a read or write operation. As the memory unit is ready to service any read or write request, the resynchronization time for this mode is the least (zero units), and the energy consumption is the highest.

- **Standby**: In this mode, a few DRAM components are disabled resulting in significant reduction in energy consumption compared to the active mode. The resynchronization time for this mode is typically one or two memory cycles. Some state-of-the-art RDRAM memories already exploit this mode by automatically transitioning into the standby mode at the end of a memory transaction \[\text{[Ram03]}\].
• **Nap:** This mode can typically consume two orders of magnitude less energy than the active mode, with the resynchronization time being higher by an order of magnitude than the standby mode.

• **Power-Down:** This mode provides another order of magnitude saving in energy. However, the resynchronization time is also significantly higher (typically thousands of cycles).

• **Disabled:** If the content of a module is no longer needed, it is possible to completely disable it (saving even refresh energy). There is no energy consumption in this mode, but the data is lost.

When a module in standby, nap, or power-down mode is requested to perform a memory transaction, it first goes to the active mode, and then performs the requested transaction. Figure 5.4 shows possible transitions between these modes (the dynamic energy \(^2\) consumed in a cycle is given for each node). The resynchronization times in cycles (based on a cycle time of 3.3ns) are shown along the arrows (assuming a negligible cost \(\epsilon\) for transitioning to a lower power mode). Note that this model is flexible enough to take in different values for energy consumption and resynchronization costs, and the default values used in the experiments are the ones given in Figure 5.4.

While one could employ all possible transitions given in this figure (and maybe more), the proposed query-directed approach only utilizes the transitions shown by solid arrows. The runtime (hardware-based) approaches, on the other hand, can exploit two

---

\(^2\)This work concentrates exclusively on the dynamic power consumption that arises due to bit switching, and does not consider the static (leakage) power consumption [RCN02].
additional transitions: from standby to nap, and from nap to power-down. The energy values shown in this figure have been obtained from the measured current values associated with memory modules documented in memory data sheets (for a 2.5V, 3.3ns cycle time, 8MB modules) \([\text{Ram03}]\). The resynchronization times have also been obtained from the same data sheets.

### 5.2.4. System Support for Power Mode Setting

Typically, several of the memory modules (that are shown in Figure 5.3) are controlled by a memory controller which interfaces with the memory bus. The interface is not only for latching the data and addresses, but is also used to control the configuration and operation of the individual modules as well as their operating modes. For example, the operating mode setting could be done by programming a specific control register in
each memory module (as in RDRAM [Ram03]). Next is the issue of how the memory controller can be told to transition the operating modes of the individual modules. This is explored in two ways in this work: hardware-directed approach and software-directed (query-directed) approach.

In the hardware-directed approach, there is a Self-Monitoring and Prediction Hardware block (as shown in Figure 5.3), which monitors all ongoing memory transactions. It contains some prediction hardware to estimate the time until the next access to a memory bank and circuitry to ask the memory controller to initiate mode transitions. The specific hardware depends on the prediction mechanism that is employed, and will be discussed later in the chapter.

In the query-directed approach, the DBMS explicitly requests the memory controller to issue the control signals for a specific module’s mode transitions. There is a set of configuration registers in the memory controller (see Figure 5.3) that are mapped into the address space of the CPU (similar to the registers in the memory controller in [Int98]). Programming these registers using one or more CPU instructions (stores) would result in the desired power mode setting. This brings up the issue of which CPU activity needs to issue such instructions. The memory control registers could potentially be mapped into the user address space directly, making it possible for the user application/DBMS to directly initiate the transitions. However, there are a couple of drawbacks with this approach. The first one is that powering down modules that are shared with

---

3Limited amount of such self-monitored power-down is already present in current memory controllers (e.g., Intel 82443BX [Int98] and Intel 820 Chip Set [Int]).
other applications brings up the data protection issue. The other problem could be that a given program does not have much knowledge of the memory activity of other programs, and will thus not be able to accommodate more global optimizations. With two or more applications sharing a memory module, the operating system may be a better judge of determining the operating (power) modes. So, the other option is to make the issuance of these instructions a privilege of the operating system, with the DBMS avail-ing of this service via a system call. However, since the focus of this work is to explore the potential benefits of memory module energy optimizations from the perspective of queries, it focus on a single program environment, and assumes that the registers are directly mapped into user space (thus, they can be controlled by the DBMS).

Regardless of which strategy is used, the main objective of employing such strategies is to reduce the energy consumption of a query when some memory banks are idle during the query’s execution. That is, a typical query only accesses a small set of tables, which corresponds to a small number of banks. The remaining memory banks can be placed into a low-power operating mode to save memory energy. However, it is also important to select the low-power mode to use carefully (when a bank idleness is detected), as switching to a wrong mode either incurs significant performance penalties (due to large resynchronization costs) or prevents one from obtaining maximum potential energy benefits.

Note that energy optimization can be performed from two angles. Suitable use of low-power operating modes can reduce energy consumption of a given query execution.
Also, the query plan can be changed (if it is possible to do so) to further increase energy benefits. This work basically explores both these aspects.

5.3. Power Management Schemes

In such a banked architecture, the memory can be managed through either of the following two approaches: (1) a runtime approach wherein the hardware is in full control of operating mode transitions; and (2) a query-directed scheme wherein explicit bank turn-on/off instructions are inserted in the query execution plan to invoke mode transitions. One also has the option of using both the approaches simultaneously (illustrated in later sections).

5.3.1. Hardware-Directed Schemes

Two hardware-directed approaches that allow the memory system to automatically transition the idle modules to an energy conserving state are first explored. The problem then is to detect/predict bank idleness and transition idle banks into appropriate low-power modes.

5.3.1.1. Static Standby Scheme. The first approach is a per-access optimization. Most of the recent DRAMs allow the chips to be put to standby mode immediately after each reference [Ram03]. After a read/write access, the memory module that gets accessed can be placed into the standby mode in the following cycle. This scheme is referred to as the static standby mode in the rest of this discussion. Note that, while this scheme is
not very difficult to implement, it may lead to frequent resynchronizations, which can be very harmful as far as execution cycles are concerned.

5.3.1.2. Dynamic Threshold Scheme. The second hardware-guided approach is based on runtime dynamics of the memory subsystem. The rationale behind this approach is that if a memory module has not been accessed in a while, then it is not likely to be needed in the near future (that is, inter-access times are predicted to be long). A threshold is used to determine the idleness of a module after which it is transitioned to a low-power mode. More specifically, this scheme puts each memory module into a low-power state using its idle cycles as the threshold for transition.

The schematic of the dynamic threshold scheme is depicted in Figure 5.5. After $idle_{standby}$ cycles of idleness, the corresponding module is put in the standby mode. Subsequently, if the module is not referenced for another $idle_{nap}$ cycles, it is transitioned to the nap mode. Finally, if the module is not referenced for a further $idle_{down}$ cycles, it...
is placed into the power-down mode. Whenever the module is referenced, it is brought back into the active mode incurring the corresponding resynchronization costs (based on what low-power mode it was in). It should be noted that even if a single bank experiences a resynchronization cost, the other banks will also incur the corresponding delay (to ensure correct execution). Implementing the dynamic mechanism requires a set of counters (one for each bank) that are decremented at each cycle, and set to a threshold value whenever they expire or the module is accessed. A zero detector for a counter initiates the memory controller to transmit the instructions for mode transition to the memory modules. Another alternative to this dynamic scheme would be one based on the adaptive threshold wherein each module adaptively learns the threshold for transition. However, the hardware implementation costs of such a scheme would be extremely high. Consequently, such complex implementations are not considered in this work.

5.3.2. Software-Directed Scheme

It is to be noted that a hardware-directed scheme works well independent of the DBMS and the query optimizer used. This is because the idleness predictors are attached to the memory banks and monitor idleness from the perspective of banks. In contrast, a query-directed scheme gives the task of enforcing mode transitions to the query. This is possible because the query optimizer, once it generates the execution plan, has a complete information about the query access patterns (i.e., which tables will be accessed and in what order, etc). Consequently, if the optimizer also knows the table-to-bank mappings, it can have a very good idea about the bank access patterns. Then, using this
information, it can proactively transition memory banks to different modes. The rest of this section further details each step in the query-directed approach, which includes customized bank allocation, query analysis, and insertion of bank turn-on/off instructions.

5.3.2.1. Bank Allocation. In the case of software-directed scheme, the table allocation is handled by the DBMS. Specifically, the DBMS allocates the newly-created tables to the banks, and keeps track of the table-to-bank mappings. When a “create table” operation is issued, the DBMS first checks for free space. If there is sufficient free space available in a single bank, the table is allocated from that bank. If a bank is not able to accommodate the entire table, the table is split across multiple banks. Also, while creating a new table, the DBMS tries to reuse the already occupied banks to the highest extent possible; that is, it does not activate a new bank unless it is necessary. However, it also tries not to split tables excessively. In more detail, when it considers an already occupied bank for a new table allocation, the table boundaries are checked first using the available space in that bank. If a bank is more than two-thirds full with the table data, the rest of the bank is padded with empty bits and the new table is created using pages from a new bank. Otherwise, the table is created beginning in the same bank. Irrespective of whether the table is created on a new bank or not, the DBMS creates a new table-to-bank mapping entry after each table creation.

\[4\] Note that the unactivated (unused) banks – i.e., the banks that do not hold any data – can remain in the disabled mode throughout the execution.
Such complexities involved in bank allocation do not exist in hardware-directed schemes since there is absolutely no software control. Consequently, the hardware-directed schemes use *sequential first touch placement policy*. This policy allocates new pages sequentially in a single bank until it gets completely filled, before moving on to the next bank. Also, the table-to-bank mapping is not stored within the DBMS since the mode control mechanism is handled by the hardware.

**5.3.2.2. Estimating Idleness and Selecting the Appropriate Low-Power Mode.** It should be emphasized that the main objective of the query-directed scheme is to identify bank idleness. As explained above, in order to achieve this, it needs table-to-bank mapping. However, this is not sufficient as it also needs to know when each table will be accessed and how long an access will take (i.e., the query access pattern). To estimate this, one needs to estimate the duration of accesses to each table, which means estimating the time taken by the database operations. Fortunately, the current DBMSs already maintain such estimates for query optimization purposes \[\text{RG02, SKS01, Sle03, Dat01, GLSW93}\]. More specifically, given a query, the optimizer looks at the query access pattern using the generated query plan. The inter-access times are calculated using the query plan. A query plan elucidates the operations within a query and also the order in which these operations access the various tables in the database. Even in current databases, the query plan generator estimates access costs using query plans \[\text{Dat01}\]. The same access cost estimation methodology is used. These access costs are measured in terms of page (block) fetches. In the memory-resident database case, a page is basically the block that is brought from memory to the cache. For instance, the cost of sequential scan is defined
as follows (taken from [Dat01]):

\[
\text{Cost}_{\text{seq,scan}} = N_{\text{blocks}} + CPU \times N_{\text{tuples}}
\]

Here, \(N_{\text{blocks}}\) is the number of data blocks retrieved, \(N_{\text{tuples}}\) is the number of output tuples, and \(CPU\) is the fudge factor that adjusts the system tuple-read speed with the actual memory hierarchy data-retrieval speed. Usually, optimizers use the above cost metric to choose between multiple query plan options before issuing a query. Thus, when a cost is attached to each page (block) read/write operation, an estimate of the access time is obtained as follows:

\[
\text{Cost}_{\text{block,fetch}} = T \text{ cycles}
\]

\[
\text{Cost}_{\text{seq,scan}} = N_{\text{blocks}} \times T + CPU \times N_{\text{tuples}} \times \frac{\text{block}}{\text{tuples}} \times T
\]

In these expressions, \(T\) is the delay in cycles to fetch a block from the memory. Thus, this new cost plan is projected in terms of access cycles. The methodology is extended to other database operations like JOIN and AGGREGATE based on the cost models defined in [Fon, Dat01].

Given a query, each operation within the plan (including sub-plans) is first identified and then the access cost (in cycles) is estimated for each primitive operation. The objective in estimating the per-operation time in cycles is to eventually identify the inter-access times of operations in the query (and hence, to put the banks that hold unused tables to low-power modes). There are table accesses associated with each operation,
and bank inter-access times depend on the table inter-access times. A query has information of the tables that it accesses. Thus, knowing the inter-access time for each operation leads to the inter-access times for each table as well. A table is mapped to certain banks, and the table-to-bank mapping is available in the query optimizer.

Consequently, if the table inter-access time is $T$, and the resynchronization time is $T_p$ (assuming less than $T$), then the optimizer can transition the associated modules into a low-power mode (with a unit time energy of $E_p$) for the initial $T - T_p$ period (which would consume a total $[T - T_p]E_p$ energy), activate the module to bring it back to the active mode at the end of this period following which the module will resynchronize before it is accessed again (consuming $T_pE_a$ energy during the transition assuming that $E_a$ is the unit time energy for active mode as well as during the transition period). As a result, the total energy consumption with this transitioning is $[T - T_p]E_p + T_pE_a$ without any resynchronization overheads, while the consumption would have been $TE_a$ if there had been no transitioning (note that this calculation considers only the idle period).

The DBMS optimizer evaluates all possible choices (low-power modes) based on corresponding per cycle energy costs and resynchronization times, and table inter-access time to pick up the best choice. Note that the DBMS can select different low power modes for different idle periods of the same module depending on the duration of each idle period. Specifically, the most energy saving low-power mode is used, without increasing the original query execution time (i.e., when the original idleness is over, the bank should be up in the active mode).
5.3.2.3. **Inserting Bank-On/Off Instructions.** The last part of the software-directed scheme is to insert explicit (operating) mode transitioning instructions in the query execution plan. For this, markers (place holders) are introduced, which are then interpreted at the low-level (interpreted later by the memory controller, which actually sets the corresponding low-power modes). This is done so that the query execution engine can issue the query without much performance overhead, and with the same transparency.

As an example, consider the following. Let tables A and B each have 1000 records, each record being 64 bytes. Consider the query plan depicted in Figure 5.6(i), taken from PostgreSQL. The per step access costs are also shown. From the generated query plan, it is evident that table A is not accessed between point P1 and point P2. Once the results are extracted after the scan at point P1, the banks that hold table A can be put to a low-power mode, and the banks that hold table B can be activated for data extraction. This is illustrated in Figure 5.6(ii) using place-markers for tables A and B. Banks holding Table A are reactivated at point P2 (banks of Table B remain off).

5.4. **Experimental Evaluation of Memory Management Schemes**

This section studies the potential energy benefits of the proposed hardware and software-directed schemes. Firstly, the evaluation setup is discussed in detail. Then, the set of queries used to study these schemes is presented. The energy consumption results are presented following that. While the energy benefits of using the new schemes are presented, the overheads associated with supporting each of the schemes are also elaborated.
5.4.1. Setup

5.4.1.1. Simulation Environment. As mentioned before, the query-directed schemes are implemented in the query optimizer of the memory database model elaborated in Section 5.2.1. The DBMS is interfaced to an enhanced version of the SimpleScalar/Arm simulator [Aus] to form a complete database system. The intermediate interface (invoked by DBMS) provides a set of operating system calls (on Linux kernel 2.4.25), which in turn invokes the SimpleScalar simulator. The SimpleScalar simulator models a modern microprocessor with a five-stage pipeline: fetch, decode, issue, write-back, and commit. The hardware techniques are implemented within the framework of the simoutorder tool from the SimpleScalar suite, extended with the ARM-ISA support [Aus]. Specifically, a processor architecture similar to that of Intel StrongARM SA-1100 is
modeled. The modeled architecture has a 16KB direct-mapped instruction cache and a 8KB direct-mapped data cache (each of 32 byte-length). A 256-entry full associative TLB with a 30-cycle miss latency also exists in the system. The off-chip bus is 32 bit-wide. For estimating the power consumption (and hence, the energy consumption), the Watch simulator from Princeton University [BTM00] is used.

The experimental banked memory model is based on [DKV+01, LFZE00]. The values from Figure 5.4 is used for modeling the delay (transition cycles) in activation and resynchronization of various power-states. It should be emphasized that the simulations performed also account for all performance and energy overheads incurred by the new schemes. In particular, the energy numbers presented here include the energy spent in maintaining the idleness predictors (in the hardware-directed scheme) and the energy spent in maintaining the table-to-bank mappings (in the query-directed scheme), and in fetching and executing the bank turn-on/off instructions. Two important statistics are presented in the experimental results. *Energy consumption* corresponds to the energy consumed in the memory system (including all overheads in other system components). Also presented are the statistics about the *performance overhead* (i.e., *increase in execution cycles*) for each of the new schemes. This overhead includes the cycles spent in resynchronization as well as the cycles spent (in the CPU datapath) in executing the turn-on/off instructions (in the query-directed scheme).

5.4.1.2. Queries. To evaluate the scheme on memory-resident databases, two classes of queries are considered. The first class is a subset of queries from the Transaction Processing Council (TPC-H) benchmark [Tra04]. TPC-H involves complex queries with
Table 5.1. The two classes of queries considered for the experiments

<table>
<thead>
<tr>
<th>Source</th>
<th>Query</th>
<th>Description</th>
<th>Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-H</td>
<td>Q6</td>
<td>Simple query</td>
<td>PART, CUSTOMER, ORDERS, LINEITEM generated using dbgen with scale 1.0</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>Complex query involving JOIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>Complex query involving NEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q17</td>
<td>Complex query involving JOIN and NEST</td>
<td></td>
</tr>
<tr>
<td>Queries targeting a simple organizer</td>
<td>P1</td>
<td>Simple name and address lookup</td>
<td>ADDRESSBOOK populated with 1.3 million entries, 50% subset of FRIENDS and 25% subset of COLLEAGUES</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>Lookup in directory of friends</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>Lookup in directory of colleagues and friends</td>
<td></td>
</tr>
</tbody>
</table>

a large amount of data accesses. Operations in decision support benchmarks (TPC-D evolved to TPC-H) have good spatial locality with abundant data intensive operations [CTLP+99a]. This assists one to perform a rigorous test of the schemes. The top part of Table 5.1 gives details of the TPC-H queries used and the corresponding database parameters. The selected operations represent a good mix and could be used to build a variety of complicated queries.

Memory-resident databases run queries that are different from the typical database queries as seen in TPC-H. The second set of queries considered are representative of applications that execute on handheld devices. The typical operations that are performed
on an organizer were imitated on the setup (the queries named as P1, P2, P3). The first query involves a simple address lookup using a ‘NAME’ as input. The SQL for query P1 is shown on the left section of Table 5.2. Recent organizers \cite{Han03, Pal03} provide an ordered view of the underlying addressbook database. For instance, organizers provide the creation of folders. A “friends” folder can be a collection of personnel with a tag set as “friend” in the addressbook. Folder is defined as a restrained/customized view of the same database (address book). Intuitively, query P2 strives to do a lookup of friends living in a particular city. A person interested in visiting a city can run this query before he/she leaves for that place. The “friends” view and hence the query P2 is defined on the right section of Table 5.2. Query P3 combines views (folders). For this, a new folder called “colleagues” is defined. P3 aims to find friends and/or colleagues whose names start with an ‘a’, living in a particular ‘CITY’. Since P3 is very similar to P2 with some extra fields, the SQL for P3 is not presented. The intermediate tables and results during query execution are also stored in the memory.

5.4.1.3. Default Parameters. For the experiments, the database tables were populated using the qgen software from TPC-H benchmark suite with a scale of 1.0. The organizer database is populated with 1.3 million records.

For dynamic threshold scheme, 10, 100 and 10,000 cycles are used for $idle_{standby}$, $idle_{nap}$, and $idle_{down}$, respectively. For all schemes, the banks are in power-down mode before their first access. On/Off instructions are inserted based on the inter-access times of table. The same cycles as in $idle_{standby}$, $idle_{nap}$, and $idle_{down}$ are used for inserting instructions. As an example, consider the inter-access (T) of a table as 25 cycles, which
Table 5.2. SQL for organizer queries

<table>
<thead>
<tr>
<th>Query P1</th>
<th>Query P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT a_name, a_address, a_city, a_office_phone, a_home_phone, a_mobile_phone, a_email, a_web, a_specialnotes FROM addressbook WHERE a_name = '[NAME]';</td>
<td>CREATE VIEW friends AS P2:</td>
</tr>
<tr>
<td>SELECT a_address, a_city, a_office_phone, a_home_phone, a_mobile_phone FROM friends WHERE a_city = '[CITY]'</td>
<td></td>
</tr>
<tr>
<td>SELECT a_address, a_city, a_mobile_phone FROM friends WHERE a_tag = '[FRIEND]' GROUP BY a_name;</td>
<td></td>
</tr>
<tr>
<td>GET GROUP BY a_name;</td>
<td></td>
</tr>
</tbody>
</table>

lies between $10 \text{(idle}_{\text{standby}}$) and $100 \text{(idle}_{\text{nap}}$ cycles. An On/Off instruction is inserted at the beginning of $T$ to put a table to standby mode for 24 cycles, taking into consideration the resynchronization period of 1 cycle as well. Similar technique is applied for inter-access times that fall in between other power modes.

A single page transfer time is needed for access cost calculation in software-directed scheme. This is derived by executing the TPC-H queries on the SimpleScalar simulator (with the SA-1100 model) and by studying the cycle times for transferring a data block.
from memory to the cache. For all experiments, the default configuration is the 512MB RDRAM memory with 8MB banks. In the following section, the energy implications of the hardware and software schemes are studied using this setup. Then the performance overhead studies and sensitivity of the schemes to various system parameters are presented.

5.4.2. Query Energy Evaluation

Figure 5.7 shows the normalized memory energy consumption for the hardware-directed schemes. While presenting the results, all values are normalized with respect to the base case, which is the version with no query optimizations. “Static Standby” in Figure 5.7 indicates the static standby scheme. By simply putting the modules to standby mode after each access, this scheme is able to achieve a 55% reduction in memory energy consumption of TPC-H queries when compared to the unoptimized case. The energy improvements are less pronounced in the case of handheld queries (37% reduction on the average). This is mainly because of the different number of tables manipulated by these two types of queries. In the TPC-H case, multiple tables are scattered across various banks and hence, there is a potential of placing more memory banks into low-power modes. In the case of handheld queries, there is just one table scattered across multiple banks, which makes putting modules to a low-power mode more difficult as modules are tightly connected, as far as query access patterns are concerned. Also observed from Figure 5.7 is the fact that the dynamic threshold scheme further extends these improvements through its ability to put a bank into any of the possible low-power
modes. On an average, there is a 60% (43%) energy improvement in TPC-H (handheld) queries.

Figure 5.7 also shows the normalized energy behavior of the query-directed scheme (denoted On/Off Instr). It is evident that this scheme outperforms the best hardware-directed scheme (by an average of 10%) in saving the memory energy consumption. This is because of two main reasons. First, when a bank idleness is estimated, the query-directed scheme has a very good idea about its length (duration). Therefore, it has a potential of choosing the most appropriate low-power mode for a given idleness. Second, based on its idleness estimate, it can also preactivate the bank. This eliminates the time and energy that would otherwise have spent in resynchronization. Consequently,
the average memory energy consumption of the query-directed scheme is just 32% of the unoptimized version for TPC-H queries, and 44% in case of organizer (handheld) queries [i.e., an additional 8% (13%) improvement over the hardware schemes for TPC-H (handheld) queries]. The last bar (marked as “History-Based”) in Figure 5.7 will be discussed later in the chapter.

5.4.3. Bank Idleness Analysis

To better understand the energy behavior of memory banks, the most widely-used low-power mode is identified. This is basically the mode in which a bank spends most of its time. For this, the per-cycle energy behavior of each benchmark is profiled. Considering the total execution cycles, it was found that on an average a given application spends only 66% of its time in active mode in the memory. For the rest of the 44% of the total cycles, the memory remains idle. This is illustrated in Figure 5.8 for the hardware-directed scheme with 10, 100, 1000 cycles as thresholds for transition to the standby, nap and power-down modes, respectively. An interesting note is that the power-down mode is the most widely-used low-power mode, and nap is the least frequently used. This proves that when an application goes to the nap mode, it is more likely to continue on to the power-down mode. This also explains why the static standby mode fails to exhibit good energy behavior as compared to the dynamic threshold scheme (which is able to utilize the most preferred low-power mode). In this case, it is seen that the nap mode is insignificant in contributing towards energy savings. Thus, power-down contributes the maximum benefits, and is also the most sought low-power mode of applications.
Figure 5.8. Utilization of various operating modes during query execution. Modules are active most of the time. Power-down is the most frequently-used mode. The threshold of transition is 10, 100, 10000 cycles for standby, nap and power-down respectively.

5.4.4. Performance Overhead Analysis

The proposed techniques are very effective in reducing the memory energy consumption. As mentioned earlier, transitions from the low-power modes to the active mode come with an overhead of resynchronization (in terms of both performance and energy). The energy values reported in previous section take into consideration the extra energy needed to activate the modules as well. This part quantifies the basic performance overheads that are faced in supporting the schemes.

Figure 5.9 shows the performance overheads for both the hardware and software-directed schemes. The static standby scheme has the maximum overhead, which is expected. This is especially the case when queries generate frequent memory accesses. The memory is brought down to the standby mode after each access, and is resynchronized in another access that follows immediately. As a result, the performance worsens
as bad as 28% for the static standby case. On the other hand, for the dynamic threshold scheme, the performance overhead is slightly better since the banks are not blindly put to a low-power mode after each access. This verifies the prediction that when a module goes to low-power mode, it would either remain for a while in that mode or may even be transitioned into a lower power mode. The query-directed scheme has the least overhead (<2%). The main reason for this is the ability of preactivating a bank before it is actually accessed. Therefore, considering both performance and energy results, one may conclude that the query-directed scheme is better than the hardware-directed schemes. However, it is also to be noted that the query-directed scheme requires access to the query optimizer. In comparison, the hardware-based schemes can work with any query...
optimizer. Therefore, they might be better candidates when it is not possible/profitable to modify the query plan.

5.4.5. Sensitivity Analysis

This section studies the sensitivity of the proposed schemes to various key parameters in the system.

5.4.5.1. Number of Banks. The bank size of the memory is varied, keeping the total memory capacity the same. When the size of a bank is increased, the number of banks decreases (for a fixed total memory size). This implies that more data fits into a bank, that is, a table fits into lesser number of banks. This reduces the opportunity to put more banks into a low-power mode. Figure [5.10] illustrates this by showing the average energy savings for the benchmarks. When the bank size is increased from 4MB to 32MB, the savings in energy starts to drop. Also, too many smaller banks lead to increased resynchronization times. So, care should be taken to choose a fitting bank size for a given system; but, this architectural design issue is beyond the scope of this work.

5.4.5.2. Idleness Threshold. In the next set of experiments, the threshold of Section [5.4.3] is tightened to study the alteration in the energy behavior of applications. Figure [5.11] shows the behavior of the queries when the threshold for standby, nap and power-down transition is tightened to 100, 250 and 500 cycles, respectively. When this is compared with Figure [5.8], it is evident that the usage of standby mode decreases drastically. Also, when a memory module enters the standby mode, it has a high probability of
Figure 5.10. Impact of bank size on the energy consumption. As bank sizes increase (number of banks reduce), there is less savings in energy.

getting transitioned all the way to power-down mode. Thus, the behavior is dependent on the chosen threshold. Such techniques of tightening the thresholds can be deployed to reduce the energy consumption. For instance, if the power-down mode is the most frequently-used mode (and if there is a high probability that when a module enters standby, it will get transitioned all the way), modules could be transitioned directly to power-down mode using turn-on/off instructions instead of using hardware-directed mechanisms. However, it should also be noted that the resynchronization times could increase if the module is frequently transitioned back to active mode from power-down mode. It should be noted that changing the threshold affects the behavior of the entire system. If the standby threshold is too low, it leads to many resynchronizations. If is too high, nap and power-down modes are used more frequently, making the impact of
the standby mode insignificant. This is the case for all thresholds. Consequently, care should be taken to ensure that all modes are utilized properly in dynamic schemes.

5.4.6. History-Based Adaptive Scheme

There are two main problems associated with the dynamic threshold scheme. First, it gradually decays from one mode to another (i.e., to get to power-down, then go through standby and nap), though one could have directly transitioned to the final mode if there had been a good estimate. Second, one pays a heavy cost of resynchronization for the memory access if the module gets transitioned. To tackle this problem, a history-based scheme was also implemented and experimented with. This scheme estimates the bank inter-access time, directly transitions to the best energy mode, and activates (resynchronizes) the module so that it becomes ready by the time of the next estimated access. While one could use sophisticated history information to estimate bank inter-access time, this work uses a relatively simple mechanism (keeping hardware implementation energy costs in mind): the estimate for the next bank inter-access time is set to the previous bank inter-access time. History-based scheme requires a mode assignment table that contains the maximum and minimum values of the estimated inter-access time for which a particular mode is optimum. This table can easily be pre-constructed based on the energy values and resynchronization times for the different modes available, and needs to hold only as many entries as energy modes. Once the target power mode is determined, the corresponding resynchronization time is subtracted from the inter-access time estimate, to determine the amount of time to spend in that mode.
Figure 5.11. Mode utilization. When threshold for transition is tightened to 100, 250 and 500 cycles (for standby, nap and power-down), the utilization of power-modes also changes. Intermediate low-power modes are very less utilized.

This scheme was implemented on the existing experimental setup, and studied for its energy and performance behavior. The last bar of Figure 5.7 shows the performance of history-based scheme. There is an average 45% reduction in energy for TPC-H queries and 35% for organizer queries. However, the improvements obtained from the hardware and software-directed schemes of Section 5.3.1 are better than history-based scheme. This is due to the following reason. It is very difficult to predict/reestimate the bank inter-access times accurately. This is partly due to the particular workloads. In particular, the decision support database workloads exhibit complex memory access behavior, and it is not easy to extract exploitable patterns. While one may argue that a more sophisticated predictor could do better, such a predictor would also have substantial energy and performance cost as well.
5.5. Summary

This work is an attempt to study the potential of employing low-power operating modes to save memory energy during query execution. Hardware-directed and software-directed (query-directed) schemes were proposed, which periodically transition the memory to low-power modes in order to reduce the energy consumption of memory-resident databases. The experimental evaluations using two sets of queries clearly demonstrate that query-directed schemes perform better than hardware-directed schemes since the query optimizer knows the query access pattern prior to query execution, and can make use of this information in selecting the most suitable mode to use when idleness is detected. This scheme brings about 68% reduction in energy consumption. In addition, the query-directed scheme can also preactivate memory banks before they are actually needed to reduce potential performance penalty. Overall, it is concluded that a suitable combination of query restructuring and low-power mode management can bring large energy benefits without hurting performance.
Data-Driven Query Optimizations for Memory Resident Databases

Memory subsystem aspects of memory resident database systems were discussed in the previous chapter. This chapter goes one level up and studies the application aspects of a memory resident DBMS. It illustrates the benefits that are got when databases smartly utilize the underlying architecture and system resources. Performance and power optimizations are proposed for embedded database applications after considering the design of the underlying system architecture.

Memory-resident databases (also called in-memory databases [Blo99]) are emerging to be more significant due to the current era of memory-intensive computing. These databases are used in a wide range of systems. With the growing complexities of embedded systems (like real-time constraints), use of a commercially developed structured memory database is becoming important [Bir03]. Even in large-scale computing systems, data is bulk loaded to the memory since I/O accesses to the database are costlier as compared to memory access [McO03]. Memory technology has also grown tremendously over the years, providing larger data storage space at a cheaper cost. In systems where the database resides in the memory, traditional techniques that assume the presence of database on the disk are no longer applicable. Memory hierarchy is the main
bottleneck in such systems. Hence, newer techniques that efficiently exploit both the database architecture and the memory scheme are needed.

In a database system, the data cache misses form as high as 90% of the memory stalls \cite{ADHW99}. Hence, any technique that reduces the cache misses, improving the performance of the memory hierarchy that hosts a database in its main memory, is valuable. The first section (Section 6.1) of this chapter presents a technique called "data windowing" that efficiently exploits the data cache locality in memory-resident databases to improve the system performance.

As mentioned in the previous chapter, recent memory designs have a more structured and partitioned layout in the form of multiple chips, with each chip having memory banks \cite{Ram03}. The numerous features offered by such new memory designs remain largely unexplored when it comes to database queries. The later section of this chapter focuses on this aspect. Here, the basic “data windowing” strategy of Section 6.1 is modified to accommodate the new memory architecture that was proposed in Chapter 5. Basically, databases are made to work in synchronization with the features offered by the underlying memory storage hardware.

\section{Data Windowing}

Obviously, one of the key issues in attaining decent performance from such an embedded database is to exploit data locality (cache behavior). In fact, it is known that
data cache misses can form as high as 90% of overall memory stalls [ADHW99]. Consequently, optimizing database queries to improve their data access patterns can be extremely useful in practice. This can be achieved by maximizing the reuse of the data that resides in the data cache. With data reuse as the primary goal, a technique called data windowing is proposed, which maximally utilizes every block of data brought into the cache. This section elaborates the data windowing strategy.

### 6.1.1. Data Windowing Methodology

A database usually consists of multiple tables. Each table has multiple columns (fields). A query views the table as a collection of data blocks. Depending on the implementation, these blocks can either be file blocks, database page blocks or just user-defined blocks. Based on this, data window is defined as a block of data from the table being accessed by a query. An optimization strategy called data windowing is proposed, which works on data windows. A query uses a set of data windows from each table that it accesses. Scheme for a single table case is presented first and then extend it to the multiple table case. Data windowing consists of the following steps.

1. Consider a single table $T$. The table $T$ is divided into $N$ data windows such that each window ($i \in N$) fits into the cache.

2. For each data window $i$ in $N$, identify a set of queries $Q$ to be performed when $i$ is loaded to the processor. The goal is to reuse the data in a data window to the maximum possible extent.
(3) The number of queries to look at, and the order in which the selected queries are executed is not fixed. That is, the order of accessing the data in a data window is not pre-determined. One way to do it is to just follow the actual execution order as done without this optimization. Given a data window, there is lot of room usually for reusing the data when the execution order is rearranged. A query queue contains the queries that can be handled at a given instance (maximum look-ahead of multiple queries).

(4) The queries are scheduled based on the window accesses. For each window that is accessed, instances of multiple queries that require data from the particular window are scheduled sequentially. That is, queries in the query queue are shuffled in a safe way to reuse the data that is being processed at a given instance. Thus, a schedule queue is built based on the data accessed by queries in the query queue.

(5) The first query finishes executing the particular code section that requires data from the window and passes control to the section of the second query that requires the same set of data. The control goes from one query to another until all queries in the schedule queue are executed.

(6) Now control is shifted to the next data window of the table. Steps 2 to 5 are repeated for this data window. The algorithm continues until all data windows are completed.

As an example, consider a single table T and three queries, Q1, Q2, and Q3. The table T is first divided into data windows (say, T1, T2, T3, and T4) such that each Ti fits
in the cache. Let $Q_{ij}$ refer to the part of $Q_i$ that access $T_j$. A classical query execution would be in this order: $Q_{11}$, $Q_{12}$, $Q_{13}$, $Q_{14}$, $Q_{21}$, $Q_{22}$, $Q_{23}$, $Q_{24}$, $Q_{31}$, $Q_{32}$, $Q_{33}$, $Q_{34}$. The proposed approach, on the other hand, executes this: $Q_{11}$, $Q_{21}$, $Q_{31}$, $Q_{12}$, $Q_{22}$, $Q_{32}$, $Q_{13}$, $Q_{23}$, $Q_{33}$, $Q_{14}$, $Q_{24}$, $Q_{34}$. That is, query parts ($Q_{ij}$) are restructured around the data to make use of the best cache locality. A graphical schematic of this example when applied for $M$ queries and $N$ windows is shown in Figure 6.1.

An extension to the algorithm is the presence of multiple tables. When multiple tables are accessed by a query, each table has its set of data windows. The queries are scheduled based on multiple data windows from multiple tables. For example, suppose there are two tables $T_1$ and $T_2$, and three queries $Q_1$, $Q_2$, and $Q_3$. Let $Q_1$ and $Q_3$ make single scans over the tables, $Q_2$ make a join between $T_1$ and $T_2$. Let each table be divided into two windows: $(T_{11}, T_{12})$ and $(T_{21}, T_{22})$. Then the following are the potential data sets for join:

- $S_1 = T_{11} \times T_{21}$
- $S_2 = T_{11} \times T_{22}$
- $S_3 = T_{12} \times T_{21}$
- $S_4 = T_{12} \times T_{22}$

For data windowing, each $S_i$ is considered in turn, and execute the query portions that access it.

The main goal of the proposed technique is to exploit spatial locality in database queries. Since data windowing changes cursor control based on the data windows, care
should be taken to ensure that the data consistency of the database and data dependencies in the original query execution plan does not change. When queries are reordered, they might corrupt the database or just follow a wrong execution order violating the dependencies.

In the following section, the basic data windowing scheme proposed above is further modified to accommodate the new memory technology that was proposed in Chapter 5. Hence, the performance results for the basic data windowing scheme is not presented here. Interested readers are referred to [PCK04] for detailed experimental results of the basic data windowing scheme.

6.2. Query Restructuring

Chapter 5 proposes two hardware and one query directed mechanisms to reduce energy consumption of queries. The approaches presented there mainly try to optimize energy consumption without modifying the queries themselves (except maybe for the query-directed scheme where turn on/off instructions are inserted). In this section, further efforts are made to demonstrate that even larger energy savings are possible if one has the flexibility of reorganizing query operations. The section demonstrates how this can be achieved in the context of both individual queries and multiple queries (optimized simultaneously). Memory is a common candidate among all queries (since all accesses are memory bound). This implies that memory-oriented multi-query optimization could lead to good energy savings. The main objective in restructuring queries is to increase memory bank inter-access times. Note that when bank inter-access time is increased, the
Figure 6.1. Data windowing applied to M queries that work on data from a single table with N data windows.

given low-power operating mode can be prolonged, thereby feeling the potential impact of resynchronization less (i.e., amortizing the cost of resynchronization); or a switch can be made to a more energy saving mode (as there is a longer idleness), which means more energy savings. Different query restructuring strategies for achieving this is presented.
The primary goal of the heuristic is to cluster the usage of tables from multiple queries together so that the overall table accesses are more uniform. That is, assuming that there are multiple queries to optimize, the objective is to interleave these query executions in such a way that the reuse of individual tables is maximized. That is, when a table is accessed, all other accesses to that table (potentially coming from different queries) should also be issued. Note that this also clusters accesses to the same bank, and tends to increase the bank inter-access times (which is very important from an energy perspective as explained above). The following first presents intra-query restructuring and then inter-query restructuring. After these two steps, bank turn-on/off instructions are inserted at the relevant points depending on the bank access patterns.

**Step 1 (intra-query optimization):** A query is first examined to see if there are any potential reuse regions. If there are any reusable regions, their accesses are grouped together.

The query execution plan is examined for this. The query plan is studied to see if there are any advantages in rearranging the operations (primitives) in a query based on its table usage. Operations that require the same (set of) table(s) are then grouped together (i.e., they are scheduled to be executed one after another). The detailed procedure is shown in Figure 6.2. Each operation in the query plan is first scanned and placed into a table group based on the table(s) that it accesses. Then, the operations are rearranged in the query plan (taking into account the dependencies between them) based on their corresponding table groups. For this, the query plan tree is used. The path from each
leaf node to the root, called *stream*, is investigated. The ultimate goal is to schedule operations (*nodes* in the plan tree) based on their table groups. Operations within one table group (which is currently active) is scheduled first before scheduling the operations from another table group (which is not active) in an attempt to increase the bank inter-access times. That is, a stream is traversed from bottom to top, and each node within the stream is put to the schedule queue (as they are encountered) based on its table group. It should be emphasized that the original semantics of the operations (*constraints*) are preserved in the algorithm. This procedure is repeated for each stream in the tree, and until all streams have the most energy-aware schedule based on their table accesses.

**Step 2 (inter-query optimization):** *Tables are examined to optimize multiple queries simultaneously. For each table that is accessed, all accesses arising from multiple queries to the particular table are grouped together.*

In this step, the *schedule list* from multiple queries are grouped together. Each list is scanned to identify nodes that access a given table. The nodes that access the same table are then scheduled to execute together (without disturbing the dependency constraints). In fact, the nodes from multiple queries are just grouped (combined) not reordered. Thus, in this step, the constraint flow for each *schedule list* (taken care of in Step 1) is automatically maintained. Additional conditional flow checks could be reinforced at this stage if desired. Figure 6.3 shows the regrouping procedure.
table_group is a table-to-operations mapping list

/* identify the group to which an */
* operation belongs */
operation_rearragement (){
  for (each operation in query i) {
    identify the table(s) in i;
    for (each table j in i) {
      add operation to table_group[j];
    }
  }
  schedule_operations();
}

/* schedule operations */ schedule_operations() {
  schedule_list = empty;
  do {
    for (each stream in query plan tree) {
      start from leaf node;
      for (each node in stream) {
        identify its constraint nodes that follow;
        /* the rest are independent nodes */
        group(constraint nodes);
        group(independent nodes);
        check for new violations;
        add new constraints if necessary;
        save the schedule_list;
        move up a node in the stream;
      }
      move to the next stream;
    }
  } until no more changes
}

/* group nodes */ group(node_list) {
  if(node_list is constraint node list) {
    for (each node in node_list) {
      lookup table_group of node;
      add node to schedule_list based on table_group;
      /* preserve the dependency order */
      preserve flow of node_list in schedule_list;
    }
  }
  else /* set of independent nodes */
    add node to schedule_list based on table_group;
    /* no need to preserve constraint flow */
    regroup to put all table_group nodes together;
}

Figure 6.2. Reorganizing operations within a query to optimize for energy
group_multiple_queries {
  for (each schedule_list) {
    do {
      pick an unscheduled node i in schedule_list;
      /* i.e. pick a node without a "complete" tag */
      for (other schedule_lists) {
        if (node j has same table_group as node i) {
          schedule node j after node i;
          mark node j as "complete";
          /* with respect to multi-query schedule */
        }
      }
    } until all node in schedule_list is "complete"
  }
}

Towards the end of the procedure, final_schedule_list stores the entire list of "complete" schedule.

Figure 6.3. Grouping schedule list from multiple queries

**Step 3 (energy optimizations):** Include energy optimizations by inserting On/Off instructions into the final schedule list.

In this step, the access costs are calculated for each operation in the final_schedule_list as shown in Section 5.3.2.2 of Chapter 5. Each operation is attached with an access cost and the turn-on/off instructions are inserted based on the table inter-access times. The methodology used for adding these instructions to the final_schedule_list is the same as in Section 5.3.2.2 of Chapter 5 and the the on/off markers are placed as elaborated in Section 5.3.2.3 of Chapter 5.

As an example, consider two queries Q1 and Q2. Their original query plan is shown in Figure 6.4(i). Q1 is revised as the table accesses are optimizable. Figure 6.4(ii) shows the result after applying Step 1. Step 2 results in the output depicted in Figure 6.4(iii) and eventually to Step 3 as in Figure 6.4(iv).
6.3. Experimental Evaluation

In this section, the query restructuring approach is evaluated by extending the database and queries discussed in Section 5.4.1.2 of Chapter 5. As before, the focus is on memory energy consumption. The impact of the technique on the overall performance is also studied. Towards the end, other alternative options are also elaborated.

6.3.1. Multi-Query Setup

Since simultaneous processing of multiple queries is needed to validate the proposed approach, a combination of queries is considered - termed as *scenarios* in the rest of this chapter. Among the queries considered in Section 5.4.1.2 of Chapter 5, there can be multiple combinations of queries that arrive sequentially, and that which are optimizable using the technique. The various combination (scenarios) of organizer queries and their naming schemes are shown in Table 6.1. For instance, P12 indicates that P1
Table 6.1. Scenarios for organizer queries

<table>
<thead>
<tr>
<th>Type</th>
<th>Legend</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two query combinations</td>
<td>P11</td>
<td>P1 + P2</td>
</tr>
<tr>
<td></td>
<td>P12</td>
<td>P1 + P2</td>
</tr>
<tr>
<td></td>
<td>P23</td>
<td>P2 + P3</td>
</tr>
<tr>
<td>Three query combination</td>
<td>P123</td>
<td>P1 + P2 + P3</td>
</tr>
</tbody>
</table>

Table 6.2. Scenarios for TPC-H queries

<table>
<thead>
<tr>
<th>Type</th>
<th>Legend</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two query combinations</td>
<td>S11</td>
<td>Q6 + Q6</td>
</tr>
<tr>
<td></td>
<td>S12</td>
<td>Q6 + Q3</td>
</tr>
<tr>
<td></td>
<td>S13</td>
<td>Q6 + Q4</td>
</tr>
<tr>
<td></td>
<td>S14</td>
<td>Q6 + Q17</td>
</tr>
<tr>
<td></td>
<td>S23</td>
<td>Q3 + Q4</td>
</tr>
<tr>
<td></td>
<td>S24</td>
<td>Q3 + Q17</td>
</tr>
<tr>
<td></td>
<td>S34</td>
<td>Q4 + Q17</td>
</tr>
<tr>
<td>Three query combinations</td>
<td>S222</td>
<td>Q3 + Q3 + Q3</td>
</tr>
<tr>
<td></td>
<td>S123</td>
<td>Q6 + Q3 + Q4</td>
</tr>
<tr>
<td>Four query combinations</td>
<td>S1111</td>
<td>Q6 + Q6 + Q6 + Q6</td>
</tr>
<tr>
<td></td>
<td>S1234</td>
<td>Q6 + Q3 + Q4 + Q17</td>
</tr>
</tbody>
</table>

is sequentially processed along with P2. The combinations shown in the table are the prominent ones and the behavior of other combinations are very similar to these, hence, they are not included here. The combination scenarios for TPC-H queries are shown in Table 6.2.

6.3.2. Query Energy Evaluation

This section evaluates the query energy of the various scenarios in the previous section. The improvements obtained from the query restructuring heuristic is first studied, and
further extended to combine query restructuring with various hardware and software-directed schemes (of Section 5.3) meant to improve the energy consumption.

Figure 6.5 shows the sole contribution of query restructuring scheme in improving the energy consumption. Energy reduces by an average 55% from the unoptimized version when the query restructuring scheme is used. This verifies that the query restructuring scheme is able to exploit the high data locality in DBMS queries. In other words, by just grouping similar accesses, query restructuring can achieve significant reduction in the energy consumption of multiple queries.

In order to identify the benefits coming solely from Step 1 (intra-query optimization) in the query restructuring scheme, Step 1 and Step 3 are combined and compared with the query-directed scheme (studied in Section 5.3.2 — which is simply Step 3 of query restructuring). Figure 6.6 shows the results. There is up to 19% improvement in energy when operations are shuffled with a query based on their table usage.

When the query restructuring scheme is combined with hardware-directed schemes, there is further improvement in energy (Figure 6.7). Static standby scheme works only for small queries that have a uniform access pattern, but when complex queries are encountered, dynamic runtime scheme outperforms static standby scheme due to its good prediction of the application behavior. This can be seen in Figure 6.5 where the dynamic threshold scheme performs better in TPC-H scenarios than for organizer scenarios. The savings obtained by putting a module into multiple low-power mode for longer periods are more than the savings obtained by periodically putting a module to just standby mode.
Figure 6.5. Contribution of query restructuring towards energy improvements. Energy values shown are normalized to the version with no optimizations.

Figure 6.6. Benefits obtained by restructuring operations within a query (contribution of Step 1).

Software-directed schemes perform similar to dynamic runtime threshold when combined with the query restructuring algorithm. In Figure 6.7, the insertion of explicit turn-on/off instructions improves the energy by an average 78% when compared to the unoptimized version. This result is comparable to the improvements obtained using the dynamic threshold scheme. In fact, the dynamic threshold scheme performs slightly better for some TPC-H queries (like S12, S13, S14). This arises due to the following fact. When multiple queries are combined using query restructuring, it becomes difficult
Figure 6.7. Energy consumption reduces significantly when low-power modes are utilized along with query restructuring scheme. Values shown are normalized to the unoptimized version. Best energy savings comes from a hybrid hardware-software scheme.

to predict the inter-access times since each query has a varying access pattern and combining random access patterns complicates the predictor. Runtime schemes work at the hardware instruction level without any knowledge of the DBMS application. But this illustrates how a simple software technique implemented at the query optimizer, by just knowing the high-level query, is able to achieve improvements as good as an equivalent but expensive hardware technique.

As mentioned before, when queries are restructured and grouped, the access pattern changes. The bank turn-on/off instructions can be inserted only in prominent “hot” and “cold” access regions respectively. There are a few modules, which is beyond the control of software. For instance, turn-on/off instructions are inserted based on tables. A given table could be scattered across many modules. The predictor estimates the inter-access time for which the entire table need to be put to low-power mode. However, even
during a table access, there are regions (modules) that are hardly used. Dynamic runtime scheme is extremely good in handling this situation by its ability to put individual modules to a low-power state based on just that module’s access. This implies that the combination of hardware and software schemes form the best alternative when query restructuring is deployed.

Figure 6.7 also shows the case when both dynamic runtime scheme and the turn-on/off instructions are used in tandem after query restructuring. The benefits obtained from such a hardware-software interaction is prominent. There is an average 90% reduction in the memory energy consumption of applications. In some cases, there is up to 95% improvement in the energy.

6.3.3. Performance Evaluation

Query restructuring and even the combination of various low-power schemes has an impact on the performance. The normalized system-wide performance of the query restructuring scheme is presented in Figure 6.8. It is evident that the performance improves by an average of 48% when multiple queries are restructured and grouped. The improvement in performance is mainly due to the improved locality utilization in the memory hierarchy. Data brought to the cache by one query is extremely reused by other queries. Even though, one could do a detailed cache performance analysis here, those results are not presented here to keep the sections focused and terse.
Figure 6.8. Performance improvement obtained from basic query restructuring over the unoptimized version.

Figure 6.9 shows the normalized performance for the combination schemes as well. When static standby scheme is used with query restructuring, the performance improvements obtained from query restructuring gets negated by the resynchronization overhead from the standby mode for each access. Thus, the performance worsens in some cases by even 65% for complicated queries. But overall, on an average, there is still a 10% performance improvement for all applications.

The turn-on/off instructions has the least performance overhead, and hence, preserves the performance improvements obtained from query restructuring. From Figure 6.9, this combination shows a 47% improvement in performance (negating the improvements obtained from basic query restructuring by a mere 1%). Dynamic runtime threshold on the other hand negates the performance improvements from query restructuring by average 6%.

Combining turn-on/off instructions with dynamic runtime threshold shows an average performance improvement of 45% for applications, which implies a 3% overhead
addition from the low-power schemes towards query restructuring. Thus, it is clearly evident that query restructuring with both turn-on/off instructions and runtime threshold forms the best alternative for both energy and performance improvements. This is due to its ability to reduce the memory energy by 90% from the unoptimized version, while concurrently improving the overall performance by 45%.

6.4. Summary

This chapter showed that banked memory architectures are apt for storing structured databases. Hardware and software-directed (query-directed) schemes proposed in Chapter 5 periodically transition the memory to low-power modes in order to control the energy consumption of memory-resident databases. Data windowing is a simple data-driven strategy that optimizes queries based on the data that is accessed from the relation tables in databases. Query restructuring scheme, a more optimized version of data windowing strategy, works on the memory bank access schemes of Chapter 5. This
acts by providing another degree of query optimization. Multiple queries are optimized based on their table accesses. All accesses to a table are clustered as much as possible. This scheme is able to put modules to a low-power mode for a longer period of time due to fewer table activations. There is up to 90% improvement in energy and 45% improvement in performance when queries are restructured and regrouped based on their table accesses. Thus, the proposed schemes efficiently retrieve the memory data, thus, reducing the energy consumption, while simultaneously improving the performance.
Database Layouts for MEMS-Based Storage Systems

In recent years, advancements have been made even in the data storage components of a system. New storage devices ensure faster data access through advanced features in their designs. This chapter illustrates how such a change in storage technology paradigm would affect the entire system, and also proposes a few remedies. Researchers and designers can in turn use similar approaches to study and propose new system designs and layouts.

Decrease in the access times of disk drives has been minimal in recent years. To overcome this barrier, researchers are taking hybrid approaches that leverage the best of both semiconductor memories and disk drives. The result of this effort is what are known as probe-based systems [fHIPSSC, VDD00, VB03]. These hybrid approaches typically abandon the rotating disk paradigm in favor of simple MicroElectroMechanical Systems (MEMS) to position probe tips over the storage media. Integration of MEMS for data storage applications promises lower cost, higher performance and increased reliability [Jia02].

Data layouts have been studied extensively for traditional magnetic disks. These layouts are not applicable for MEMS-based devices due to the absence of rotational components. Traditional layouts are designed targeting reduction of the disk access
times. Moreover, MEMS-based devices allow bits to be accessed in a random fashion unlike magnetic disks. They also offer a high level of parallelism during data access. A fresh look at the layout for MEMS-based storage systems is thus essential. Some of the major questions that arise when considering data layouts for a MEMS-based storage system are:

- Is there an obvious mapping between the data accessed by an application and the underlying MEMS device that actually stores the data?
- Does the layout of data depend totally on the MEMS device characteristics and physical access patterns? If so, what are the impacts of changing a layout? If not, can there be a global layout?
- What are the feasible ways to store a database in a MEMS-based storage? What are the advantages and disadvantages of each layout? What makes it unique?
- Given a set of layouts, which layout scales well with both the database and the operations done on that database? Does the layout support multi-table operations very well? Does the layout handle multiple queries efficiently?
- What is the impact of these layouts on a given system architecture? Does the layout utilize the data localities available at multiple levels? Is there an improvement in the performance if data is cached at the system level?

Such questions form the motivation for this chapter. Preliminary research verified that data layouts should definitely consider the characteristics of the underlying MEMS
Researchers have already started looking at some of the aspects of data placement in MEMS [GSGN00b, PBL02, YAA03a]. This chapter proposes a mapping between application (high-level) layouts and device (low-level) layouts by taking advantage of the features associated with the underlying MEMS storage device. The MEMS device is considered as a two dimensional storage space and a given database table is mapped in multiple ways onto the device. Row-based, column-based and a combination called object-based layouts are proposed in the following section for both single and multiple table cases. These layouts are assessed in subsequent sections of the chapter by studying the I/O service times, system execution times, and the system cache behavior.

### 7.1. MEMS Device and Characteristics

MEMS-Based Storage Architecture Current MEMS-based storage devices contain a movable sled coated with magnetic media [GSGN00a]. This media sled is spring-mounted above a two-dimensional array of fixed read/write heads (probe tips) and can be pulled in the X and Y dimensions by electrostatic actuators along each edge. To access data, the media sled is first pulled to a specific location (x,y displacement). When
Figure 7.2. MEMS device with respect to traditional terminology

this seek is complete, the sled moves in the Y dimension while the probe tips access the media. Note that the probe tips remain stationary except to adjust for surface variation and skewed tracks while the sled moves. Figure 7.1 illustrates the design of a MEMS-based storage device (as in [GSGN00a]), which forms the basis for this chapter. Researchers have defined an analogy between MEMS-based storage devices and the traditional disk-based storage systems to enable MEMS storage management from the operating system [GSGN00b]. Figure 7.2 shows an example MEMS device with the
analogical terms. Tip region is defined as the media area that is accessible by a tip. Each tip has its own defined tip region, and can access only its tip region. There are 16 tip regions in the given example. A cylinder is defined as the set of all bits with identical X offset within a region (i.e., at identical sled displacement in X). In other words, a cylinder consists of all bits accessible by all tips when the sled moves only in the Y dimension, remaining immobile in the X dimension. There are 5 cylinders in the example. Each cylinder is divided into tracks. A track consists of all bits within a cylinder that can be read by a group of concurrently active tips. For instance, in the considered example, there are 4 tracks within each cylinder. Tracks are divided into sectors. A sector contains bits that are accessed by a tip at a given position. That is, to read from a sector, no X or Y movement is needed. At a given position, a single tip reads all data from a single sector. Typically, it is 8 bytes of data. Hence, there are multiple sectors accessed concurrently in an (x,y) position. New layouts are proposed in this chapter using this basic analogy. The term “logical blocks” is not used in the proposed layouts to keep the discussions simple. The definitions hold good even with/without the concept of logical blocks.

7.2. Data Layouts for a Single Table

The MEMS physical model forms the basis to layouts. Understanding the model is a key to developing effective data layouts. Placing a record-relational table on the device needs careful consideration of the physical model since the access mechanisms (and physical restrictions) impact the performance. For instance, a MEMS device can only
activate 1280 tips simultaneously [GSGN00a], which places a restriction on the number of fields or records that are retrieved from a relational table. This section considers a basic MEMS model to propose various data layouts for a relational table. These layouts are centered around the database table. A table is viewed as a two-dimensional entity, one dimension of records and another with fields. With such a case, the table could be laid out based on the entities that come either from the rows of the table, or columns of the table, or even a combination of records and fields. Each of these possibilities with the MEMS device is considered as a unique layout.

7.2.1. Row-Based Layout (Streaming)

Row based layout is built based on the rows (records) of a table. Let AMEMS be the maximum number of tips that can be simultaneously activated (1280 in this chapter). Each tip (sector) can support B bytes of data (8 bytes in this chapter). If there are R
bytes in the record inclusive of all fields. Then, the number of records that can be stored is

\[ N_r = \frac{A_{MEMS} \cdot B}{R} \]

The tips are viewed as a \( T_r \times T_c \) array (80x80 in this chapter) with a location ID \((i,j)\) for ease of representation (their regions also carry the same ID). Figure 7.3 shows the row based layout. This row based layout is similar to the one presented in [GSGN00b] and [YAA03a]. The proposed approach to data layout design is different from their work. Also, a detailed operation analysis is done here. Records are written as they arrive (streaming model) and each record occupies \( R/B \) tips. If a record does not fit in a single tip, it is striped across multiple sectors (tips) in the X direction. When all \( T_r \times T_c \) tips are used up, the sled is moved down in Y direction by one sector. When all tracks are covered (no more Y displacement), the tip is positioned to the next cylinder (X movement). These steps are repeated until all \( N \) records are written. The access mechanism is similar for read access. The cost of each read/write access is derived, which in turn is used for modeling the total I/O access time for the entire layout.

---

1. In Figure 7.3, the term direction of movement of tips is used to illustrate the order of sector access when the media moves under the tips. It should be noted that it is the media that moves and not the tips.
2. The goal is to present the intuitive interpretation of the layouts in this chapter. Interested readers are requested to refer to the paper [PLC05] for a detailed explanation of the techniques and cost modeling of all layouts.
7.2.2. Column-Based Layout

In this layout, the idea is to spread instances of a single field from multiple records across multiple active tips, so that more instances of the same field from multiple records are read in a single read. Different fields occupy different (adjacent) sectors in the Y direction. At any given point of time, there would be one field from Tr x Tc records available for read. Figure [7.4] illustrates the column based layout.

A given table is laid out in a column major order. Let AMEMS be the number of tips that can be simultaneously activated. Each tip can support B bytes of data. Records from a database table are spread in a vertical format across adjacent Y sectors in the same tip region. Each field occupies a Y sector, which implies the sled needs to move in Y direction to retrieve multiple fields from a particular record. No field occupies multiple sectors in the X direction. Instead if a field requires more sectors, it is spread in the Y direction (within the same tip region) to ensure alignment across multiple cylinders and
even tip regions. Say, if a field F1 is 14 bytes long, it would require 2 adjacent sectors in
the Y direction. Field F1 of all records occupies 2 sectors in Y direction. This scheme
has an advantage. Even if the fields are of varying length, the fields would automatically
be aligned across all records. That is, B bytes of the same field would be read across
all records. The advantage of column based layout lies in its ability to skip unnecessary
fields. For instance, when running a query, all fields of a record are not needed. The
unnecessary fields are basically not retrieved, that is, these fields are skipped during the
Y direction access. The fields are not read and the sled is moved to the beginning offset
of the next relevant record. Column based layout involves a complicated write process
than row based layout, but the reads are simpler and faster than row based layout, as
would be seen from the experimental results.

7.2.3. Object-Based Layout

Each database table has core fields that are unique to the records. Among all fields,
keys are in general more frequently accessed by the queries. There are also other non-
key fields of the table that are often accessed. Based on these facts, an object based
layout is proposed by defining what are known as objects. In the case of a database,
these objects come from a table. For this work, an object is defined as a set of fields.
A core object contains core fields from a table that are repeatedly used in queries. The
rest of the uncommon fields are put in a back object. With the MEMS layout in mind,
the technique strives to place the core object in sectors from where they can be read
with maximum parallelism. The back object are placed in a position where the access
Figure 7.5. Object based layout of 8 records from a table with 4 fields. The MEMS device has 36 sectors in a 3x3 tip region.

costs are high. That way, the access to the core object is cheap and fast. This object is repeatedly accessed as well. The core and back objects are created either by the user or just by using key fields as default core object and the other fields as back object. If preferred, the back objects could be regrouped as sets of further core objects. Each set has a priority assigned based on their need to access, and the group of sets is ordered in decreasing order of the priority to access.

Let $F_c$ be the number of fields in the core object and $F_b$ be the number of fields in the back object. The $F_c$ fields are placed in sectors that are accessible by the same set of active tips. That is, the $F_c$ fields should not need a Y direction movement. On the other hand, the $F_b$ fields are placed in successive sectors in Y direction (within the same tip region). All fields are aligned to make sure that the same field is accessed across all records. Figure 7.5 shows the object based layout. Again, if a field $f$ is longer than $B$
bytes, it is striped across f/B tips in X direction, with boundary and alignment conditions also taken into consideration. Object based layout is good for typical queries since only selected fields are accessed in a query. The unnecessary fields are not read and hence, there is a saving in the access times. Moreover, the records are compactly packed in the object based layout as compared to column and row based layouts. Object based layout exploits data locality very well as would be seen from the experimental results.

7.3. Data Layouts for Multiple Tables

In this section, the ideas developed in the previous section are extended to build layouts for multiple tables in a database. The primary objective behind all multi-table layouts is to place the data in a format that allows concurrent retrieval of multiple fields from multiple tables in an efficient and quick manner.

7.3.1. Row-Interleaved Layout

For this layout, the row based layout is extended in an interleaved fashion. The layout attempts to retrieve the same number of records and fields from multiple tables simultaneously, when the tips are activated. If there are NTab tables in a database, the technique interleave one field from each table before moving to the next field from each of these tables. Figure 7.6 shows the row-interleaved layout. The problem with this layout is that if the number of fields in the tables are not equal, there needs to be a padding (using empty sectors) to align the records. In Figure 7.6 there are empty sectors for B until A4
Figure 7.6. Row-interleaved layout of 4 records from Table A (5 fields) and Table B (3 fields). Device has 36 sectors on a 3x3 array. Note the empty sectors used for alignment between tables A and B (shown as x).

and A5 are covered. This way, equal number of records (with same amount of fields or maximum, in case of non equality) are read from all tables.

The read and write mechanisms are easily extendable from the row based layout (Section 7.2.1) to this layout. During a write, a record from each table is written in a streaming fashion before moving to the next record. It should be noted there are empty sectors left during alignment (these tips are not activated). In each iteration of a read operation, AMEMS tips read equal number of records (minNr/NTab) from each of the NTab tables. This implies that the access time of row based layout is directly extendable for each of the NTab tables. The total access time is the sum of the access (read/write) times for NTab tables, with each of the NTab tables accessed in row based fashion.
Figure 7.7. Column-interleaved layout of 6 records from Table A (3 fields) and Table B (3 fields). MEMS device has 36 sectors in a 3x3 tip array.

7.3.2. Column-Interleaved Layout

In column-interleaved layout, each table occupies a different Y position as shown in the Figure 7.7. This layout can be considered as a horizontal partitioning (logical) of the MEMS storage into NTab portions. Intuitively, the MEMS storage space for each table has reduced by NTab in the Y direction. Each of these NTab tables is arranged in a row based layout and the sum of the access times for NTab tables is the total access time for the entire column-interleaved layout.

7.3.3. Row-Fill Layout

This layout is a pure extension of the row based layout presented in Section 7.2.1. The entire MEMS storage is partitioned logically based on the number of tables that are there in the database. By doing this, the row based layout is extendable directly for
Figure 7.8. Row-fill of 8 records from Table A (3 fields) and 4 records from Table B (3 fields). MEMS device has 36 sectors.

Each partition, aka. for each table that is laid out in the MEMS. For instance, a two table layout looks like as shown in Figure 7.8. This layout is different from the row interleaved layout since the number of records read from each table need not be the same in row-fill layout. In Figure 7.8 table A has 2 records, whereas table B has just 1 record retrieved in the one iteration. Hence, if there is an unequal partition (or even unequal number of fields in tables with equi-sized partitions), the number of records read differ for each table. The advantage of this layout is that the entities are tightly packed in a partition, without any wastage of sectors. Since every table is laid out in the row based format, sum of the access times of each such table gives the total access time for the entire row-fill layout.
7.3.4. Column-Fill Layout

When the column layout in Section 7.2.2 is extended to multiple tables, it gives rise to the column-fill layout. The entire MEMS is partitioned logically based on the number of tables that are in the database. Each table in the partition follows the column based strategy of Section 7.2.2. Figure 7.4 shows the column-fill layout. Total access time for the column-fill layout is the sum of the access times of each table (with column based layout).

7.3.5. Object-Fill Layout

Figure 7.10 shows the object-fill layout for multiple tables. This is just an extension of the object based layout (Section 7.2.3) to multiple tables. As before, the total access time is the sum of the access times for each table.
Figure 7.10. Object-fill layout of 6 records from Table A (3 fields) and 2 records from Table B (3 fields). MEMS device has 36 sectors in a 3x3 tip array

7.4. I/O Performance Evaluation of Layouts

In this section, a setup is designed to perform I/O cost estimation for each layout discussed in Sections 7.2 and 7.3. The following section discusses the experimental setup. Some database primitives are then extended to span a set of standard queries. The I/O costs associated with supporting the execution of these standard queries, is then calculated for each layout. The impact of new layouts on the cache behavior of a system is also examined.

7.4.1. Experimental Setup

A new framework was built to accommodate the layouts discussed in Sections 7.2 and 7.3. This framework models the layouts and the access times associated with each layout. The MEMS device characteristics from [CGN00], and summarized in Table 7.1
Table 7.1. Device parameters used in experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tips available</td>
<td>6400 (80 x 80 tip region)</td>
</tr>
<tr>
<td>Simultaneously active tips</td>
<td>1280</td>
</tr>
<tr>
<td>Per-tip data rate</td>
<td>700 Kb/s</td>
</tr>
<tr>
<td>Sector size</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Average Y seek time</td>
<td>0.35ms</td>
</tr>
<tr>
<td>Average X seek time</td>
<td>0.52 ms</td>
</tr>
<tr>
<td>Settling time</td>
<td>0.22ms</td>
</tr>
<tr>
<td>Average turn around time</td>
<td>0.06ms</td>
</tr>
</tbody>
</table>

are used for the experiments. Storage space for the data is allocated in the framework based on a tables layout and the size of each field in the table. Data is accessed based on a specific access pattern derived from queries run on the database. To ensure consistency, the same set of queries are executed in PostgreSQL [Sys] and the generated query execution plan were ported to the custom implementation. Based on a data access pattern (interpreted from query plan), the framework calculates the times associated with read/write access of each data layout of MEMS. Each page is sized 10K after considering the number of available active tips [GSGN00b]. A 32MB memory limit was also put forth to study the worse case scenarios. Figure 7.11 shows the flow of the experimental framework.

7.4.2. Queries

To evaluate the proposed layouts using standard techniques, a set of queries from TPC-H benchmark [Tra04] were ported to the framework. TPC-H involves complex queries
with a large amount of data accesses. Operations in decision support benchmarks (TPC-D evolved to TPC-H) have good spatial locality with abundant data intensive operations [CTLP+99b]. This assists one to perform a rigorous test of the new layouts. The queries considered Q3, Q4, Q6, and Q17 involve both simple and complex operations. Q6 is a simple query involving a pure lookup, whereas Q3, Q4 and Q17 involve multiple complex operations like join, nest, sort, aggregate, and groupby. These queries work on the CUSTOMER, LINEITEM, ORDERS, and PART tables. These tables were populated using dbgen provided with the TPC-H benchmark [Tra04]. The SQL was generated using the corresponding qgen program. No indexing schemes were supported. Hence, the database is scaled for each query to avoid huge database access times. Care was taken to ensure that the database is big enough and that there are enough X and Y accesses to layouts residing in the MEMS device. The implementation is based on the query plan generated by PostgreSQL [Sys] to maintain consistency. The rest of this section presents the performance of each query in detail. For each query, the various layouts that were proposed were considered. Only the required set of tips are activated during data accesses and if enough data is not available for all tips to read, unnecessary tips remain inactive.
Q6: This query requires a simple lookup of 3 fields from the LINEITEM table. The LINEITEM table was populated with 6 million records using dbgen. The performance of this query when LINEITEM is stored as row, column and object based format is shown in Figure 7.12(a). For object layout, the core object consists of l_orderkey, l_partkey, l_linenumber, l_quantity, l_shipdate, and l_receiptdate. The rest of the 11 fields go as a part of the back object of LINEITEM. It is evident that object layout outperforms all
the other layouts. Column based layout is better than row based layout. The various primitive operations that build these queries were also evaluated. For Q6, 99% of the time is spent on scan of LINEITEM. An elaborate discussion on primitive operations is not presented here to keep the chapter terse.

Q4: Q4 involves a nested loop that scans on LINEITEM and ORDERS. There is a two-level sorting at the final stage of the query for groupby and orderby. LINEITEM was populated with 300,000 records and ORDERS with 150,000 records. This query involves multiple tables, and hence, uses the layouts from Section 7.3. The core object for LINEITEM remains the same across all queries (consists of l_orderkey, l_partkey, l_linenumber, l_quantity, l_shipdate, and l_receiptdate). The core object for ORDERS is o_orderkey and o_orderdate. Other fields of LINEITEM and ORDERS form their respective back objects.

Figure 7.12(b) shows the performance of multi-table layouts for this query. Row-interleaved layout performs the worst. As a result of the equal distribution of records from different tables, this layout spreads the records from a table across numerous cylinders, which directly affects the resultant performance as cylinder movements are costly. In column interleaved format, the tables are made independent through vertical interleaving of records from different tables. This independence in turn decreases the access time by 69%. The same improvement is not achieved in row-fill layout. The improvement in performance is just 38% from row-interleaved layout. It is now evident that
simultaneously reading many records from different tables does not necessarily improve the performance.

The column-fill layout provides a remedy to this by splitting the fields across sectors in Y direction, thus minimizing the junk (unnecessary fields) that is brought into the system from multiple tables. Only the necessary fields are read from the tables, which obviously reduces the access costs. Added advantage is that a lot more records (with just the field of interest) are read from the device. This improves the overall performance by a huge 95% from row-interleaved layout. Surprisingly, the object-fill layout does not perform the best in the case of this query. This is because of the core objects that were chosen. There are a lot of unnecessary fields that exist in the core object (typically using only 2 of the 6 core fields of LINEITEM). But yet, the performance of object based layout is comparable to column-fill layout (just 6% worse than column-fill layout). This proves the flexibility of object based layout in representing multiple queries.

The most improvement comes from column-fill layout since it skips the unnecessary fields during the LINEITEM scan operation (which forms almost 80.27% of the overall query). Thus, for the Q4 query, column-fill layout performs the best closely followed by object-fill layout.

Q3: This query retrieves important orders that are yet unshipped. In this query, there is a scan on three tables: CUSTOMER, ORDERS and LINEITEM. There is a JOIN operation to facilitate this three level lookup. Since this query is really complicated and the emphasis is on studying the layouts, the database was scaled down to 125,000
records for LINEITEM, 65,000 for ORDERS and 15,000 for CUSTOMERS. Enough cylinders and tip regions are occupied to prove the results. Moreover, the results are scalable to more records with the access times being more for such a case.

Figure 7.12(c) shows the performance of multi-table layouts when this query is executed. From this query on, row-interleaved layout is not presented as the performance was consistently bad for that layout. Similar to the previous query Q4, column-fill layout performs the best for Q3 (with access time of 46.1 secs). Object-fill layout follows column-fill layout. The operations within the query were analyzed to study the differences between each layout. Object-fill layout performs best for hash join. The attributes needed are simultaneously read from both tables, because of which the join process is much smoother than other layouts. For column interleaved format, each table needs a Y access during the join. This adds as overhead to the access times. Column-fill layout performs best due to its better performance in LINEITEM scan (which is 42% of the query). As before, object-fill layout falls behind due to many unused fields in core objects. Even in this query, there are just 2 of the core fields that are used.

Q17: By looking to reduce overhead expenses, this query does a nested loop scan of PART and LINEITEM. This query was included in the experiments due to the complexity in its nested loop. The nested loop requires the creation of a sub-plan involving another sequential scan of LINEITEM during the iterations. Basically, LINEITEM is retrieved multiple times from the storage. The database was scaled to contain 150,000
records for LINEITEM and 4500 records for PART. Figure 7.12(d) shows the performance of the multi-table layouts for this query. Column-interleaved layout is performing better than row-fill layout. The reason for this is that the query repeatedly works on the same table and hence, the column-interleaved layout is able to present more records of the same table in one iteration, which in turn, reduces the access times. In the case of row-fill, the number of records retrieved are lesser than column-interleaved layout. But, the column-fill layout optimizes further by reading only very specific fields from the LINEITEM table. Analysis similar to the previous queries are extendable for this query as well.

7.4.3. Effects of Data Caching

The access patterns arising from I/O requests change dramatically when a cache is present in the system. Furthermore, the I/O requests sent to a MEMS device are different from those sent to a traditional storage device. Researchers have already proved that MEMS devices out perform traditional devices in optimizing I/O requests [CGN00]. Since MEMS allows selective activation of tips, only the necessary data is brought into the cache. Effects on caching data from a MEMS device have been investigated for a streaming data layout [YAA03a].

This section investigates the cache behavior that arises when a system supports the proposed data layouts. The layouts have been designed with an aim of exploiting the locality of data that resides on a MEMS device. This section further proves the case. Simulation Setup The I/O access pattern generated by PostgreSQL is captured and passed to
the cache simulator of MEMS framework (Figure 7.11). A 16KB I-L1 cache (2-way), 16KB D-L1 cache (2-way) and a 256 KB (4-way) L2 cache is incorporated in the framework of Figure 7.11. The cache simulator is a version of the SHADE simulator from SUN Microsystems [V604] modified to read the generated access pattern traces. A generated access trace is first loaded to the simulator for execution. The resulting misses at each level of the cache is then captured and analyzed. Some crucial parameters that affect the layouts are varied, and their impact on the cache behavior is studied in detail.

Selection Sensitivity: The selectivity of a primitive SELECT operation (on 3 million records, 64 bytes tuples, varying selectivity) is varied. Basically, in this operation, all records are scanned the database table. The predicate is modified in order to change the selectivity (which results in varied number of output records). Figure 7.13(i) shows the performance of this primitive for a range of selectivities on row, column, and object based layouts. The L1 cache performance is similar to L2 in most of the cases. Hence, only the L2 cache results are presented in this chapter. Moreover, recent L2 caches have a 19-20 cycle latency as compared to a 2-3 cycle latency in L1 [X-b03], which makes L2 caches a potential candidate for performance optimizations.

Looking at Figure 7.13(i), object based layout performs the best. In row based layout, all fields are brought to the cache for all selectivities, which results in generation of many cache requests. The L2 cache miss number barely changes because of the same reason. In the case of column based layout, only the predicate field (1 field in the considered case) is brought into the cache. When selectivity is increased, more misses are generated as more output records are retrieved. Object based layout brings in core fields
Figure 7.13. Cache performance after varying the selectivity (i), projectivity (ii), cache line size (iii), and cache associativity (iv) of primitive operations executed on the proposed layouts.

(4 fields) and hence, works well when selectivity is low. When selectivity is increased more than 25%, more back objects (of selected records) tend to start coming into the cache. This results in more cache pollution as in other layouts. Therefore, object layout converges with other layouts beyond this point.
**Projection Sensitivity:** To enable projectivity variations, the 64 byte record of the SELECT operation in the previous section is divided into 8 equi-sized fields. Figure 7.13(ii) shows the L2 cache performance of the layouts when the projectivity changes 1 to 8 fields. Similar to selectivity, changing the projectivity has minimal impact on row based layout. The miss number remains high irrespective of the number of fields projected. All fields are always retrieved irrespective of the projectivity. The behavior of object based layout is as expected. L2 misses increase when back objects are brought into the cache. When back objects are retrieved, the cache flushes some of the already retrieved (but unused) core objects. The back objects are then loaded. After their use, these back objects are evicted to get back the core objects that were once evicted. Thus object and row based layouts converge once the number of attributes retrieved is more than the core objects. In case of column based layout, the misses consistently increase when more fields are retrieved. This is because in each iteration the same field is retrieved from multiple records. The cache is full of single-field tuples at an given of time. A miss gets generated for each additional field that needs to be retrieved. Hence, one can explain the linear behavior of column based layout as more fields are retrieved. Column based layout shows the best behavior for projectivity variations.

**Sensitivity to Cache Parameters:** The size of the cache line affects the cache performance of layouts. As seen in Figure 7.13(iii), when the line size is increased, the performance improves. When the size is increased, a given cache line is able to accommodate more fields (of same record, and also from multiple records). Hence, there is a consistent improvement in cache misses.
One would expect analogous behavior on changing the associativity (as changing the cache line size). But the behavior changes on varying associativity. Note in Figure 7.13(iv) that the behavior of row based layout (with respect to other layouts) has changed as compared to its behavior on changing cache line size. This is because, in a direct mapped configuration for row-based layout, the entire set of fields gets accommodated in a single block (which is not possible for a small line size). This reduces the misses generated. But when the associativity increases, all layouts converge generating the same number of misses. Again, object based layout performs the best.

The L2 cache size was varied from 32KB to 2MB. The misses did not reduce significantly and there were no interesting derivations. Hence, the results are not presented here. Increasing the cache line size seems to be a better alternative.

7.5. Discussion

Single table layouts were purely extended to form multi-table layouts. From the analysis, it is clear that a pure row based layout is not applicable for multiple tables. Row-interleaved layout is the first layout that comes out of this process, and it does not scale very well and also wastes storage space. Other new layouts were derived. Each of these layouts is good for a specific purpose. Row-fill layout is an optimized version of row-interleaved layout. Column-fill is an extended version of column based layout from single table to multiple tables. It performs well for queries that retrieve fewer fields. Object-fill layout is an extension of object based layout in single tables. This layout is very consistent in the performance. It supports any kind of queries. If
the core objects are smartly allocated, object-fill format outperforms all other layouts. Column-interleaved layout is good for queries that require multiple records from a table, provided there are no sudden transitions from one table to another. The interleaved layouts are not very dynamic. Like in a real scenario, if a table is created on the fly, both row-interleaved and column-interleaved layouts will require an entire reorganization of records. This could be avoided in fill based layouts if additional space is left for future partitions. Such dynamic strategies require careful analysis, and forms a part of the future work.

A summary of the layout and the circumstance that leads to its best performance is shown in Table 7.2. Experimental results and few intuitive derivations were used for this purpose. Some of the core operations have been covered in that table. Other operations like sort, aggregate, groupby could be categorized in the scan type as these operations mostly involve retrieving records in a way similar to one of the fashions listed in the table for scan. Table 7.2 could be extended for many more operations, but, it is beyond the scope of this work. For a database system, one of the layouts can be chosen from the table by considering the type of queries that would be run on the system.

7.6. Summary

MEMS-based storage systems are a viewed as an alternative to disk-based storage systems due to their improved design. But, there are no standard layouts for the data or even any well-defined schemes for the applications to access the data in the device. The MEMS device characteristics were studied to propose various layouts for a database.
Table 7.2. Summary of layout performance

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Row Interleaved</th>
<th>Column Interleaved</th>
<th>Row Fill</th>
<th>Column Fill</th>
<th>Object Fill</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scan</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one field, one table</td>
<td>↔</td>
<td>↔</td>
<td>↔</td>
<td>↔</td>
<td>Depends on initial positioning</td>
<td></td>
</tr>
<tr>
<td>one field, multiple tables</td>
<td>↔</td>
<td>↓</td>
<td>↔</td>
<td>↑</td>
<td>-do-</td>
<td></td>
</tr>
<tr>
<td>multiple fields, one table</td>
<td>↔</td>
<td>↑</td>
<td>↑</td>
<td>↓</td>
<td>For small number of fields, object-fill is better, otherwise, column-interleaved is better</td>
<td></td>
</tr>
<tr>
<td>multiple fields, multiple tables</td>
<td>↓</td>
<td>↔</td>
<td>↑</td>
<td>↓</td>
<td>↑</td>
<td>Smart selection of core objects is suggested</td>
</tr>
<tr>
<td><strong>Join</strong></td>
<td>↔</td>
<td>↓</td>
<td>↔</td>
<td>↑</td>
<td>↑</td>
<td>Column-fill layout works well for single field operations</td>
</tr>
<tr>
<td><strong>Nest</strong></td>
<td>↓</td>
<td>↑</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
<td>Object-fill layout works well for multi-field operations. Prefer column-interleaved layout for nest iterations. Column-fill is for nests that rely just on a single field</td>
</tr>
</tbody>
</table>

Legend: ↓ = bad, ↔ = average, ↑ = good, ↑ = very good

These layouts map the database onto the device using a dimension based partitioning of both MEMS and the database. There are three single table layouts: row based, column based and object based layouts. All single table layouts are extended to form multi-table layouts: row-interleaved, column-interleaved, row-fill, column-fill, and object-fill layouts. Single table layouts can also be viewed as a special case of multi-table layouts.
The presented experimental results prove that column based layouts outperform row based layouts for both single and multiple tables. Object based layouts perform consistently across all queries and database types. The core objects need to carefully defined. If many fields are included in the core object, the performance might be hurt. Object based layout provides excellent data cache locality.

In conclusion, each layout has its own advantages and disadvantages as stated in Table 7.2. A designer or user could implement the database based on a particular layout depending on the type of queries that would be run on the database. Overall, object-fill layout is suggested for any database as the performance is consistent for multiple and single table operations. Column-fill layout is good when the fields of interest are very less, that is, when the queries work on fewer fields from the database.
CHAPTER 8

Workload Characterization Framework for Data Mining

Applications

With the enhanced features in recent computer systems, increasingly larger amounts of data are being accumulated in various fields. Recent trends indicate that data being collected doubles every year. A survey done by Intel Corporation indicates that an average person collects 800MB of data a year [Cor05]. This does not include the scientific data collected by corporations and research institutions. Future systems are bound to be data intensive. Data would be processed either offline using data collection tools, or real-time using a streaming data model. Data analysis tools and frameworks are bound to become more sophisticated. Such complex analysis tools are bound to be performance-hungry due to the amount of data that they require to handle.

On the other hand, recent computing trends suggest that the system performance (data based on memory and I/O bound workloads like TPC-H) has been improving at a rate of 10-15% per year, whereas, the volume of data that is collected doubles every year. The important obstacle is the fact that the performance of computer systems is improving at a slower rate when compared to the increase in the data and the requirements of data analysis. Having observed this trend, researchers have focused on efficient implementations of different data mining algorithms. Among these, a major approach
taken is the development of parallel and distributed versions of such algorithms. While these algorithms have been efficiently improved, the basic characteristics that define these algorithms remain under studied. Such information in turn can be utilized during the implementation of the algorithms and the design/setup of the computing systems. Understanding the performance bottlenecks is essential not only for processor designers to adapt their architectures to data mining applications, but also for programmers to adapt their algorithms to the revised requirements of applications and architectures. This forms the motivation for this work.

The rest of the chapter is organized as follows. Section 8.1 discusses the methodology used in this work to perform the characterization. The details of the workloads used in this characterization study (the workloads form the NU-MineBench benchmark suite) are provided in Section 8.2. Following which, Section 8.3 elaborates the sophisticated analysis framework that was developed to evaluate and analyze the applications. The characteristics and results of scalability studies are presented in Section 8.4.

8.1. Methodology

The primary goal of this study is to gain an in-depth understanding of the characteristics of data mining applications. In other words, a detailed workload characterization of data mining applications is the primary goal of this work. This is achieved by studying data mining applications through benchmarking schemes. The ultimate goal is to assemble a benchmark that effectively represents data mining applications.
Benchmarks play a major role in all domains. SPEC [Sta01] benchmarks have been well accepted and used by several processor manufacturers and researchers to measure the effectiveness of their design. Other fields have popular benchmarking suites designed for the specific application domain: TPC [Tra04] for database systems, SPLASH [WOT+95] for parallel machine architectures, MediaBench [LPMS97] for media and communication processors. Benchmarks do not only play a role in measuring the relative performance of different systems. They also aid programmers in the specific domain in various ways. For example, a programmer implementing a new data mining application can compare the performance (in terms of output quality, scalability, and execution time) of the new application to the applications in the benchmarking suite. In addition, the programmer can use certain types of algorithms and programming styles from the applications in the existing suite.

Although there has been previous work analyzing individual data mining applications [BF98, KQH98], analyzing the behavior of a complete benchmarking suite will certainly give a better understanding of the underlying bottlenecks for data mining applications. This work analyzes data mining applications from many perspectives and presents the key characteristics of these applications. Another important aspect of this study is implementing and analyzing scalable versions of the benchmark applications. As the size of the available datasets and their high-dimensionality grow, high performance computers are becoming essential platforms to execute the data mining applications.
The first question that is addressed by this work is the uniqueness of data mining applications. If they are unique, the next task lies in identifying the characteristics that make them distinct from other existing applications and domains. In order to answer these questions, this work takes up an extensive characterization study. The various perspectives of the characterization study is shown in Figure 8.1. Like traditional studies, applications are studied from an algorithmic perspective by analyzing their execution times, run times (order of execution) and other high level metrics. Various system characteristics involving runtime overheads, system resources (locks), synchronization costs are also studied. A low-level architectural study is then performed extensively to understand the system architecture performance. This includes several components like the cache, processor, memory, and disks. Another important factor in data mining is the input data sizes. This study also characterizes the workload behavior based on input data sizes.

Scalability is deemed to be an important (and unavoidable) requirement in future systems. Scalability comes in many forms in both hardware and software. Traditionally, scalable versions of applications are developed by purely extending high-performance, parallel and distributed paradigms to the respective serial versions of the algorithms. This study analyzes various traditional scalability approaches as well. This is crucial since the final goal of this work is to enable development of high performance data mining systems and algorithms. Hence, this work studies the scalability of applications based on processor resources and data sizes (dimensionality, etc) as well.
The following section describes the workload named as NU-MineBench that is developed and used for this study. Following that, a brief description of the analysis framework is presented. Then finally, the characteristics of data mining applications are discussed in detail.

### 8.2. Data Mining Workload

Data mining, a technique to understand and present raw data, is becoming popular and is starting to be used in a variety of fields like marketing, business intelligence, scientific discoveries, biotechnology, internet searches and multimedia. In data mining, the process of extracting information from raw data is automated, and mostly predictive. This is the aspect that makes data mining different from OLAP and common database...
Data mining is essential to automatically find useful (predictive) information from such large input data. For instance, a cellular phone company would apply clustering to find a probable customer segment (regular travelers?) suitable for a new cellular phone plan.

Data mining applications are broadly classified into classification, clustering, association rule mining, sequence mining, similarity search, text mining, multimedia mining, and other categories based on the nature of their algorithms. Algorithmically, each domain is different. For instance, classification algorithms perform mining by building predictive models that represent data classes or concepts, whereas, clustering involves grouping a set of physical or abstract objects into categories of similar objects. Figure 8.2 shows a taxonomy of data mining applications (partly based on [HK00]). As can be seen, each data mining application differs by the nature of mining it performs.

A classification of data mining applications was shown in Figure 8.2. A subset of application domains of Figure 8.2 is used to establish NU-MineBench, a benchmarking
suite containing well-known (and representative) data mining applications. The selection of categories as well as the applications in each category is based on how commonly these applications are used in industry and how likely to be used in the future, thereby achieving a realistic representation of the existing applications. Another concern of the algorithm selection is the scalability when executing on parallel or distributed systems.

NU-MineBench has applications from several domains and categories. The applications as well as important characteristics of the applications are listed in Table 8.1. Note that these are full-fledged application implementations of these algorithms (as against stand-alone algorithmic modules), which have been extensively optimized to remove all implementation inefficiencies. Also, the algorithms and the set of operations performed within them can be seen in commercial data mining tools, like Clementine (SPSS Inc.), Intelligent Data Miner (IBM Corporation) and SAS Enterprise Miner (SAS Institute Inc.).

There are several types of clustering algorithms in this study. The first clustering application in MineBench is K-means [Mac67]. K-means is a partition-based method and is arguably the most commonly used clustering technique. K-means represents a cluster by the mean value of all objects contained in it. Given the user-provided parameter k, the initial k cluster centers are randomly selected from the database. Then, K-means assigns each object to its nearest cluster center based on the similarity function. For example, for spatial clustering, usually the Euclid distance is used to measure the closeness of two objects. Once the assignments are completed, new centers are found by finding the mean of all the objects in each cluster. This process is repeated until two consecutive
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-Means</td>
<td>Clustering</td>
<td>Mean based data partitioning method</td>
</tr>
<tr>
<td>Fuzzy k-Means</td>
<td>Clustering</td>
<td>Fuzzy-logic based data partitioning method</td>
</tr>
<tr>
<td>BIRCH</td>
<td>Clustering</td>
<td>Hierarchical data segmentation method</td>
</tr>
<tr>
<td>HOP</td>
<td>Clustering</td>
<td>Density based grouping method</td>
</tr>
<tr>
<td>Naive Bayesian</td>
<td>Classification</td>
<td>Statistical classifier</td>
</tr>
<tr>
<td>ScalParC</td>
<td>Classification</td>
<td>Decision tree based classifier</td>
</tr>
<tr>
<td>Apriori</td>
<td>ARM</td>
<td>Horizontal database, level-wise mining based on Apriori property</td>
</tr>
<tr>
<td>Eclat</td>
<td>ARM</td>
<td>Vertical database, equivalence class based method</td>
</tr>
<tr>
<td>SNP</td>
<td>Bayesian Network</td>
<td>Hill-climbing search method for DNA dependency extraction</td>
</tr>
<tr>
<td>GeneNet</td>
<td>Bayesian Network</td>
<td>Microarray based structure learning method for gene relationship extraction</td>
</tr>
<tr>
<td>SEMPHY</td>
<td>Expectation Maximization</td>
<td>Phylogenetic tree based structure learning method for gene sequencing</td>
</tr>
<tr>
<td>Rsearch</td>
<td>Pattern Recognition</td>
<td>Stochastic Context-Free Grammar based RNA sequence search method</td>
</tr>
<tr>
<td>SVM-RFE</td>
<td>Support Vector Machines</td>
<td>Recursive feature elimination based gene expression classifier</td>
</tr>
<tr>
<td>PLSA</td>
<td>Dynamic Programming</td>
<td>Smith Waterman optimization method for DNA sequence alignment</td>
</tr>
</tbody>
</table>

iterations generate the same cluster assignment. The clusters produced by the K-means algorithm are sometimes called “hard” clusters, since any data object either is or is not a member of a particular cluster.
The Fuzzy K-means algorithm [Bez81] relaxes this condition by assuming that a data object can have a degree of membership in each cluster. The Fuzzy K-means assigns each pair of object and cluster a probability. For each object, the sum of the probabilities to all clusters equals to 1. Compared to the Euclid distance used in K-means, the calculation for the fuzzy membership results in higher computational cost. However, the flexibility of assigning objects to multiple clusters might be necessary to generate better clustering qualities.

BIRCH [ZRL96] is one of the hierarchical clustering methods that employ a hierarchical tree to represent the closeness of data objects. BIRCH first scans the database to build a clustering-feature (CF) tree to summarize the cluster representation. Then, a selected clustering algorithm, such as K-means, is applied to the leaf nodes of the CF tree. For a large database, BIRCH can achieve good performance and scalability. It is also effective for incremental clustering of incoming data objects.

Density-based methods grow clusters according to the density of neighboring objects or according to some other density function. HOP [EH98], originally proposed in astrophysics, is a typical density-based clustering method. After assigning an estimation of its density for each particle, HOP associates each particle with its densest neighbor. The assignment process continues until the densest neighbor of a particle is itself. All particles reaching this state are clustered as a group. HOP is highly scalable when applied to large databases.

The Naive Bayesian classifier [DP96], a simple statistical classifier, uses an input training dataset to build a predictive model (containing classes of records) such that
the model can be used to assign unclassified records into one of the defined classes. It is based on Bayes’ Theorem. It is comparable in performance to decision tree based classification algorithms, and exhibits high accuracy and speed when applied to large databases. ScalParC, a scalable decision tree based classifier \cite{JKK98}, builds the decision tree by recursively splitting the training dataset based on an optimal criterion until all records belonging to each of the partitions bear the same class label.

Apriori \cite{AMS+96} is arguably the most influential association rule mining (ARM) algorithm. It explores the level-wise mining using the Apriori property: all nonempty subsets of a frequent itemset must also be frequent. Eclat \cite{Zak99}, another ARM algorithm, uses a vertical database format instead of the hash trees (horizontal format) as in apriori. This enables breaking the search space into small, independent, and manageable chunks. Efficient lattice traversal techniques are used to identify all the true maximal frequent itemsets.

The primary goal of Bayesian network based methods is to build a learning network that represents the input data set. This is done by identifying the statistic relationship between the several variables present in the input data. A scoring function is introduced that evaluates a network with respect to the training data and outputs a value that reflects how well the network scores relative to the available data. Then the possible network structures are searched to find the best scored network, which usually is considered to be the network learned from the data. In general, the search problem is NP-hard, most algorithms use heuristic search methods, such as the MCMC (Markov Chain Monte Carlo)
sampling, K2, Simulated Annealing etc., of which the greedy hill-climbing algorithm is the most efficient and popular approach [CDD+05, DCD+05].

Single nucleotide polymorphisms (SNPs) are DNA sequence variations that occur when a single nucleotide is altered in the genome sequence. Identifying valid sequence variations is a goal of genomic research. This work focuses on a version of SNP that uses the hill climbing search method [CDD+05]. This method first selects a specific point (an initial Bayesian Network structure) in the search space as the starting point. Algorithm then searches all the nearest neighbors for the current point in the search space, and then selects the neighbor that has the highest score as the new current point. This procedure iterates until no neighbor has higher score than the current point (i.e., reached a local maximum). GeneNet [CDD+05, DCD+05] uses a similar hill climbing algorithm as in SNP. The difference here is that the input data is the microarray data, which requires lot more computations to perform the learning process. There are lot more variables used during the learning.

SEMPHY [CDD+05, DCD+05] is a structure learning algorithm that is based on phylogenetic trees. Phylogenetic tree represents the genetic relationship of species by a tree where closely related species are placed in nearby branches. For DNA/protein sequences from different species, a phylogenetic relationship among them can be inferred to reflect the course of evolution. The goal of this algorithm includes searching for the best tree topology and the best branch lengths representing the distance between the two neighbors. Note that there are numerous branch length probabilities for each topology.
SEMPHY uses Structural Expectation Maximization (probability estimation) algorithm to address this complication.

Typically, RNA sequencing problems involve searching the gene database for homologous RNA sequences. Rsearch \[\text{CDD}^{+05}, \text{DCD}^{+05}\] uses a grammar based approach to achieve this goal. Rsearch uses SCFG (Stochastic Context-Free Grammar) to build and to represent a single RNA sequence with its secondary structure, and utilizes a local alignment algorithm named CYK algorithm, which is a decoding algorithm for SCFG, to search a database for homologous RNAs.

Support Vector Machines Recursive Feature Elimination (SVM-RFE) \[\text{CDD}^{+05}, \text{DCD}^{+05}\] is a feature selection method that uses SVM techniques to refine and identify the optimum feature set in the feature data. It selects or omits dimensions of the data depending on a performance measure of SVM classifier. It is much more robust to data overfitting than other methods, including combinatorial search. SVM-RFE uses microarray data as the input data for analysis. SVM-RFE is used extensively in disease finding (gene expression). It eliminates gene redundancy automatically and yields better and more compact gene subsets. The selection is obtained by recursive feature elimination process: at each RFE step, a gene is discarded from the active variables of a SVM classification model. The features are eliminated according to a criterion related to their support to the discrimination function and the SVM is re-trained at each step.

Sequence alignment is an important tool in bioinformatics used to identify the similar and diverged regions between two sequences, e.g. biological DNA/protein sequences or text strings. PLSA \[\text{CDD}^{+05}, \text{DCD}^{+05}\] uses a dynamic programming approach to
solve this sequence (string) matching problem. It is based on the algorithm proposed by Smith and Waterman, which uses the local alignment to find the longest common substring in sequences. Since this method is dependent on the sequence length, it is computationally very intense.

In the subsequent sections, full-fledged applications of the above methods are taken and analyzed in detail using an extensive evaluation framework.

8.3. Evaluation and Analysis Framework

The goal of this work is to extensively analyze the NU-MineBench applications discussed in Section 8.2 from multiple perspectives. The multiple evaluation aspects were shown in Figure 8.1. In order to perform the evaluation and the analysis, an extensive framework was developed. Figure 8.3 shows the framework, which basically includes two parts: the data collection component and the analysis component.

The evaluation (data collection) part includes real evaluation setups and simulators. The applications are hand optimized and compiled using well known compilers. Then the application parameters and the actual executables are fed into either real systems or simulators to get the performance numbers. The performance numbers are collected using profiling tools. Profiling is done at multiple levels to ensure comprehensive analysis. Functional profilers, system profilers and architectural profilers are used.

There are several parameters varied in the experiments. Input data sizes, processor configurations (speed, type, branch), memory hierarchy configurations (cache, memory) were a few of the core parameters varied. As can be seen, each of these parameters
Figure 8.3. The experimental evaluation and analysis framework used for studying NU-MineBench Applications

produce a different output. Some key output parameters include the execution times, performance overheads and other real performance numbers, hardware events, function overheads, system locks, etc. With such a large amount of performance data produced for analysis, it becomes complex and tedious to analyze the patterns and characteristics of data. Hence, an analysis framework was also added to the framework to automate and ease up the analysis phase.
Now the rest of this section elaborates each component of the framework in detail. Specifically, the setup used in this work includes the following.

8.3.1. Evaluation Hardware and Simulator Setup

An Intel IA-32 multiprocessor platform is used for evaluation of applications on a real system. This setup consists of an Intel Xeon 8-way Shared Memory Parallel (SMP) machine running Red Hat Linux Advanced Server 2.1 operating system. The system has a 4GB shared memory and 1024 KB L2 cache for each processor. Each processor has 16KB non-blocking, integrated L1 instruction and data caches. The compilers used for creating the application executables were gcc 3.2 and Intel C++ compiler 7.2.

Due to the limitations in real systems (like inability to change memory and cache sizes), a simulator was also deployed for conducting evaluation experiments that needed significant changes in the system configurations. The application codes were compiled using a modified version of gcc 2.91. The executable is interfaced to an enhanced version of the SimpleScalar simulator [Aus] to form a complete system. The intermediate interface (invoked from above by the data intensive applications) provides a set of operating system calls (on Linux kernel 2.4.25), which in turn invokes the SimpleScalar simulator. The SimpleScalar simulator models a modern microprocessor with a five-stage pipeline: fetch, decode, issue, write-back, and commit. The hardware techniques are implemented within the framework of the sim-outorder tool from the SimpleScalar suite. The modeled architecture has a 16KB direct-mapped instruction cache and a 8KB direct-mapped data cache (each of 32 byte-length). A 256-entry full associative
TLB with a 30-cycle miss latency also exists in the system. The off-chip bus is 32 bit-wide. The simulator outputs several characteristics ranging from processor efficiency, issue queue load to memory bandwidth, cache misses. The simulator also gives high level (function/procedural) details.

8.3.2. Evaluation Software

VTune Performance Analyzer [Int05] is for profiling the functions within the NU-MineBench applications, and for measuring their execution times on real systems. To trace subroutine calls, VTune calling graph is used, which presents a hierarchical decomposition of the execution time. VTune counter monitor provides a wide assortment of metrics. Different characteristics of the applications are extracted: execution time, fraction of time spent in the OS space, communication/synchronization complexity, I/O complexity, memory behavior, and CPI behavior. Related VTune parameters are used to collect data for these properties.

In parallel implementations of the applications, OpenMP [Con05] pragmas and MPI [MD04] communication paradigms were used. OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism. Due to its simplicity, OpenMP is quickly becoming one of the most widely used programming styles for SMPs. In SMPs, processors communicate through shared variables in the single memory space. Synchronization is used to coordinate processes. On the other hand, MPI is a library specification for message
passing between distributed machines (either physically, and virtually distributed memory). MPI was designed for high performance on both massively parallel machines and on workstation clusters.

Intel VTune provides the aggregate time spent on different types of functions and pragmas that are used for job parallelization and synchronization (includes individual loops and routines as well). This way, one can accurately measure the time spent on synchronization, and other relevant contentions.

**8.3.3. Evaluation Datasets**

Input data is an integral part of data mining applications. The data considered for the experiments are either real data got from various fields or widely-accepted synthetic data generated using existing tools that are used in scientific and statistical simulations. During evaluation, multiple data sizes were used to investigate the characteristics of the NU-MineBench applications.

For non-bioinformatics applications, three input data were used in 3 different sizes: Small, Medium, and Large. For ScalParC and Nave Bayesian, three synthetic datasets (see Table 8.2 - “Classification”) were generated by the IBM Quest data generator [Sys04]. The notation Fx-Ay-DzK denotes a dataset with Function x, Attribute size y, and Data comprising of z*1000 records. Function 26 is a relatively complex function and produces large trees. Apriori and Eclat use three synthetic datasets from IBM Quest data generator (see Table 8.2 - “ARM”). D denotes the number of transactions, T is the average transaction size, and I is the average size of the maximal potentially
Table 8.2. Classification and Association Rule Mining Dataset Characteristics (Dataset size in MB).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classification Parameter (Size)</th>
<th>ARM Parameter (Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>F26-A32-D125K (27)</td>
<td>T10-I4-D1000K (47)</td>
</tr>
<tr>
<td>Medium</td>
<td>F26-A32-D250K (54)</td>
<td>T20-I6-D2000K (175)</td>
</tr>
<tr>
<td>Large</td>
<td>F26-A64-D250K (108)</td>
<td>T20-I6-D4000K (350)</td>
</tr>
</tbody>
</table>

large itemsets. In Table 8.2, the number of items is 1000 and the number of maximal potentially large itemsets is 2000. For HOP and BIRCH, three sets of real data were extracted from a cosmology application, ENZO [NSLD99], each having 61440 particles, 491520 particles and 3932160 particles. A section of the real image database distributed by Corel Corporation is used for K-means and Fuzzy K-means. This database consists of 17695 scenery pictures. Each picture is represented by two features: color and edge. The color feature is a vector of 9 floating points while the edge feature is a vector of size 18. Both K-means implementations use Euclid distance as the similarity function and execute it for the two features separately. Since the clustering quality of K-means methods highly depends on the input parameter k, both K-means were executed with ten different k values ranging from 4 to 13.

For the bioinformatics applications, the datasets were provided by Intel Corporation [CDD+05, DCD+05]. SNP uses the Human Genic Bi-Alletic Sequences (HG-BASE) database [BLS+00] containing 616,179 SNPs sequences. For GeneNet, the microarray data used for this study is assembled from [SSZ+98]; they are the most
Table 8.3. SEMPHY datasets and their characteristics

<table>
<thead>
<tr>
<th>Dataset:</th>
<th>S</th>
<th>M</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taxa number</td>
<td>53</td>
<td>108</td>
<td>220</td>
</tr>
<tr>
<td>Sequence length</td>
<td>394</td>
<td>397</td>
<td>389</td>
</tr>
</tbody>
</table>

popular cell cycle data of Yeast. SEMPHY considers three datasets from Pfam database [BCD+04]. The three datasets (labelled in this work as S,M,L) and their characteristics is shown in Table 8.3. The software and the corresponding dataset for Rsearch were obtained from [Lab05]. The experiments use the sequence “mir-40.stk” with the length of 97 to search a part of database “Yeastdb.fa” with size of 100KB. SVM-RFE uses a benchmark microarray data set on ovarian cancer [AM02]. This dataset contains 253 (tissue samples) x 15154(genes) expression values, including 91 control and 162 ovarian cancer tissues with early stage cancer samples. For PLSA, nucleotides ranging from 30K to 900K length are chosen as test sequences. Since true sequences can seldom satisfy this specific size, some artificial sequences were used in the experiments [CDD+05]. To make the experiments more comprehensive, several real DNA sequences were also chosen from a test suite provided by the bioinformatics group at Penn Stat University. The longest sequence pair used here is named TCR where the human sequence is 319,030 bp long and the mouse sequence is 305,636 bp long.
8.3.4. Analysis Setup

As mentioned before, a large amount of output performance data is produced from the evaluation (both simulation and real system evaluations). This section presents the analysis toolkit that was built to automate and ease this complex analysis phase. Figure 8.4 presents the flow of the toolkit.

The analysis toolkit is an assembly of various analysis software. It encompasses a mix of a statistical tool, a data mining software and a visualization tool. To be specific, the following is the flow. The output produced by the evaluation software is saved to a spreadsheet with each row representing a run of the application with a specific configuration. This spreadsheet is fed to a data mining software, Clementine from SPSS Inc., to identify similarities and patterns in the data. This data mining software provides well-known data mining algorithm modules that can be applied to the data. Several algorithms are applied to the output data to extract meaningful patterns within the data. This gives an idea about the unique characteristics of applications within the suite. Then, a mix of Clementine and Microsoft Excel enhanced with statistical models is used to extract so called kernels (the core computationally intense modules) within each application. Together, this toolkit also produces visual models of the patterns that are produced.

The subsequent sections discuss the several patterns that were seen in the applications. The use of the above tools become apparent as the results are discussed. The patterns extracted prove the unique characteristics of data mining applications.
Figure 8.4. The toolkit used for analyzing the performance of NU-MineBench applications
Figure 8.5. Measures of interest for evaluation and analysis

8.4. Characteristics of Data Mining Applications

8.4.1. Uniqueness of Characteristics

There were several measures and metrics of interest that were collected during the evaluation of NU-MineBench applications. These measures were chosen with care to identify the performance from the system and application perspectives. Figure 8.5 shows the collected measures used for the analysis.
For all NU-MineBench applications, the measures of Figure 8.5 were collected. These measures were then normalized correspondingly for analysis. For instance, clock-ticks were normalized using instructions retired to form CPI and IPC. L2 cache misses were studied using miss ratios and references per instructions. Once these values were normalized, they were then fed into the analysis framework. Statistical analysis tools were invoked on this data to extract the characteristics of data mining applications.

The first question that is to be addressed is the uniqueness of data mining applications. For this, the NU-MineBench applications were compared against applications from other prominent benchmark suites. Specifically, data mining applications were compared against compute intensive applications, multimedia applications, streaming applications and database applications to identify the core differences from the other benchmarks. Applications were taken from integer applications benchmark (SPEC INT from SPEC CPU2000 [Sta01]), floating point applications benchmark (SPEC FP from SPEC CPU2000), multimedia applications benchmark (MediaBench from UCLA [LPMS97]) and decision support applications benchmark (TPC-H from Transaction Processing Council [Tra04]).

The uniqueness test was performed by comparing the application characteristics. This was done by executing all benchmark applications in the same experimental setup of Figure 8.3 and then measuring and studying their corresponding (normalized) measures of interest as listed in Figure 8.5. Since manual comparison of characteristics is hard to do, the results were fed into the analysis framework and the statistical tools were applied. Specifically, clustering algorithms were applied to the characteristics. The goal
of this clustering is to find clusters of characteristics. That is, the assumption is that each benchmark is unique. This is obvious since intuitively characteristics of each benchmark is different. For instance, SPEC is heavy on computations. The goal of this clustering is to find if data mining applications show any similarity with other well known benchmarks (i.e. they both will belong to the same cluster in such a case). If that is the case, existing well-known optimizations from the corresponding field can be applied to data mining applications as well.

Surprisingly, the clustering results from statistical analysis proved the following. Figure 8.6 shows the result of applying clustering algorithms on the characteristics of data mining applications and other applications. Different clustering algorithms were applied to the characteristics (obtained by monitoring the performance counters during execution of applications) of data mining, SPEC, MediaBench and TPC-H applications. Figure 8.6 shows the scatter plot of the final cluster configuration when a Kohonen network based clustering method is applied to the characteristics. Clearly data mining algorithms fall into different clusters, with a few of them sharing characteristics that are seen in SPEC and TPC-H. MediaBench and TPC-H shows similar characteristics due to their streaming nature of data retrieval and computations. Note that data mining does not fall in this category even though it has a streaming nature of data access. This is due to the fact that in data mining, repeated computations (on streamed data) are performed in multiple phases, as against a single (long) phase of computations on the stream of data. Most data mining algorithms are unique in their characteristics, and hence, fall
Figure 8.6. Automated classification of data mining, SPEC INT, SPEC FP, MediaBench and TPC-H benchmark applications based on their characteristics. Statistical analysis tools with clustering algorithms were used for this classification. Data mining applications tend to form unique clusters.

into different clusters. This approach identifies data mining applications to be even more distinctly unique.

Now the next question that arises is what makes data mining applications distinct from other applications. Table 8.4 shows the core differences between NU-MineBench data mining applications and other well known application benchmarks: SPEC Integer, SPEC Floating Point, MediaBench and TPC-H. One key attribute that signifies the differences is the number of data references per instruction retired. For data mining applications, this rate is 1.103, whereas for other applications, it is significantly less. In Table 8.4, the number of bus accesses originating from the processor to the memory (per
Table 8.4. Comparison of data mining application with other benchmark applications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>SPECINT</th>
<th>SPECFP</th>
<th>MediaBench</th>
<th>TPC-H</th>
<th>Data Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data References</td>
<td>0.8071</td>
<td>0.5502</td>
<td>0.5676</td>
<td>0.4831</td>
<td>1.1032</td>
</tr>
<tr>
<td>Bus Accesses</td>
<td>0.0303</td>
<td>0.0344</td>
<td>0.0027</td>
<td>0.0104</td>
<td>0.0371</td>
</tr>
<tr>
<td>Instruction Fetches</td>
<td>1.6427</td>
<td>1.6325</td>
<td>1.0000</td>
<td>1.3193</td>
<td>2.7247</td>
</tr>
<tr>
<td>ALU Operations</td>
<td>0.2550</td>
<td>0.2920</td>
<td>0.2650</td>
<td>0.3000</td>
<td>0.3080</td>
</tr>
<tr>
<td>CPI</td>
<td>1.7583</td>
<td>1.7077</td>
<td>1.1185</td>
<td>1.3625</td>
<td>1.4724</td>
</tr>
</tbody>
</table>

† The numbers shown here for the parameters are values per instruction

instruction retired) verifies this fact as well. These results solidify the intuition that data mining is data-intensive by nature. Another important difference is the fraction of total instruction fetches to the instructions retired. This measure, the total instruction fetches, includes the non-cached instruction fetches, branch prediction fetches, and including those fetched as result of wrong prediction. Besides, the number of ALU (computation) operations per instruction retired is also surprisingly high for data mining applications, which indicates the extensive amount of computations performed in data mining applications. In fact, on average data mining applications have a higher rate of compute operations than the other applications. What makes the data mining applications unique is this combination of high data rates combined with high computation power requirements. The applications oscillate between such phases regularly, making the current processors and architectural optimizations mostly inadequate.

The data mining applications have a significantly low CPI (cycles per instruction retired), which indicates that the applications are able to be hosted well on the processor.
From the above data, it is clear that the applications are adequately hosted by the processor. Most of the bottlenecks are less. While the applications can be further optimized (using loop transformations, data fetch optimizations), it is clear that the application execution times would not improve by any traditional optimizations since the CPI is already at the best levels (comparing with traditional application CPIs). This suggests that radically new hardware and algorithmic design techniques are the keys to further application speedups.

In order to achieve this, one needs to revisit the facts about what makes data mining applications unique. Data mining applications are computationally intense. It is also data intensive, which is what makes it different from SPEC like applications. Data mining applications have lot of similarities with streaming applications since a consistently changing set of data is read for processing. But they are different from pure streaming applications by the fact that there are bursts of streaming data instead of data arriving at a consistent arrival rate. Figure 8.7 shows the data flow. There are a lot of core (may be repetitive) operations on these bursts of data. The next goal of this work is to extract such core operations, which are termed as “kernels”. Once these kernels are extracted, new operation-specific performance optimizations can then be enforced on these kernels to achieve massive application speedups.

8.4.2. Kernels of Data Mining Applications

Unlike the previous performance studies of data mining algorithms (based on their execution times and high-level algorithmic run times), this study focuses on analyzing
Figure 8.7. Multi-phased *data burst + kernel* operations seen in data mining

the applications as a workload for computing systems and performs extensive characteriza-
tion of the workload. This workload characterization proved that data mining applications have multi-phased operations that are both compute and data intensive by nature.

By performing an extensive characterization, this study was able to identify the core computation components of applications, which are termed as “kernels”. A functional kernel (say, function X) could be different from a kernel as seen by the processor (function X could translate to a set of add and multiply assembly operations). The current goal is to identify such relationships, then extract the common operations and also define abstract kernels. In other words, kernel is defined to be *an abstract representation of a set of operations that are performed frequently in an application*. Such extraction of kernels also helps in identifying how the kernel actually maps to the underlying hardware (basically, the processor, memory and other computational resources).
Once the kernels are identified, application speedups are guaranteed if the kernels are able to run faster with fewer bottlenecks. The goal is to enable scaling of these kernels using high-performance techniques, parallel and distributed methodologies, and also custom hardware-based techniques (a.k.a. new systems). Such scalable kernels can actually run on any system. That is, the granularity of parallelization is open. For instance, there is a huge shift in the recent technology paradigms - there exists multiple processors in a chip. The identified kernels should be able to smoothly run on such systems. The rest of this section focuses on identifying the core kernels of each application.

Table 8.5 presents the top three kernels of each application considered in this study. For each application, the name of the kernel and the percentage of the system time spent executing the kernel are presented. The last column shows the cumulative sum of the three kernels. It is evident the top three kernels constitute to most of the execution time (up to 99%); in some cases, the first kernel is the only significant one. For instance, in k-Means, the distance calculation forms 68% of the execution, whereas in HOP, the percentages are evenly distributed across three kernels. The kernels are now analyzed further.

In k-Means, each point is assigned to a cluster based on its distance. That is, the distance between every input point and each cluster is computed, and the point is finally assigned to the closest cluster (min of distances). The kernel distance is responsible for calculating the Euclidean distance and minDist calculates the minimum of the distances. The kernel clustering, assigns the actual cluster and recalculates the centers in each iteration (mean of points in a cluster). Since, Fuzzy k-Means is tightly connected to
Table 8.5. The top kernels of each application and their percentage contribution towards the total application execution time. The last column shows the total of the three kernel contributions.

<table>
<thead>
<tr>
<th>Application</th>
<th>Top 3 Kernels (%)</th>
<th>Sum %</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-Means</td>
<td>distance (68%) clustering (21%) minDist (10%)</td>
<td>99</td>
</tr>
<tr>
<td>Fuzzy k-Means</td>
<td>clustering (58%) distance (39%) fuzzySum (1%)</td>
<td>98</td>
</tr>
<tr>
<td>BIRCH</td>
<td>distance (54%) variance (22%) redistribution (10%)</td>
<td>86</td>
</tr>
<tr>
<td>HOP</td>
<td>density (39%) search (30%) gather (23%)</td>
<td>92</td>
</tr>
<tr>
<td>Naive Bayesian</td>
<td>probCal (49%) variance (38%) dataRead (10%)</td>
<td>97</td>
</tr>
<tr>
<td>ScalParC</td>
<td>classify (37%) giniCalc (36%) compare (24%)</td>
<td>97</td>
</tr>
<tr>
<td>Apriori</td>
<td>subset (58%) dataRead (14%) increment (8%)</td>
<td>80</td>
</tr>
<tr>
<td>Eclat</td>
<td>intersect (39%)addClass (23%) invertClass (10%)</td>
<td>71</td>
</tr>
<tr>
<td>SNP</td>
<td>compScore (68%) updateScore (20%) familyScore (2%)</td>
<td>90</td>
</tr>
<tr>
<td>GeneNet</td>
<td>condProb (55%) updateScore (31%) familyScore (9%)</td>
<td>95</td>
</tr>
<tr>
<td>SEMPHY</td>
<td>bestBrnchLen (59%) expectation (39%) lenOpt (1%)</td>
<td>99</td>
</tr>
<tr>
<td>Rsearch</td>
<td>covariance (90%) histogram (6%) dbRead (3%)</td>
<td>99</td>
</tr>
<tr>
<td>SVM-RFE</td>
<td>quotMatrix (57%) quadGrad (38%) quotUpdate (2%)</td>
<td>97</td>
</tr>
<tr>
<td>PLSA</td>
<td>pathGridAssgn (51%) fillGridCache (34%) backPathFind (14%)</td>
<td>99</td>
</tr>
</tbody>
</table>

k-Means, the *distance* calculation appears to be a prominent kernel for this application as well. In this case, the difference is that the *clustering* kernel is more time consuming than the *distance* calculation. This is because in fuzzy logic, the computations involved in performing the membership calculation (owing to multiple membership property) is more intense than the actual distance calculation. *fuzzySum* is used during the convergence process.

BIRCH is a hierarchical algorithm that actually uses distance (between a point and the cluster) as the “similarity” measure similar to k-Means. Hence, the *distance* kernel is common between k-Means and BIRCH. *variance* is used to calculate the inter-cluster distance variation, and if the cluster distances are beyond the threshold, the points are redistributed, which is handled by the *redistribute* kernel.
In the density based algorithm, HOP, the density generation phase forms the main computation component. density, search, and gather are used during the density generation phase. search searches the neighboring particles of each particle and then generates density for the particle. The actual density generation is done using density (using the distance and mass metrics [EH98]). gather gathers the density of a neighborhood of particles. It is used to compare densities of particles while forming the clusters.

Since Bayesian classifier is based on a statistical method, the Bayes’ theorem, the two prominent kernels include the probability calculation (probCalc) and variance estimation (variance). Every input object (in the learning data) is read (using dataRead) and the probability is recalculated by including that object. Note that the learning part of the classifier is the most crucial operation. Once the classifier (predictor) model is built using the learning data, all other unclassified objects are easily classified by using the probability and variance models. ScalParC is a more sophisticated classifier model that uses decision trees to classify data. Every input data is read and classified in the tree node using its gini index as the split criteria. This forms the core operation of the classify kernel. The actual gini index computation (for each node) is done in giniCalc. Data items are compared during the classification phase (in the compare kernel).

Apriori, the association rule algorithm, forms rules by identifying frequently occurring pattern of item combinations (frequent itemset). All nonempty subsets of a given frequent itemset should also be frequent in order to be considered. This operation is performed in the subset kernel. That is, at the kth iteration (for k > 1), this kernel computes the frequent (k+1)th-itemset candidates based on the frequent k-itemsets. Then
the database is scanned to find the complete set of frequent (k+1)th-itemsets (done by `dataRead` kernel). Simultaneously, a `count` operation is performed to keep track of the number of items in the database that has this (k+1)th candidate item. On the other hand, ECLAT works differently by first generating the potential maximal frequent itemsets called equivalence classes (done in `addClass` kernel). Then it uses these classes to identify potential candidates and determine the presence of these candidates in the database by performing the subset operation (`invertClass` kernel). Since a vertical database format is used, the support of any k-itemset is determined by simply intersecting the first two (k-1)-length subsets that share a common prefix (`intersect` kernel).

SNP uses the hill-climbing search algorithm. The core part of this algorithm involves calculation of `scores`. The algorithm chooses a point as an initial start point. Then it identifies the neighbors of this point and computes the scores for all neighbors. Note that the neighbors are represented using a Bayesian network structure, which is basically a DAG tree. The neighbor that has the highest score is used as the next initial point. This procedure is done recursively until all no neighbor has a score higher than the current point. The `compScore` kernel computes the scores, which involves conditional probabilities (Bayes theory of $P(A|B)$) [DCD+05]. Once the scores are computed, `updateScore` updates the scores of the neighbors. The kernel `familyScore` is in charge of identifying the highest scoring neighbor and designating it as the next iteration start point.
Since GeneNet is closely associated with SNP (both use the Bayesian network structures and hill-climbing search algorithm), the kernels are very identical for both applications. The only difference is that the input data used is different, in the case of GeneNet, it is the microarray data. This introduces more variables and conditions. Hence, the top kernel is different for GeneNet. \textit{condProb} calculates the Conditional Probability Distribution (CPD) of the variables. It again involves the Bayesian approach, but here, the probability computation is more intensive because of the data characteristics (basically more dimensions). All other kernels are alike.

SEMPHY uses the Structural Expectation Maximization approach to identify the core gene relationships. The application has two core kernels which constitute 99\% of the execution times. The first phase of the algorithm involves expectation estimation (probabilistic). The kernel \textit{expectation} performs this part. The gene data is represented using phylogenetic trees. The expectation is computed by traversing the tree iteratively based on the edge length. Once the expectations are estimated, the second phase of the algorithm involves traversing the tree to find the best possible branch lengths. This is covered by the \textit{bestBranchLen} kernel. This phases uses the expectation values calculated by \textit{expectation} kernel. In this phase, the expectations are recalculated for the edge nodes based on the best “edge” length path.

RSEARCH uses Stochastic Context-Free Grammar to store and also search RNA sequences. There are three steps in the RSEARCH application. The first phase involves building a covariance model based on the RNA sequences. SCFG grammar is applied to store the sequences (as they are read from the input) in a tree format. Covariance
is simultaneously calculated. This forms the core part of the covariance kernel. Then this model is used as a substring to scan the database and identify matching sequences. As the database is scanned, histograms are built to keep track of the alignments. The kernels histogram and dbRead perform these actions. The alignments and the homolog results are then projected and displayed.

SVM-RFE uses Support Vector Machines to refine and identify a set of features that define and represent a gene of interest. The problem is projected as a global quadratic programming model that tries to minimize the variable distribution (based on weight) in the input training data set. The features within the data form the variables. Thus, it becomes a feature elimination and refinement problem. The problem is solved by using an induction approach. Decomposition method (based on LibSVM [DCD+05]) is applied to break this large QP problem into a series of small QP problems, which are then inductively solved. The top kernel of the application is quotMatrix, which involves definition of features in the form of a quotient matrix (with weights assigned). quadGrad, the second kernel, computes the gradient for all training samples. This gradient is used for solving the quadratic subproblem. Once this subproblem is solved, the gradients are used to update the original matrix (quotUpdate kernel). Features are then updated and history data is eliminated from the training set. The training set is thus updated and made ready for the next iteration. This iteration continues until the optimal weight (feature set) is achieved.

In PLSA, dynamic programming based on the Smith and Waterman algorithm is used to identify similarities between gene sequences. The goal of this algorithm is
to find the longest matching (common) substring between two sequences (gene, SNP, DNA). The version used in this work is based on the concept of a grid cache. Grid cache is basically a subset (or a block) of the original similarity matrix that is used for finding similar sequences. The usage of grid caches enables the problem to be broken down into simpler subproblems. These subproblems are then solved using a dynamic programming approach to find a final global solution. The kernel \textit{fillGridCache} is in charge of identifying grid caches, which are basically blocks of the input matrix. These grid caches are then distributed to different nodes (virtual nodes or even physical processor nodes). \textit{pathGridAssgn} takes care of this grid cache distribution. Then, each node works on its individual grid cache, solving the smaller dynamic program. Solutions are combined through grid cache boundaries, which are basically shared portions of the matrix. Once the subproblems are solved, the final global alignment is done by combining these subproblem alignments to an optimal path. This is termed as the “backward phase” of the algorithm, which is basically the kernel \textit{backPathFind}.

It was found that the underlying computations and the kernels always remain the same, independent of the data size variations. Since these kernels contribute significantly (up to 99%) to the execution times, they contribute to significant portion of the performance bottlenecks as well. The rest of this work focuses on optimizing the various architectural performance aspects of just these kernels.
8.4.3. Scalability of Kernels

Scalability is deemed to be an important (and unavoidable) requirement in future systems. Collaterally, the goal of this work is also to study the scalability of data mining applications. The kernel of each data mining application was elaborated in the previous section. In this section, the kernels are extracted and scaled further by extending traditional data parallelism techniques to these kernels. The kernels and hence the applications were parallelized using well-known techniques available in the literature. The techniques and the algorithms for parallelization are discussed in [LPL+04, CDD+05, DCD+05]. This section focuses purely on the scalability of the overall applications.
Figure 8.9. Scalability of data mining applications with datasets. This graph shows the speedups of each data mining application for different data sizes. Applications are executed on multiple processors.
Figure 8.8 shows the scalability of applications with the processors. Applications were executed on an Intel Xeon based 8-way Shared Memory Parallel (SMP) machine with a 4GB shared memory and 1MB L2 cache per processor. The scalability of the parallel applications is measured by executing each of them on 1, 4 and 8 processors. The speedups shown in the graph are based on the execution times obtained when a large sized dataset is used. For some applications, the margin of speedups begin to reduce. In apriori, this is because of inefficient data structure accesses (atomic accesses on the hash tree data structures). For SVM-RFE, the problem arises due to unnecessary communication problems and locking of memory structures. This redundant locking is done to ensure the code works on distributed and shared memory machines. If the locks are removed (using shared memory programming), the program and its kernels scale linearly.

Figure 8.9 shows the scalability of applications with datasets. The figure proves that the applications scale well with larger datasets (shown here for the 8 processor case). As the data sizes increase, the margin of speedups also improve. The above two factors clearly prove that the kernels within the application are very scalable. This opens up abundant venues for performance improvement through parallel, distributed and other high-performance techniques.

8.5. Summary

Data mining has become one of the most essential tools for various businesses as well as researchers in diverse fields. The surge in the operational speed of computing
systems, and also the emergence of compact, low-cost, high-performance parallel and distributed systems have provided abundant venues for improving the performance of data mining algorithms. However, in recent years, there has also been a tremendous increase in the size of data that is collected and also the complexity of data mining algorithms themselves. The rate of this growth exceeds the rate of performance improvements in computing systems, thus widening the performance gap between data mining systems and algorithms. In this work, the goal is to narrow this gap by enabling designers to build systems that are tuned in accordance with the requirements and developments of data mining algorithms. This is achieved by performing a detailed characterization of a set of representative data mining programs from both the hardware and software perspectives.

Several widely-used data mining algorithms from multiple categories were studied and then, a benchmark suite was designed. Named NU-MineBench, this benchmarking suite contains representative data mining applications. NU-MineBench suite includes applications from diverse applications ranging from cosmology, grocery stores to bioinformatics. The applications in NU-MineBench were evaluated using real systems and simulators. Automated analysis framework was used for this purpose. The information derived from the analysis and evaluation results aid designers of future systems as well as programmers of new data mining algorithms to achieve better system and algorithmic performance. Moreover, current trends indicate that systems are moving towards the deployment of multiple processing cores as a way to increase their total computation power. Hence, scalability is an inevitable factor expected in future algorithms.
and systems. With this in mind, this work also addresses the scalability characteristics of data mining applications. The evaluation was performed from many aspects, which span from the software level to the hardware level. The results clearly indicate that data mining applications are unique. Also, data mining applications have distinctive kernels, which opens up abundant venues for design optimizations. Algorithmic and architectural optimizations can be performed on the multi-phased data and compute stages of these applications.
CHAPTER 9

Architectural Acceleration of Data Mining Kernels

As mentioned in the previous chapter, data mining is emerging as a powerful tool for automated discovery of useful information. The extensive characterization of data mining applications proved that these applications perform non-traditional multi-phased operations on massive amounts of data. Such operations are typically not seen in other application domains. Current systems lack the complete capabilities to perform such operations. Besides, the data handled by such applications is also growing at a rapid pace. Future systems should be redesigned to efficiently accommodate these unconventional computations in order to ensure a quick and precise result throughput.

The ultimate goal of this work is to enable a smooth integration of data intensive applications into computer systems. This is achieved by driving new technologies and by using existing technologies in a non-traditional fashion. For instance, this work takes a new approach of using reconfigurable logic to accelerate core execution components. On the other hand, existing parallel, distributed and other high-performance techniques are also extended to maximize the availability and utilization of data-intensive systems. Figure 9.1 shows the overall goal of this work. This figures compares the non-traditional method of designing high speed data analysis systems with existing systems and tools used for data analysis. It is clear that as data the amount of data increases, the data
Figure 9.1. Goal of this work is to design high performance data mining system. This figure compares this goal with other existing designs and methodologies in high speed data analysis.

handling becomes more complex. This in turn gives rise to the need for customized data analysis systems that are fast and reliable. The goal of this part of the thesis is to design customized systems that provide high performance by accelerating the inherent computations seen in large volume data analysis.

The above fact serves as the core motivation to propose a new accelerator-based system that aims to tremendously speed up data mining applications. For this, the inherent characteristics (computations) of data mining applications, termed as *kernels*, are first extracted. Then, these individual kernels (hence, the entire application) are speeded up using a reconfigurable accelerator hardware. The goal of this work is to develop
a data mining system that tremendously improves the execution times of data mining applications through massive large-scale speedups, that are not achieved by traditional techniques.

The rest of the chapter is organized as follows. Section 9.1 covers the details of the applications considered for this study. Section 9.2 presents the characteristics of these applications. The overall program flow and the core kernels of each application are presented in that section. The accelerator-based data mining system is introduced in Section 9.3. In Section 9.4 the accelerator is evaluated and the performance of the accelerator is discussed in detail.

9.1. Data Mining Applications Mix

The data mining applications were studied in detail in Chapter 8. A classification of data mining applications was shown in Figure 8.2 of Chapter 8. In this chapter, the goal is to use the unique nature of data mining applications as a major driving force to customize computing systems in order to accommodate data mining applications. A subset of application domains from Chapter 8 is used for this study. A set of four clustering, two classification and one association rule based algorithms is presented here. Table 9.1 shows the selected algorithms and their description.

These algorithms were explained in detail in Chapter 8. In the succeeding sections, these applications are analyzed through an integrated hardware-software approach to identify their inherent characteristics and the bottlenecks. These characteristics are then
Table 9.1. Algorithms used in this study and their descriptions

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-Means</td>
<td>Clustering</td>
<td>Mean based partitioning method</td>
</tr>
<tr>
<td>Fuzzy k-Means</td>
<td>Clustering</td>
<td>Fuzzy-logic based partitioning method</td>
</tr>
<tr>
<td>BIRCH</td>
<td>Clustering</td>
<td>Hierarchical method</td>
</tr>
<tr>
<td>HOP</td>
<td>Clustering</td>
<td>Density based method</td>
</tr>
<tr>
<td>Naive Bayesian</td>
<td>Classification</td>
<td>Statistical classifier</td>
</tr>
<tr>
<td>ScalParC</td>
<td>Classification</td>
<td>Decision tree classifier</td>
</tr>
<tr>
<td>Apriori</td>
<td>ARM</td>
<td>Mining based on apriori property</td>
</tr>
<tr>
<td>Eclat</td>
<td>ARM</td>
<td>Equivalence class based mining</td>
</tr>
</tbody>
</table>

used as a motivation to redesign and customize computing systems to accommodate data mining applications.

9.2. Workload Characteristics

In this section, the various unique characteristics of data mining applications presented in Chapter 8 are revisited. To be specific, the kernels presented in Chapter 8 would be revisited in this section. The goal of this work is to extract such well-defined kernels that carry a meaning at the algorithmic, functional and architectural levels. Since the final goal is to build an accelerator, isolation of such kernels is absolutely essential to alleviate the real bottlenecks. Once the kernels are identified, application speedups are guaranteed if the kernels are able to run faster with fewer bottlenecks. This forms the goal of this chapter.

Extensive profiling of the data mining applications of Table 9.1 based on functional and architectural characterization using well known profiling tools (setup already described in Section 8.3) revealed the following.
9.2.1. Kernel Extraction

Table 9.2 presents the top three kernels of each application. For each application, the name of the kernel and the percentage of the system time spent executing the kernel are presented. The last column shows the cumulative sum of the three kernels. It is evident the top three kernels constitute to most of the execution time (up to 99%); in some cases, the first kernel is the only significant one. For instance, in k-Means, the distance calculation forms 68% of the execution, whereas in HOP, the percentages are evenly distributed across three kernels.

In k-Means, each point is assigned to a cluster based on its distance. The distance between every input point and each cluster is computed, and the point is finally assigned to the closest cluster (min of distances). The kernel *distance* is responsible for calculating the Euclidean distance and *minDist* calculates the minimum of the distances. The kernel *clustering*, assigns the actual cluster and recalculates the centers in each iteration (mean of points in a cluster). Since, Fuzzy k-Means is tightly connected to k-Means, the *distance* calculation appears to be a prominent kernel for this application as well. In this case, the difference is that the *clustering* kernel is more time consuming than the *distance* calculation. This is because in fuzzy logic, the computations involved in performing the membership calculation (owing to multiple membership property) is more intense than the actual distance calculation. *fuzzySum* is used during the convergence process.
BIRCH is a hierarchical algorithm that actually uses distance (between a point and the cluster) as the “similarity” measure similar to k-Means. Hence, the distance kernel is common between k-Means and BIRCH. variance is used to calculate the inter-cluster distance variation. If the cluster distances are beyond a defined threshold, the points are redistributed, which is handled by the redistribute kernel.

In the density based algorithm, HOP, the density generation phase forms the main computation component. density, search, and gather are used during the density generation phase. search searches the neighboring particles of each particle and then generates density for the particle. The actual density generation is done using density (using the distance and mass metrics [EH98]). gather gathers the density of a neighborhood of particles. It is used to compare densities of particles while forming the clusters.

Since Bayesian classifier is based on a statistical method, the Bayes’ theorem, its two prominent kernels include the probability calculation (probCalc) and variance estimation (variance). Every input object in the learning data is read (using dataRead) and the probability is recalculated by including that object. Note that the learning part of the classifier is the most crucial operation. Once the classifier (predictor) model is built using the learning data, all other unclassified objects are easily classified by using the probability and variance models. ScalParC is a more sophisticated classifier model that uses decision trees to classify data. Every input data is read and classified in the tree node using its gini index as the split criteria. This forms the core operation of the classify kernel. The actual gini index computation for each node is done in giniCalc. Data items are compared during the classification phase, in the compare kernel.
Apriori, the association rule algorithm, forms rules by identifying frequently occurring pattern of item combinations (frequent itemset). All nonempty subsets of a given frequent itemset should also be frequent in order to be considered. This check is performed by the subset kernel. That is, at the kth iteration (for \( k > 1 \)), this kernel computes the frequent (k+1)th-itemset candidates based on the frequent k-itemsets. Then the database is scanned to find the complete set of frequent (k+1)th-itemsets (done by dataRead kernel). Simultaneously, a count operation is performed to keep track of the number of items in the database that has this (k+1)th candidate item. On the other hand, ECLAT works differently by first generating the potential maximal frequent itemsets called equivalence classes (done in addClass kernel). Then it uses these classes to identify potential candidates and determine the presence of these candidates in the database by performing the subset operation (invertClass kernel). Since a vertical database format is used, the presence of candidates is determined by simply intersecting the first two (k-1)-length subsets that share a common prefix (intersect kernel).

It is found that the underlying computations and the kernels always remain the same, independent of the data size and system variations. Since these kernels contribute significantly (up to 99%) to the execution times, the rest of the chapter will focus on just these kernels. The following section shows the various architectural performance aspects of the kernels.
Table 9.2. The top kernels of each application and their percentage contribution towards the total application execution time. The last column shows the total of the three kernel contributions.

<table>
<thead>
<tr>
<th>Application</th>
<th>Top 3 Kernels (%)</th>
<th>Sum %</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-Means</td>
<td>distance (68%)</td>
<td>99</td>
</tr>
<tr>
<td>Fuzzy k-Means</td>
<td>clustering (58%)</td>
<td>98</td>
</tr>
<tr>
<td>BIRCH</td>
<td>distance (54%)</td>
<td>86</td>
</tr>
<tr>
<td>HOP</td>
<td>density (39%)</td>
<td>92</td>
</tr>
<tr>
<td>Naive Bayesian</td>
<td>probCal (49%)</td>
<td>97</td>
</tr>
<tr>
<td>ScalParC</td>
<td>classify (37%)</td>
<td>97</td>
</tr>
<tr>
<td>Apriori</td>
<td>subset (58%)</td>
<td>80</td>
</tr>
<tr>
<td>Eclat</td>
<td>intersect (39%)</td>
<td>71</td>
</tr>
</tbody>
</table>

Table 9.3. Architectural performance characteristics of the top three kernels (totaled up) of the applications.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>k-Means</th>
<th>Fuzzy</th>
<th>BIRCH</th>
<th>HOP</th>
<th>Bayesian</th>
<th>ScalParC</th>
<th>Apriori</th>
<th>Eclat</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of Execution Cycles</td>
<td>99</td>
<td>98</td>
<td>86</td>
<td>92</td>
<td>97</td>
<td>97</td>
<td>80</td>
<td>71</td>
</tr>
<tr>
<td>% of Resource Stalls</td>
<td>99</td>
<td>99</td>
<td>80</td>
<td>86</td>
<td>73</td>
<td>99</td>
<td>92</td>
<td>69</td>
</tr>
<tr>
<td>L1 Miss Rate</td>
<td>0.003</td>
<td>0.077</td>
<td>0.022</td>
<td>0.079</td>
<td>0.008</td>
<td>0.763</td>
<td>0.448</td>
<td>0.449</td>
</tr>
<tr>
<td>L2 Miss Rate</td>
<td>0.036</td>
<td>0.887</td>
<td>0.006</td>
<td>0.091</td>
<td>0.001</td>
<td>0.627</td>
<td>0.161</td>
<td>0.342</td>
</tr>
<tr>
<td>L1 Misses Per Instr</td>
<td>0.002</td>
<td>0.034</td>
<td>0.013</td>
<td>0.032</td>
<td>0.004</td>
<td>0.639</td>
<td>0.714</td>
<td>0.523</td>
</tr>
<tr>
<td>L2 Misses Per Instr</td>
<td>0.0005</td>
<td>0.0005</td>
<td>0.00004</td>
<td>0.0015</td>
<td>0.000001</td>
<td>0.0061</td>
<td>0.0049</td>
<td>0.0146</td>
</tr>
<tr>
<td>Data References Per Instr</td>
<td>0.565</td>
<td>0.438</td>
<td>0.577</td>
<td>1.170</td>
<td>0.501</td>
<td>0.838</td>
<td>1.594</td>
<td>0.859</td>
</tr>
<tr>
<td>FP Ops Per Instr</td>
<td>0.112</td>
<td>0.036</td>
<td>0.105</td>
<td>0.904</td>
<td>0.025</td>
<td>0.028</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>CPI</td>
<td>1.57</td>
<td>1.00</td>
<td>1.26</td>
<td>1.55</td>
<td>0.92</td>
<td>1.90</td>
<td>1.77</td>
<td>1.57</td>
</tr>
</tbody>
</table>

9.2.2. Architecture Performance of Kernels

Table 9.3 shows the core performance components of the top three kernels of each application. The values shown are the cumulative sum for the top three kernels of each application (presented in Table 9.2). These kernels contribute significantly to the resource stalls in the processor (some kernels contribute up to 99%). The L2 cache miss rates are high for certain applications due to the streaming nature of data retrieval, which does not provide opportunities for data reuse. For instance, in apriori, transactions are
read once in each iteration but never reused. The minimum data references per instruction is 0.438 (max 1.59), which implies memory is fairly used. The number of floating point operations in this application is significantly high, which indicates the extensive amount of (repetitive) computations performed on the fetched data. Note that ECLAT and apriori work on pure integer data. The kernels have a significantly low CPI, which indicates that the applications are hosted well by the processor. Most of the bottlenecks are less. While these applications can be further optimized (using loop transformations, data fetch and locality optimizations), massive application speedups are hard to achieve using traditional optimizations since the CPI is already at the best levels when considering the traditional workload CPIs. This suggests that hardware redesign and optimization is the key to substantial application speedups.

9.3. Architectural Exploration

The results of the previous section highlight the fact that data mining applications have hit the performance wall of existing computing systems. One way to further reduce the execution times of these applications is to optimize the existing algorithms furthermore or extend them to high-performance systems like parallel and distributed systems, which has already been explored by researchers in the past. On the hand, another way to achieve application speedups is to optimize architectures and systems to the application under consideration. For instance, graphics accelerators, network processors and gaming machines are highly optimized systems for their respective applications. Further speedups can be achieved in data mining applications if these applications are run
on customized systems with high-speed data mining engines in their architectures. The above fact is used as a motivation to present an accelerator-based system that speeds up the kernels of data mining applications. This is under the belief that when the kernels are accelerated, the applications would automatically speedup (since kernels form a significant portion of the execution times).

### 9.3.1. Accelerator Design

Figure 9.2(a) illustrates the generic design of the proposed data mining systems architecture. There is a reconfigurable data mining accelerator, which exists as a coprocessor along with the general purpose processor. The decision to go with a coprocessor is because the kernels (that are accelerated by the accelerator) tend to run for a considerable amount of time, allowing the code to be offloaded to the coprocessor (ensuring good speedups that is worth the offload overhead) and thus, allowing the general purpose processor take up other tasks without getting blocked. The accelerator interfaces with the main processor using its own interface, which is also connected to the global memory subsystem.

Figure 9.2(b) shows the details of the accelerator of Figure 9.2(a). When an application is loaded, the accelerator has to be loaded (configured) with the kernels of the application. This will allow the general purpose processor to offload the kernel execution to the coprocessor. During the application load, the general purpose processor invokes the configuration controller, which in turn initiates a transfer of the control logic
Figure 9.2. (a) The data mining systems architecture, and (b) the design of the reconfigurable data mining accelerator in the data mining systems architecture.

(set of kernel configurations) from the configuration memory to the reconfigurable execution unit. The configuration memory is preprogrammed to hold all necessary program kernels for each application. For instance, for the k-Means application, there would be two logics (for distance and minDist kernels) stored in this memory. The corresponding logic(s) is (are) loaded to the reconfigurable execution processor. The reconfigurable processor is thus configured for the application, and made ready to perform the core kernel operations. Once the logics have been loaded, the execution unit hardly needs to be reprogrammed. This is due to the fact that the kernels remain the same for a given application. Only during an application change, the execution unit needs to be reprogrammed. During program execution, the general purpose processor on encountering
the beginning of a kernel will offload the kernel execution to the accelerator and continue with other tasks. The data required for kernel execution would also be loaded to the accelerator using the input/output buffer, which is also connected to the memory (just to ensure that the accelerator works independently – in case it needs data from the memory during execution). The accelerator then exclusively executes the kernels. There is a “scratch” memory attached to the accelerator to store intermediate values if needed.

The accelerator is generic enough to be used for any data mining application. All it would require is to extract the core kernels of the application, and to load the corresponding kernel logics for that application into the configuration memory. For each application, kernels can be extracted by studies as shown in Section 9.2.1 or even simple algorithmic analysis (which might not be accurate). Once the kernels are identified, corresponding hardware logics can be built by examining the underlying computations within each kernel. The next subsection illustrates how this was achieved for the application kernels under study.

9.3.2. Defining Hardware Logics for Kernels

The kernels of the applications were presented in Section 9.2.1. Within each kernel, there lies a set of repetitive operations (computations). For instance, in k-Means, distance is the kernel and the actual point-point Euclidean distance calculation is the underlying computation. Since the final goal is to build a kernel(s) accelerator, isolation of such basic operations at the computational granularity is absolutely essential to efficiently offload the kernels to the accelerator.
In k-Means application, the distance kernel computes the distance between a point and a cluster (which is another point). Each point is represented by its position in the multi-dimensional space (N dimensions). The Euclidean distance metric for M points is defined as:

\[ D_i = \sum_{j=1}^{N} (a_j - b_j)^2 \forall i = 1, \ldots, M \]

where \( a_j \) and \( b_j \) are the \( j^{th} \) coordinates of point i and a cluster. There are k clusters in the space at any given instance.

Following this operation of finding distance is the operation of assigning the point to its closest cluster, done using minDist kernel. The closest cluster is obtained by finding the minimum of the distances (\( D_i \)’s).

\[ \text{closestCluster} = \text{index}(\text{MIN}(D_i)) \forall i = 1, \ldots, M \]

The above two operations (Equations 9.1 and 9.2) are now translated into their equivalent hardware logics. Figure 9.3 shows the translated hardware logic. The top part of Figure 9.3 shows the distance calculation (Equation 9.1) and the bottom part shows the minimum calculation (Equation 9.2). The distance calculator involves a level of N subtractors (adder logic) followed by a set of N multipliers. The third level is of depth \( \log(N) \), and involves N-1 cumulative adders. The minimum computation involves a combination (with multi-level depth) of multiplexer and comparator logics to compare and send the actual data item to the final level (minimum neighbor). It should be noted that
the levels are tightly pipelined, allowing the results to be produced every cycle (best case). For instance, in distance calculation, only the first set of data incurs a three level hierarchy delay (of one level of N subtractors, one level of N multipliers and another level of N-1 cumulative adders) before actually producing the results, after which the resultant data is produced every cycle for a streaming input set of points. These logics are preprogrammed and stored in the configuration memory for use during application execution.

Fuzzy k-Means also has the distance and closest cluster computations, which now form the configuration logics for this application. BIRCH also uses distance as a measure to form clusters. Hence, the distance logic of Figure 9.3 is applicable for BIRCH as well. But besides distance, BIRCH also has an additional kernel, variance. The variance of inter-cluster distances is computed as follows (based on [ZRL96]):

\[
\text{clusterVariance} = \sum_{k=1}^{N_1+N_2} (\overrightarrow{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \overrightarrow{X}_l}{N_1+N_2})^2 - \sum_{j=1}^{N_1} (\overrightarrow{X}_j - \frac{\sum_{l=1}^{N_1+N_2} \overrightarrow{X}_l}{N_1})^2 - \sum_{i=1}^{N_1+N_2} (\overrightarrow{X}_i - \frac{\sum_{l=1}^{N_1+N_2} \overrightarrow{X}_l}{N_2})^2
\]

The above equation basically follows the traditional statistical definition:

(9.3) \[ \sigma^2 = \frac{\sum (X-\mu)^2}{N} \]

Equation 9.3 can now be realized by using the same logic defined for distance in Figure 9.3. This is due to the following reason. The operation \( \sum (X - \mu)^2 \) is the same as
\[ \sum_{j=1}^{M} (a_j - b_j)^2 \] since it involves a level of subtraction, followed by multiplication, and finally a cumulative sum (as shown in Figure 9.3). Thus, BIRCH can accommodate the same logic for variance calculation as well.

In both the search and gather kernels of HOP, multi-dimensional distance calculation is the core computation since density is computed using the mass and relative distance of particles. Neighboring particles are searched for (and gathered) using the distance measure, and then the “nearest” densest neighbor is identified. Thus, the logics of Figure 9.3 is directly applicable for HOP. The final kernel that needs to be accelerated for HOP is the actual density calculation part. Density for a particle is defined as (based on adaptive smoothing kernel algorithm [EH98]):

\[
Den_i = \sum_{j=0}^{N_{dens}} smooth_j \cdot mass_j \forall i = 1, \ldots, N_{particles}
\]

where \( smooth_j \) is the weighted (smoothed) distance of the particle \( j \) with respect to its \( N_{dens} \) neighbors and \( mass_j \) is the mass of particle \( j \).

The density equation is translated to the hardware logic on the accelerator as shown in Figure 9.4. As before, the design is fully pipelined, which implies the first result incurs the delay of the two level hierarchy, after which results are produced every cycle.

The defined logics can be easily extended to other applications. For instance, Equation 9.3 (and its logic) is applicable for the Naive Bayesian classifier as well, since variance is one of its compute-intensive kernel. Adding new kernels is also an easy task since only the new logics need to be loaded to the configuration memory. Once a new
Figure 9.3. Hardware logics for the *distance* calculation and the *minimum* computation kernels. The logic shown on the top part of the figure calculates the distance $D_i$ and the one on the bottom of the figure calculates the minimum of $D_i$'s supplied to it.
application and its associated logic is loaded to the configuration memory, the general purpose processor can appropriately load the necessary logics from the configuration memory during the actual execution. The following section discusses the implementation details of the above accelerators and also presents the results of the evaluation.

9.4. Evaluation of Acceleration

Figure 9.5 shows the applications under study and the accelerated kernels (looks similar to the contents of the configuration memory). The following section presents the experimental set up, which is followed by the performance results of the accelerator after implementing the logics of Figure 9.5.
<table>
<thead>
<tr>
<th>Application</th>
<th>Kernels (logics needed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-Means</td>
<td>distance, minimum</td>
</tr>
<tr>
<td>Fuzzy k-Means</td>
<td>distance, minimum</td>
</tr>
<tr>
<td>BIRCH</td>
<td>distance, variance</td>
</tr>
<tr>
<td>HOP</td>
<td>search, gather, density</td>
</tr>
<tr>
<td>Naive Bayesian</td>
<td>variance</td>
</tr>
</tbody>
</table>

Figure 9.5. Applications for acceleration and their logics

9.4.1. Experimental Setup

For profiling the functions within the applications, and for measuring the breakdown of execution times, Intel VTune Performance Analyzer 7.2 (hardware/software event tracking) is used alongside with GNU gprof. The applications are hosted and evaluated on an Intel Xeon machine (700 MHz) running Linux operating system, with a 4GB shared memory, 1024 KB L2 cache and 16KB L1 i- and d-caches. The measure used to judge the performance is the number of processor cycles spent in executing the application (values obtained from VTune). The accelerator is attached to the overall processor model (setup) as an architecture-level cycle accurate simulator. The accelerator has the ability to perform at the same clock speed as the regular processor. The core processor simply offloads the kernel computations to the accelerator. In order to enable this, markers (place holders) were attached in the actual application code to indicate the entry points of the kernel computations. The processor uses this as a clue to offload the kernel and the related input data to the accelerator; the same markers are used as a clue by the accelerator to begin the kernel computations. Since the accelerator serves as a coprocessor to the main processor (and also exists independently), the new total cycles
spent executing the application is the cumulative sum of the cycles spent by the application in the main processor and by the kernels in the accelerator. It should be noted that the cycles spent in the main processor include the cycles taken up by the non-kernels and also the cycles that arise as a result of the kernel offload overheads. During the actual evaluation, this new total cycles (spent by the application in the new accelerator driven model) is compared against the original total cycles (base model without the accelerator) to identify the total savings obtained by using the accelerator.

For each application, input data were generated in 3 different sizes: Small (S), Medium (M), and Large (L). Table 9.4 shows the S, M, L datasets, and their characteristics for each application. A section of the real image database distributed by Corel Corporation is used for k-Means and Fuzzy k-Means. Table 9.4 shows the number of images (and the dimension of each image feature vector). For HOP and BIRCH, three sets of real data from a cosmology application, ENZO [NSLD99] were used. The table shows the number of particles (each of 5 dimensions) in each case. For Bayesian, three synthetic datasets were generated by the IBM Quest data generator (http://www.almaden.ibm.com/software/quest/Resources/index.shtml). The notation Fx-Ay-DzK denotes a dataset with Function x, Attribute size y, and Data comprising of z*1000 records. The datasets generated are of sizes 27MB, 54MB and 100MB for S, M and L respectively. In the succeeding sections, the performance improvement seen in each kernel of the application is first discussed, and then the application-wide performance based on the consolidated total application execution cycles is then presented.
Table 9.4. Datasets used in the experiments. S refers to Small, M to Medium and L to Large datasets.

<table>
<thead>
<tr>
<th>Application</th>
<th>S</th>
<th>M</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{kMeans}, \textit{Fuzzy kMeans}</td>
<td>1000 (8)</td>
<td>10000 (18)</td>
<td>180000 (18)</td>
</tr>
<tr>
<td>BIRCH, HOP</td>
<td>61440</td>
<td>491520</td>
<td>3932160</td>
</tr>
<tr>
<td>Bayesian</td>
<td>F26-A32-D125K</td>
<td>F26-A32-D250K</td>
<td>F26-A64-D250K</td>
</tr>
</tbody>
</table>

9.4.2. Kernel Performance

This section presents the accelerator performance for each logic of Figure 9.5. Figure 9.6 shows the speedups of each application. The graph shows the speedups for every application kernel logic and for each of the 3 different data sets. As the data size increases, the speedups also improve. This is mainly due to the fact that the pipeline is more effectively used when data sizes are significantly large. That is, as the data is streaming, resultant data is produced every cycle, with the exception of the first result which takes a delay proportional to the depth of the pipeline. Thus, the results clearly support the fact that a coprocessor works better than a conventional general purpose processor when more data is offloaded to it. The tremendous speedups prove that the idea of extracting core kernels (compute intense, but not fully supported by regular processors) for acceleration is effective. It should be noted that the speedups presented are for the real data set, which is 5 to 18 dimensional in size, thus, requiring minimal hardware resources (adders/multipliers in each level).

In order to understand the design space and the sensitivity of the proposed schemes, the parameter N (number of hardware resources in one level) of the accelerator was varied. This is basically the number of parallel compute blocks (horizontal parallelism)
Figure 9.6. Performance of the kernels after deploying the accelerator. Each application and its kernel is shown. The graphs show the speedup in execution cycles of the kernels of each application obtained after using the accelerator as against executing these kernels purely on the main processor.

that is offered in each level of the accelerator. For instance, current transistor trends might not allow more than 20 adders, 20 multipliers and 20 cumulative adders to be on an accelerator chip (considering a “realizable” area design). Figure 9.7 shows the scaling
Figure 9.7. Speedups for increasing values of N (horizontal parallelism) for top three kernels (combined) of the applications, executed with the L dataset.

of the three kernels (cumulative) with respect to the different sizes of N. The results clearly indicate that the application scales with the number of computation blocks, that is, the application exploits horizontal parallelism to the extent of maximum available hardware. This study indicates that the parameter N can be proportionately increased with newer designs (since more transistors are predicted to be available on a single chip in future). The application would scale with increasing N.

9.4.3. Application Performance

Figure 9.8 shows the speedups obtained by the applications when the accelerator is used. The graph covers all three data sets. It is clear that the applications speed up
tremendously due to the fact that the coprocessor effectively accelerates the core kernels of the application. It should be noted that such speedups are obtained with minimal amount of resources (equal to the number of dimensions in the real data set). With more resources, the kernel speedups of Figure 9.7 translate to application speedups. Note that when the kernels are accelerated, the non-kernels might become more dominant and contribute more towards the total execution times, thus, partially masking the benefits obtained from the kernel speedups. The reduced margin of improvements in speedups of Figure 9.8 is because of this fact. Certain non-kernal bottlenecks can be alleviated by simple software optimizations. For instance, a data read in the non-kernel of the HOP application became redundant after the addition of the accelerator, which can be optimized further. Such optimizations are out of the scope of this work. But it stresses the point that this design opens up venues for software optimizations.

The results were presented with the accelerator performing at the same clock speed as the main processor (700 MHz). However, the sensitivity of the clock speed on the accelerator speedups was also studied by varying the clock speed of the accelerator. It was found that the kernel speedups do decrease as the clock speeds drop. But, even with

<table>
<thead>
<tr>
<th>Application</th>
<th>S</th>
<th>M</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>kMeans</td>
<td>23</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>Fuzzy kMeans</td>
<td>11</td>
<td>48</td>
<td>76</td>
</tr>
<tr>
<td>BIRCH</td>
<td>23</td>
<td>31</td>
<td>90</td>
</tr>
<tr>
<td>HOP</td>
<td>29</td>
<td>49</td>
<td>74</td>
</tr>
<tr>
<td>Bayesian</td>
<td>31</td>
<td>48</td>
<td>58</td>
</tr>
</tbody>
</table>

Figure 9.8. Application speedups
a low clock speed of $<100\text{MHz}$, the accelerator produces a kernel speedup of at least $7X$ (max $150X$) for the smallest set.

The kernel can be easily extended to other data mining applications, and in fact, in its current form without any modifications to all data mining applications that use density, variance, and distance calculations in their core.

### 9.5. Summary

In recent years, the input data sizes and the problem complexities of data mining applications have increased significantly, leading to slower execution of these applications. Faster engines that perform the application tasks quickly and efficiently are essential. The algorithmic, architectural and statistical analysis of data mining applications proved that data mining applications are different from conventional applications, which makes it difficult to host them on existing computing systems. This fact is alleviated by using the above analysis results to extract the core kernels of data mining applications, which are basically core computations that are performed repeatedly on a set of large data. Based on this, an accelerator-based data mining system was designed, which aims to speedup these computation kernels by providing custom logic for the underlying kernel operations. This accelerator is designed, modeled using a realistic experimental setup, and then evaluated by executing a set of data mining applications on it. The results suggested that the kernels see a minimum speedup of $10X$ (and a maximum speedup of $3800X$) when these kernels are executed on the accelerator instead of on the main processor. As a result of this, the applications also speed up (up to $90X$ on
a real data set). When the accelerator resources are scaled up, the kernels (and also the application) are able to scale with the resources.
CHAPTER 10

Conclusion

Data in terabytes range is not uncommon today and is expected to reach in petabytes range for many application domains in science, engineering, businesses, bioinformatics, medicine and security among others in the near future. In addition, the complexity of data is also increasing. As smarter tools and application data access techniques are emerging, system performance seem to lag behind. In future, user expectations are also bound to go up. More accurate results would be expected from such large data in a timely manner. Hence, current systems have to be redesigned. This research work uses this fact as a motivating factor to propose new architectural techniques that enable high speed access to data. A three-tier data driven approach is taken to propose new architectural designs and techniques. The three perspectives of data include streaming data, structured databases and new-age massive datasets (could be structured or unstructured).

Streaming data is widely popular and is still emerging. In this work, new scheduling and resource allocation strategies are proposed for such systems by taking into consideration the data handled by these systems. These techniques efficiently execute the streaming applications by ensuring optimal usage of the underlying processor resources. Both performance and energy improves dramatically when the proposed schemes are deployed in existing heterogeneous systems. This study also identified another key
fact. Current analysis frameworks lack the potential to study performance and energy consumption within a single framework. Hence, a new framework of analysis called *Energy-Resource Efficiency* is proposed, which is comprehensive in studying performance and energy consumption of any system irrespective of its type.

The second area of focus for this work is modern database systems. Storage technology has evolved significantly in the recent years. Most database systems assume that the traditional underlying storage architectures are still valid. In this study, modern memory technology is considered as a motivation to tune and adapt modern DBMS. First, hardware schemes are proposed. These schemes automatically adjust the memory configuration to the needs of the database. The memory dynamically tunes itself to improve the performance and the energy consumption of the system. To make the schemes comprehensive, the query optimizer is also modified to reflect the change in the storage architectures. Specifically, intra- and inter-query optimizations are proposed, namely: *data windowing* and *multi-query restructuring*. New storage paradigms like *Micro-Electro Mechanical Systems* (MEMS) are also considered in this work. Designing database layouts for emerging storage architectures is a challenging problem. New layouts are proposed for MEMS after considering the query access patterns and the inherent storage device characteristics.

Given the current data growth rates, smarter data analysis tools become absolutely essential. Data mining is emerging as an excellent tool to automatically extract useful information from large datasets. The growth of data, the advancements in data mining tools and the user requirements is totally incongruent to the improvement in general
purpose system performance. Because of this reason, data mining applications tend to run slower and are becoming to be one of the challenging workloads in the high performance computing community. While several high performance schemes have been proposed, this research work takes a non traditional approach to achieving tremendous improvements in the execution times of data mining applications. For this, applications are extensively characterized and then the inherent characteristics of these application (so called kernels) are extracted. These kernels are then accelerated by designing a new data mining system. This accelerator driven data mining system massively speeds up the kernels of data mining applications to achieve speedups that there never possible using traditional high performance techniques.
References


