AMERICANUNIVERSITY OF BEIRUT

DESIGN METHODS FOR SOFTWARE ENERGY-AWARE PROFILING AND COMPUTING

by
MEHIAR MOHAMED ZOUHAIR DABBAGH

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
August 2012

AMERICAN UNIVERSITY OF BEIRUT

DESIGN METHODS FOR SOFTWARE ENERGY-AWARE PROFILING AND COMPUTING

by

MEHIAR MOHAMED ZOUHAIR DABBAGH

Approved by:

Dr. Hazem Hajj, Professor Electrical and Computer Engineering	Advisor
Dr. Mohammad Mansour, Professor Electrical and Computer Engineering	Member of Committee
Dr. Wassim El-Hajj, Professor Computer Science Department	Member of Committee

Date of thesis defense: August14, 2012

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

I, MehiarMohamadZouhairDabbagh

☐ authorize the American University of Beirut to supply copies of my thesis to libraries or individuals upon request.

☐ do not authorize the American University of Beirut to supply copies of my thesis to libraries or individuals for a period of two years starting with the date of the thesis defense.

Signature

Date

ACKNOWLEDGMENTS

I would like to express my greatest gratitude tomy advisor Professor Hazem Hajj for all the guidance, the advice and the great support that I received during my masters' study. I would like also to thank the committee members Professor Wassim El-Hajj and Professor Mohamad Mansour for all their help and for the great research experience and the knowledge that I gained from their supervision throughout this work. I am also grateful to Professor TawfikArabi and the MER program for supporting my research work.

My deep thanks go to my parents and my sister Rima for their love and continuous encouragement and support.

Finally, special thanks to the great people and friends that I met here at AUB. You were my family here in Lebanon. The list of great people that I met here goes forever but I would like to mention some of them. Thank you FarahYahya, CesarGhali, Naoum Sayegh, Ali Ali, Jalal Awed, Mounira Sayour, Doaa Al-Otoom, Imad Sarji and the great friends in Qatar lab and 202 lab for all the great moments that I shared with you in these two years.

AN ABSTRACT OF THE THESIS OF

Mehiar Mohamed ZouhairDabbagh for Master of Engineering
Major:Network and Security

Title: Design Methods for Software Energy-Aware Profiling and Computing

Energy has become an important factor in different aspects of computing technologies, such as reducing server energy for lower financial costs, or mobile device energy for longer battery life. In fact, energy efficiency is the major challenge for Exascale computing and beyond. The goal of our work is to present a unique top-down design methodology for developing energy aware algorithms based on energy profiling. The key idea revolves around identifying and measuring components of code with high energy consumption. Optimizing these software components for performance or energy leads to a major impact on overall computational efficiency. As a result, there are two major contributions in our work: 1. A method for identifying components with high energy consumption in compute-intensive applications. We target operations called kernels, which are frequently used operations in the algorithm. 2. A method for estimating software energy for the identified software components, in particular for kernels and load/store operations. The energy evaluation method involves using isolated code with assembly injection. Furthermore, to ensure reliable results, we use physical energy measurements conducted on specially instrumented circuit boards to provide actual and not just simulated measurements. To evaluate the proposed methods, we conducted three cases studies using well-known DM algorithms: back-propagation (BP) neural network, K-Nearest Neighbors, and Linear Regression. We then conducted a benchmark of energy kernelsfor most commonly used DM algorithms. The results highlight the contributions of kernels and memory energy to total algorithms' energy. These studies form building blocks for understanding software energy distribution and ultimately energy optimization for DM algorithms

CONTENTS

ACKNOWLEDGEMENTS.. v

ABSTRACT.. vi

LIST OF ILLUSTRATIONS.. xi

LIST OF TABLES.. xiii

LIST OF ABBREVIATIONS.. xiv

Chapter

1. INTRODUCTION.. 1

2. RELATED WORK.. 4

2.1 Overview of Energy Optimization Techniques.. 4

2.2 Overview of Energy Profiling Techniques.. 6

2.3 Conclusion from Related Work.. 8

3. PROPOSED METHODS.. 10

3.1. Methodology for Energy-aware Algorithms.. 10

3.2. Methodology to estimate the energy of the kernels.. 15

3.2.1. Energy Cost of Primitive Kernels.. 15

3.2.2. Energy Cost of Aggregate Kernels.. 18

3.2.3. Energy Cost of Load and Store Operations.. 20

4. EXPERIMENTS AND RESULTS.. 22

4.1. Experiment Setup for Energy Measurement.. 23

4.2. Kernel Identification in Common Data Mining Algorithms.. 25

4.3. Case Study for Energy-Aware Profiling and Computing with Focus on Primitive Kernels 29

4.4. Additional Case Studies with Considerations for Aggregate-level Kernels and Load/Store Energy 39

4.4.1. Energy-Aware Computing – KNN Case Study 40

4.4.2. Energy-Aware Computing – LR Case Study 45

5. CONCLUSION AND FUTURE WORK 48

REFERENCES 50

ILLUSTRATIONS

Figure Page

3.1 - Proposed four-step methodology for an energy-aware algorithm ……………...11

3.2 - Proposed System for collection of physical energy measurements……………....13

3.3 - Isolated code to determine primitive kernels energy………………………….. 16

3.4 - Isolated code to measure theenergy cost of the addition primitive kernel ……...17

4.1 - Number of times BP primitive kernels are executed……………………….........31

4.2 - WATTCH results for the energy of BP primitive kernels……………………........32

4.3 - Contribution of BP primitive kernels to the overall cost……………………….. 33

4.4 - The impact of reducing different BP primitive kernels by 25%
on the overall algorithm for different iterations…………………….……………. 34

4.5 -A comparison between first output and approximated output…..……….………. 36

4.6 -A comparison between second output and approximated output ……..……….... 36

4.7 - Number of required iterations to reach MSE<0.8 …………………….………….39

4.8 - Frequency of KNN aggregate kernels for the considered dataset…….………..... 42

4.9 - Energy cost of KNN aggregate kernels for the considered dataset……..……...... 42

4.10 - Energy contribution of KNN aggregate kernels to the total energy……..……... 43

4.11 - Effect of reducing the energy of KNN aggregate kernels by 25%……………..... 46

4.12 - Energy contribution of LR aggregate kernels to the total energy……………....... 47

4.13 - Energy effect of reducing LR aggregate kernels by 25% on total energy …….. 45

TABLES

Table Page

3.1 - Aggregate kernels that are popular in many Data Mining algorithms along with the energy cost of these aggregate kernels based on the energy cost of primitive kernels and dataset properties …………………………..……… 20

3.2 - Different types of load and store assembly instructions ………………………………….. 21

4.1 - A summary of the studied algorithms, the considered kernel level in the energy analysis, the energy collection approach, the studied architecture along with the experiments that will be conducted for each studied algorithm and the objective of these experiments…………………….. 23

4.2 - Primitive and aggregate kernels of top data mining algorithms based on the mathematical equations that are executed repeatedly in these algorithms………………………………………………………………………………………….. 27

4.3 - Results of Asymptotic Analysis for of BP primitive kernels………………………………... 30

4.4 - A cost comparison with and without normalizing the training set……………………….. 39

4.5 -Energy cost of primitive kernels using physical measurements…………………………….. 40

4.6 - Energy cost of move operations (load/store) using physical measurements…………………………………………………………………………………………. 40

4.7 - Order of KNN kernels…………………………………………………………………………………….. 40

CHAPTER 1

INTRODUCTION

Energy has become an important factor in different aspects of computing technologies, such as reducing server energy for lower financial costs, or mobile device energy for longer battery life. In fact, energy efficiency is the major challenge for Exascale computing and beyond [1]. Furthermore, energy costs in large computer centers are having impacts on the environment. In fact, it was noted in [2] that Information and Communication Technology (ICT) is responsible for 2% of the global emissions, equivalent to aviation. These energy challenges have driven the search for efficient ways to save energy.

Reduction of energy can be achieved by performing optimizations at the platform level, or examining different computer layers and their interactions, including hardware, architecture, compiler, operating system, and application. The first step in optimizing energy is to address the problem of determining where and how much energy is consumed. While many researchers and companies have driven extensive research at the hardware and architecture levels, fewer efforts, such as the one in this paper, have focused on starting from the application layer, and examining a top-down energy reduction approach.

Towards the goal of reducing the energy consumed by software applications, we present in this work two new contributions related to software energy profiling: 1. A method for identifying software components with high energy consumption. We target operations called kernels, which are frequently used operations in the algorithm. 2. A method for estimating the energy consumed by the identified software components, in particular for kernels and load/store operations. The energy evaluation method involves using isolated code with assembly injection. Furthermore, to ensure reliable results, physical energy measurements conducted on specially instrumented circuit boards are collected to provide actual and not just simulated measurements.

Previous work on energy profiling aims at providing a hardware component-wise energy breakdown that shows the energy contribution of the different hardware components when running the whole software application from beginning till the end. Our work is different from previous techniques as it provides a software-wise energy breakdown where we show how energy is consumed among the different parts of the software code. We demonstrate the different steps of our profiling methodology by applying them on data mining (DM) algorithms. The reason behind choosing DM algorithms is twofold. First, DM algorithms are widely used in many domains (bioinformatics, business, social networks, etc.). The second reason is due to the nature of DM algorithms. During the training phase, DM algorithms usually have a segment of code that is repeatedly executed for different tuples. Therefore, optimizing the energy of these segments will be highly reflected on the overall energy consumption. Researchers in the field of data mining have focused on improving accuracy and performance of data mining algorithms. To the best of our knowledge, no previous work has analyzed these algorithms from the energy efficiency point of view.

The major contributions of our thesis work are:

- A four-step method for identifying software components with high energy consumption.
- An approach for software energy assessment with the use of assembly injection. The proposedapproach helps in isolating the impact of cache misses when measuring kernel energy.

- An approach for measuring the energy cost of memory to cache load and store operations.
- An energy assessment for popular kernels from basic kernels such as addition and subtraction to more aggregate equations and kernel operations such as Euclidean distance.
- Simulation results are supported by physical energy measurements to prove that our methods can rely on either simulation tools or special instrumented boards in order to profile the studied algorithm.
- Threecasestudiesare conducted to illustrate our design methods for energy-profiling with three algorithms: Back Propagation (BP), K-Nearest Neighbors (KNN) and Linear Regression (LR).
- Two techniques to reduce the energy of parts with high costs based on approximation-energy trade off and based on preprocessing techniques to simplify calculations.
- An energy evaluation for selected data mining algorithms, where we generalize our approach and estimate the energy cost of most frequent kernels that are widely used in those different algorithms.

The rest of the thesis reportis organized as follows: In chapterII, we present literature review on the topics of energy optimization and energy profiling. In chapterIII, we present our proposed methodologies. We explain in details the different steps of the simplified four-step methodology for creating an energy-aware algorithm. We also show the approach to measure the kernel's energy. In chapterIV, we present our case studies and discuss the results. In chapterV, we conclude and present our future work.

CHAPTER 2

RELATED WORK

Energy optimization techniques have been suggested for the various layers of the computer platform. In this chapter, we overview these techniques in addition to describing the methods used in energy profiling. We end the chapterby highlighting how our work differs from the other works.

2.1 Overview of Energy Optimization Techniques

The increasing demand for saving energy has made researchers go beyond the low-level circuit layer to explore optimizations in the upper platform layers including the compiler, the operating system, and the application layers.

In the compiler layer, optimization techniques have been used to improve performance by transforming codes into more efficient translations with lower execution time. These techniques can be also used to save energy. In [6-9], different compiler optimization techniques (e.g. loop unrolling and instruction rescheduling) were applied on benchmarks in order to save energy. The main limitation of these techniques was that the effects of the methods on both performance and energy varied from one code to another. The setof possible compiler optimization techniques is huge. Consequently, determining the best combination of optimization techniques using brute-force search is infeasible. Therefore, heuristic methods were used to determine the combination of techniques with the highest energy saving and highest performance for a certain code. An example of such an approach was presented in

[10] where the authors proposed a heuristic method that prunedunpromising optimization techniques to reduce the search space. Another attempt to overcome the problem of the large search space was presented in [11-14]. The authors used machine learning [11-13] and genetic algorithms [14] to predict the best combination and the best sequence of compiler optimizations such that the energy or the delay for running a given code s minimized.

In the operating system layer, researchers worked on optimizing and efficiently using the different power management policies that are used to control power. These policies switch idle devices into lower power states if the systems predict that the full capacity of these devices will not be needed for the coming events. It is worth noting that in OS power policies, the device is not switched into a lower power state whenever it is idle because of the high energy required to wake the device up. Therefore, the device is switched into a lower power state only if the systempredicts that it will remain idle for a time long enough to compensate for the transition energy. Additional information about how these predictive techniques and algorithms work is provided in [15-17].In [18] a comparison is presented between the power policies in windows 7 and windows vista. Results showed that windows 7 achieved higher energy savings due to more available low power-states and better idleness predictive techniques. Another technique that operating systems use to manage power is Dynamic Voltage and Frequency Scaling (DVFS). In (DVFS), the operating voltage and frequency for executing a task are reduced in order to save the consumed energy. (DVFS) leads to a great saving of energy but has a negative effect on performance especially if the code is computation-bound [19].

In the application layer, the focus has been on specific applications. Examples include energy optization for sensor network applications [20], WiMAX frame construction [21], multimedia [22], bioinformatics [23], and file access [24]. The optimization methods for at the application layer can be classified according to [25] into three main categories: contextual awareness, data efficiency and computational efficiency. In contextual awareness, applications change their behavior based on the available energy. A dynamic compilation that adapts battery changes was proposed in [26], where precompiled parts of the code that have low energy were used when the battery was low.In data efficiency techniques, data was stored, accessed, and transferred in an energy efficient way. For instance, in [27] the authors showed that considerable energy savings can be achieved by using SSD instead of HDD as a storage device.In computational efficiency, the maximum available system capabilities were used so that the work finishes early, enabling the switching of the devices into idle mode to save energy. Examples of computational efficiency techniques include the use of parallel programming to reduce the execution time or any other technique that enhances performance. The authors in [28] used parallel programming and examined the number of required parallel nodes such that the overall energy for running different benchmarks is minimized. Examples of computational efficiency also include the work in [29,30] where the authors proposed parallelizing different machine learning and data mining algorithms by distributing calculations on different cores in order to improve runtime performance.

2.2 Overview of Energy Profiling Techniques

Energy profiling is often considered a critical step in identifying bottlenecks for energy optimization. In software, energy estimation methods can provide insight into the power consumption of the different parts of the code and into the different components of the architecture. As a result, energy profiling has also received significant research attention. All the previously mentioned work used one of two ways to estimate the energy required to run a program on a given architecture: Physical measurements or Simultation.

With physically measurements, the current and voltage of the different components were collected using ammeters and special acquisition systems. These methods have high accuracy but they require special equipment, such as the work in [35 - 38]. Researchers executed benchmark applications on special instrumented boards and collected energy measurements. However, their work provided a hardware component-wise energy breakdown showing energy contribution of each component of the platform when executing the whole benchmark code. The main limitation of the work in [35 – 38] is that the methods are more hardware centric, and do not show how energy is consumed among the different parts of code.

With simulation, themethodsare inexpensive and allow users to analyze the performance and power behavior with a cycle level of granularity for the different platform components. However, these methods are less accurate than the methods using physical measurements. Simulation is a good alternative when the special equipment and boards of a certain architecture are not available. In [39] researchers tried to show how energy is consumed among the different parts of code. The main idea was collecting traces from the operating system and then estimating the active time and the idle time of the different components using performance profiling tools. The active time and idle time were multiplied by two constant powers that represented the active and idle power of the selected devices. This approach assumes that the power consumed by any platform component is constant in the active state regardless of the nature of workload. Although this work showed how energy is consumed among the different parts of code, it had many assumptions about power. As a result, the method is not as accurate as when usingactualphysical energy measurements. Our proposed methods for energy profiling are different than the work in [39] as our methods do not assume that power is constant during all the active state and thus provide more accurate energy estimations. In our work, simulation tools are supported by physical measurements in order to validate our profiling methods rather than relying only on simulation tools as in [39].

There have been other enhancements to the different energy profiling techniques. In [31], the authors proposed different frameworks for providing energy and thermal profiles as enhancements to previous techniques. In [32], the authors proposed solutions to reduce the time required to estimate the energy cost of long code segments using clustering techniques.

2.3 Conclusion from Related Work

While previous work has focused on providing a hardware centric energy profiling for different applications [35 – 38], none of the previous worktargeted profiling how energy is consumed among the different parts of the application code based on accurate energy costs. The only attempt to profile how energy is consumed among the different parts of code was in [39] and is based on performance simulation tools which have many assumptions about power. The objective of our work is to provide methods for energy profiling that show how much percent each part of code contributed to the total energy and that do not assume that the power is constant during the active state in order to have higher accuracy.

After determining the energy contribution of the different parts of code, we provide some techniques that can be used to reduce the energy cost of these energy hotspots. Unlike all the optimization techniques used to reduce energy in general applications (computational efficiency [28-30], data efficiency [27], and power-aware behavior [26]), our technique takes advantage of the algorithm specifics and investigates the opportunities that might lead to energy savings. For instance, we updated the BP algorithm to use approximation techniques via lookup tables and preprocessing techniques via normalization leading to considerable improvements in both performance and energy.

CHAPTER 3

PROPOSED METHODS

This chapter describes the two proposed methods for: energy-awareness, and for estimating the energy of compute-intensive components in code. These two methods are related in that the energy estimation method is used in one of the steps for creating energy-aware algorithm. However, it is described separately due to its importance and general applicability.

3.1. Methodology for Energy-aware Algorithms

We propose a four-step process for creating energy-aware algorithm. We explain briefly the objective of each step of our methodology. The purpose of these steps is to analyze the program details and identify opportunities for reducing energy consumption. Figure3.1 shows the flow of our proposed four-step methodology.

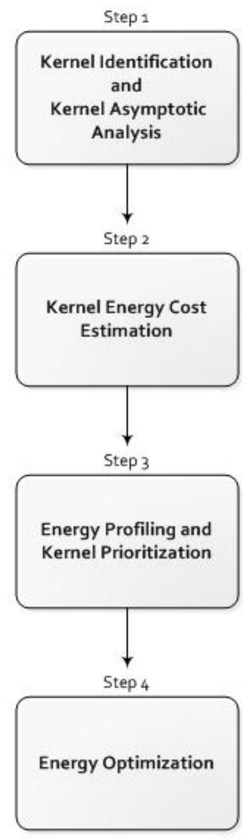
Step 1
Kernel Identification and Kernel Asymptotic Analysis
Step 2
Kernel Energy Cost Estimation
Step 3
Energy Profiling and Kernel Prioritization
Step 4
Energy Optimization

Figure 3.1 - Proposed four-step methodology for an energy-aware algorithm.

Step 1: Kernel Identification and Kernel Asymptotic Analysis: Kernels are operations that are repeatedly used in the algorithm, often found in “loops”. A typical data mining algorithm may have several repeated mathematical operations, producing a specific set of kernels. We distinguish two types of kernels normally found in computational algorithms: primitive kernels and aggregate kernels. Primitive kernels are the basic operations such as: addition, multiplication, division, subtraction, logarithm and exponential. Aggregate kernels are equations that are composed of several primitive kernels such as: Information gain, Euclidean distance, and normalization. In this step of the methodology, the primitive and aggregate kernels are identified by examination of the code. Asymptotic analysis is then conducted, where the number of execution times for each kernel operations is determined as a function of the dataset properties such as the number of tuples and the number of features.

*Step 2: Kernel Energy Cost Estimation:*Once the kernels are identified from the previous step, the goal of this step is to get the energy measurements for the identified kernels, and the load/store transactions representing memory and cache interactions. The measurements are then used for profiling the code, and prioritizing energy optimization opportunities. The method used to get these measurements step is an essential part of our work, and is described separately in section 3.2. The method relies on either simulation tools or on physical measurements in order to get energy numbers. Example of simulation tools are Simplescalar and WATTCH [7] which emulate the behavior of computer architectures and give an estimated energy cost for code execution. Physical measurements require special equipments. The setup for energy collection and the way measurements are collected are shown in Figure 3.2. The setup consists of a specially instrumented board that collects the current and voltage of the major platform components such as CPU and memory. These measurements are fed to a Data Acquisition System (DAQ) which converts collected measurements from analog to digital based on a pre-set sampling rate, and stores the collected traces. The collected traces are then passed to a monitoring computer that shows how power varies over time, and calculates the average power and energy for the executed code.

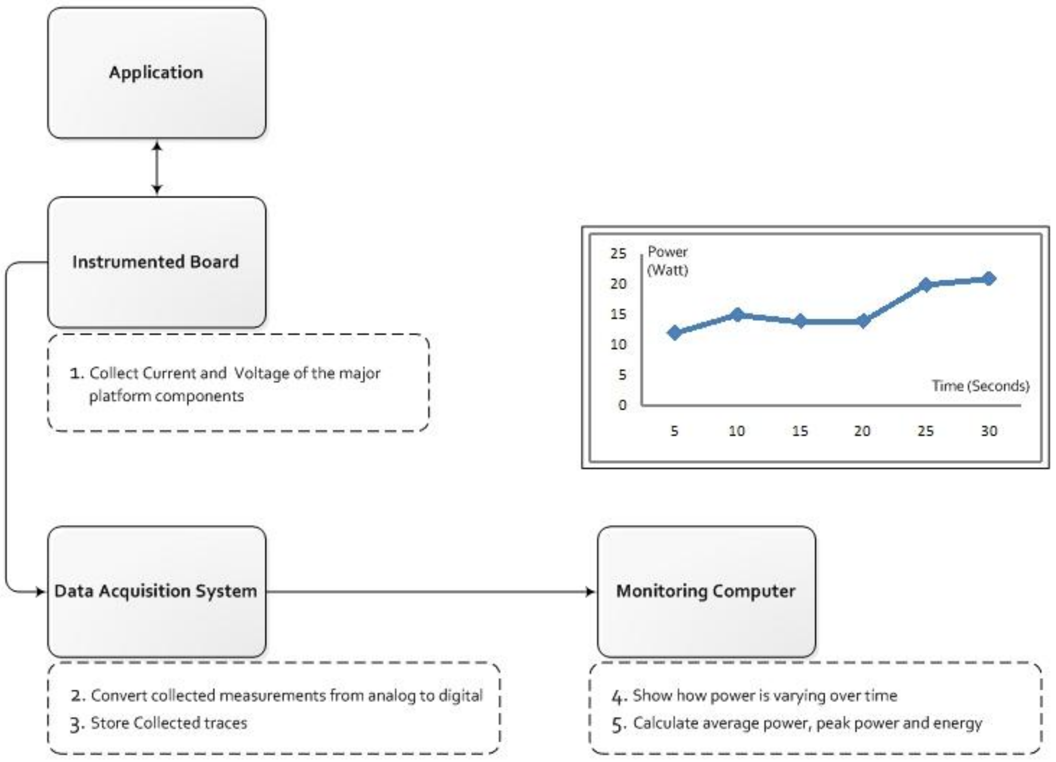
Application
Instrumented Board
1. Collect Current and Voltage of the major platform components
Data Acquisition System
2. Convert collected measurements from analog to digital
3. Store Collected traces
Monitoring Computer
4. Show how power is varying over time
5. Calculate average power, peak power and energy
Power (Watt)
Time (Seconds)
25
20
15
10
5
0
5
10
15
20
25
30

Figure 3.2 - Proposed System for collection of physical energy measurements.

*Step 3: Energy Profiling and prioritizing kernels in terms of energy impact:*This step consists of profiling the code by determining the overall energy of the algorithm and the energy contributions of each kernel relative to the overall energy. The energy contribution of each kernel is measured based on the results of the two previous steps, by multiplying the energy cost of executing a single kernel by the number of time this kernel is executed. Once the energies of all kernels are determined, the kernels can be prioritized for optimization according to their energy cost. At this stage, further assessment can be conducted into total potential energy saving based on kernel energy reduction. The kernel that gives the highest overall energy reduction is the best target for energy reduction.

*Step 4: Energy Optimization:*Research on code optimization is extensive, and is beyond the scope of our work. But we propose some options here to close the loop on the energy-awareness process. The results of the previous step are used in the optimization process to give an indication of which kernels to optimize for energy. Several approaches can be useful for compute-intensive application including:

a) Preprocessing techniques to simplify algorithm computations. Examples of preprocessing techniques include data normalization and data reduction. The most important aspect when applying these preprocessing techniques is to make sure that the energy cost of preprocessing data in addition to the energy cost of running the algorithm for the pre-processed dataset is smaller than the energy cost of running the algorithm on the original dataset without preprocessing. The results of the algorithm with the preprocessed dataset should be within acceptable range of the results with the original dataset.

b) Using alternative approximations to the kernels with less computations and lower energy, but with a tradeoff for lower accuracy. In this scenario also, it is important to assess the error resulting from the approximation to make sure that the algorithm still yields acceptable results.

c) Using alternative hardware implementation of the instruction set with lower energy and without compromising performance. For example a totally new optimized instruction set such as those used in DSP processors can be utilized. In this case, there is no need to check for accuracy issues, since it is expected to have the same kernel but with different implementation.

d) Use the best set of compiler optimization techniques for kernels with high energy cost rather than searching for the best set of optimization technique for the whole algorithm, as it requires less time and it will produce significant energy savings.

The main focus in our work is on the first three steps as they provide the required methodology for energy profiling. However, in one of the case studies with BP algorithm, we also study and demonstrate the opportunities for optimizing algorithm by using approximation-energy tradeoffs and by using preprocessing techniques. We explain in the next subsection how to measure the energy of kernels needed for step 2.

3.2. Methodology to estimate the energy of the kernels

The goal in this section is to find an accurate and efficient method to measure the energy consumed by kernels. We explain first how to measure the energy of primitive kernels and then show how to derive the energy cost of aggregate kernels. This section also includes the method to measure the energy cost of the load and store operations.

3.2.1. Energy Cost of Primitive Kernels

As described earlier in the thesis report, primitive kernels are the basic operations in code. Examples include addition, subtraction, logarithmic, etc… To estimate kernel energy, the main idea is to isolate the kernel in a separate controlled piece of code, where the kernel energy can be measured by itself. To reduce noise in the measurement, to the code is run many times, say N times, then calculate the average energy by dividing total energy measured by N.

The isolated code consists of the kernel and only the needed variables to run the kernel. Figure 3.3 gives an illustration of the isolated pseudo-code and desired kernel. The code includes the primitive kernel operation, and initialization of the variables.

```
1)  Initialize kernel_variables
2)  For (i=0;i<N;i++){
3)  Primitive_Kernel_execute(kernel_variables);
4)  }
```

Figure 3.3 - Isolated code to determine primitive kernels energy

To assess the energy of the kernel alone, three steps are executed:

1) In the first step, the isolated code is executed N times, and the energy is measured, call it E_{Kernel}
2) In the second step, the primitive kernel operation is removed from the code and replaced by a “no op”. The code is then executed again N times, and the energy cost is measured for the isolated code, call it $E_{woKernel}$.
3) Finally, the two energy measurements are subtracted and then averaged to give the average energy of the desired kernel:

$$\mathrm{E_{Kernel}} = \frac{\mathrm{E_{wkernel}} - \mathrm{E_{woKernel}}}{\mathrm{N}} \qquad (1)$$

One challenge in the proposed approach is that at compile time, the high-level code of the kernel operation is translated into more than one assembly instruction. These assembly instructions include load operations and the actual primitive operation instructions. The energy of the load operation depends on whether or not there is a cache hit and thus the obtained energy costs for the primitive kernel operation may not be accurate. To address this problem, we propose the use of assembly injection in the isolated code to further isolate the energy cost of executing the primitive operation without the impact of load/store operations. This approach also helps at estimating the impact of load/store operations in code and its overall contribution, which is further described in subsection. 3.2.3

As an example, Figure 3.4 shows the isolated code for assessing the energy of “addition” operation using assembly injection. In lines 1 and 2 we declare the variables needed to execute the primitive kernel operation. Line 3 represents the “for” loop that is executed N times. Lines 4 to 6 show the injected assembly code in the body of the “for” loop. In lines 4 and 5, the declared variables are loaded from the stack to registers “eax” and “ebx” respectively. In line 6, the addition primitive kernel operation is executed on the register values. The energy cost of the addition operation is calculated by subtracting the energy cost of executing the code without the addition operation (without line 6) from that with the addition operation and dividing the result by N.

```
1)  Declare x;
2)  Declare y;
3)  For (i=0;i<N;i++){
4)  Asm(”move x, eax”);
5)  Asm(”move y, ebx”);
6)  Asm(”add eax, ebx”);
7)  }
```

Figure 3.4 - Isolated code with assembly injection used to determine theenergy cost of the addition primitive kernel.

If the isolated codes in Figure 3.4 did not include assembly injected code (lines 4 to 6) but rather the high-level code (z=x+y;), the estimated energy cost would be inaccurate. This is due to the fact that the high-level code (z=x+y;) gets translated at compile time into assembly instructions that include not only the addition operation but also the load and store operations. The costs of load and store operations vary depending on whether or not a cache miss occurs, which produces even more sources of inaccuracy.

In our work and to obtain accurate estimation, we propose the use of the assembly injection approach to estimate the energy cost of the primitive kernels.

3.2.2. Energy Cost of Aggregate Kernels

Aggregate kernels are composed of multiple primitive kernels. Information Gain is such an example, where it is composed of logarithmic, addition, and probability, which is in turn composed of division. The energy of aggregate kernels can be estimated by isolating the kernels as described in the previous section, but isolating load/store operations becomes more complex. As an alternative, we describe an approach that can be conducted without the need to re-running isolated code for aggregate kernels. The idea is to derive the energy cost of executing an aggregate kernel based on the energy costs of the related primitive kernels. First the frequencies of the primitive kernels composing the aggregate kernel are determined. The energy of the aggregate kernel is then determined by summing the product of the energies of the primitive kernels multiplied by their frequency of occurrence.

As an example, consider the Euclidean distance given by equation (2).

$$\mathrm{dist}(X1, X2) = \sqrt{\sum_{i=1}^{N_f} (x_{1i} - x_{2i})^2} \qquad (2)$$

Where X1, and X2 are two tuples in the dataset. x_{ji}: is the feature number (i) for the tuple (j). N_f: is the number of features in the studied dataset.

From equation (2), it can be seen that the number of subtractions, additions and square operations is dependent on the number of features in the dataset. Therefore, the energy cost of the Euclidean distance is dependent on the properties of the dataset such as the number of features. From equation (2), the Euclidean distance includes: $(N_f - 1)$ additions, (N_f) subtractions, (N_f) square operations, and one square root operation. Based on the energy costs of the addition, subtraction, square and square root operations (refer to them as$E_+, E_-, E_{sq}, E_{sqrt}$ respectively), the energy cost of the Euclidean distance can be calculated as in equation (3):

$$E_{Euclidean} = E_+ \times (N_f - 1) + (E_- + E_{sq}) \times N_f + E_{sqrt} \qquad (3)$$

This method is further applied to well-known data mining algorithms. Table 3.1 provides the summary of the finding for aggregate kernels in common DM algorithms along with the related primitive kernels for each. The table also includes the energy cost of the aggregate kernels as a function of the energy cost of the primitive kernels and the dataset properties. We refer to the energy cost of the primitive kernels addition, subtraction, multiplication, division, logarithm, square and square root by $E_+, E_-, E_\times, E_\backslash, E_{log}, E_{sq}, E_{sqrt}$ respectively.$N_f, N_{class}, N_{tuples}$ refer to the number of features, classes and tuples respectively in the dataset.

Table 3.1 - Aggregate kernels that are popular in many Data Mining algorithms along with the energy cost of these aggregate kernels based on the energy cost of primitive kernels and dataset properties.

Aggregate Kernel	Mathematical Equation	Energy Cost of Aggregate Kernel
Euclidean Distance	$\mathrm{dist}(X1, X2) = \sqrt{\sum_{i=1}^{N_f} (x_{1i} - x_{2i})^2}$	$(N_f \times (E_{sq} + E_{-}) + (N_f - 1) \times (E_{+} + E_{sqrt})$
Info	$\mathrm{Info}(D) = -1 \times \sum_{i=1}^{N_{class}} p_i \times \log(p_i)$	$(N_{Class} + 1) \times E_{\times} + (N_{Class} - 1) \times E_{+} + N_{Class} \times E_{log}$
Regression Coefficient (w_1)	$w_1 = \frac{\sum_{i=1}^{N_{tuples}} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N_{tuples}} (x_i - \bar{x})^2}$	$N_{tuples} \times (3 \times E_{-} + E_{\times} + E_{sq}) + (N_{tuples} - 1) \times 2 \times E_{+} + E_{/}$
Regression Coefficient (w_2)	$w_2 = \bar{y} - w_1 \times \bar{x}$	$E_{-} + E_{\times}$
Average	$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$	$(N - 1) \times E_{+} + E_{/}$
Gini	$\mathrm{gini}(D) = 1 - \sum_{j=1}^{N_{class}} p_j^2$	$N_{Class} \times E_{sq} + (N_{Class} - 1) \times E_{+} + E_{-}$
Normalization	$V' = \frac{V - \min}{\mathrm{range}}$	$(N - 1) \times E_{+} + E_{/}$

3.2.3. Energy Cost of Load and Store Operations

When executing aggregate kernels like the Euclidean distance, the compiled assembly code does not only include primitive kernel operations (subtraction, addition …etc) but also includes load and store operations. Assembly injection, as described earlier, can help in isolating the impact of the kernel from the impact of the load/store operation. Furthermore, it is important to estimate the energy cost of these load and store operations for memory and cache exchange, as they may represent significant contributions of energy consumption. Load and store operations can be classified into five types based on the source and destination of the move operation as shown in Table 3.2. The source can be either a value, a register or the stack whereas the destination is either a register or the stack. The energy cost of the memory and cache exchange is calculated by multiplying the number of times each type of load and store operation is executed by the energy cost of the operation. The frequency of execution for the load and store operations is determined by examining the compiled assembly code of the code and counting the number of times each type of these operations is executed in the assembly code. The energy cost of the load and store operations can then be calculated following the method described for estimating energy for primitive kernels. The only difference is that only a load/store instruction is injected in the "for" loop. The isolated code is then executed with and without the injected load/store assembly operation. The average energy of a load/store operation is calculated using equation (1).

Table 3.2 - Different types of load and store assembly instructions.

Move Instruction	Example
Move value, stack	move $10, esp(4)
move register, stack	move eax, esp(4)
move stack, register	move esp(4), eax
move value, register	move $10, eax
move register, register	move eax, edx

CHAPTER 4

EXPERIMENTS AND RESULTS

In this chapter, we conduct experiments to demonstrate the four-step methodology for creating an energy-aware algorithm, and to illustrate the use of the method for estimating energies for kernels and load/store operations.

We explain in the first subsection the experiment setup for the simulation tools and the special equipments that are used in our experiments. In the second subsection, we identify primitive and aggregate kernels for top Data Mining algorithms. Next, three case studies are conducted on three algorithms to validate our four-step methodology for energy-awareness. Table 4.1 summarizes the studied algorithms, how energy measurements are collected, the studied architecture, the conducted experiments and the objective of these experiments for each algorithm.

Table 4.1 - A summary of the studied algorithms, the considered kernel level in the energy analysis, the energy collection approach, the studied architecture along with the experiments that will be conducted for each studied algorithm and the objective of these experiments.

Algorithm	Kernel Level	Energy Collection Approach	Architecture	Experiment	Objective of Experiment
BP NN	Primitive level	Simulation tools	RISC	Determine the contribution of primitive kernels to the total energy	- Prioritize primitive kernels in term of Energy impact.
				Energy-approximation trade-off	- Present an approach that trades accuracy with energy. - Estimate the energy savings obtained by this technique - Study the effect of the approximation technique on the behavior of the algorithm
				Preprocessing techniques to save energy	- Present how preprocessing techniques can lead into energy savings - Estimate the overhead involved in preprocessing technique - Estimate the total energy saving obtained from preprocessing data.
KNN	Aggregate level	Physical Measurements	CISC	Determine the contribution of aggregate of kernels to the total energy	- Prioritize aggregate kernels in term of Energy impact.
LR	Aggregate level	Physical Measurements	CISC	Determine the contribution of aggregate kernels to the total energy	- Prioritize aggregate kernels in term of Energy impact.

4.1. Experiment Setup for Energy Measurement

There are two standard methods for collecting energy measurements, either through the use of specially instrumented boards or through the use of simulation running on top of computer architecture emulators. In our work, both approaches were used.

For simulation setup, we used Simplescalar [33] emulator and WATCH [7] simulation tools. Simplescalaremulatesthe execution of the code on a RISC architecture and WATTCH is used to give relative energy numbers that determine the energy cost of the different units of the architecture when the code is executed. In our simulation experiments, we used the default configurations forSimplescalar and WATTCH. Details of these configurations are found in [33]. Simple Scalar and WATTCH operates on any UNIX 32 bit operating system or requires the installation of a virtual machine in order to operate on Windows operating systems. It also requires the installation of a number of compiler tools including Bison, Yacc and Flex. Simple Scalar consists of a collection of microarchitecture simulators that emulate the microprocessor at different levels of detail. In our experiments, we used the “sim-outorder” simulator as it provides a detailed emulation of out of order micro architectures and models the different units including the cache, external memory and the branch prediction. WATTCH is run on top of Simple Scalar in order to provide an energy model for the studied architecture.

For the physical measurements, the hardware setup is shown in Figure 3.2. We run the code on a special board instrumented by Intel with sensors on all the platform components to measure current and voltage. The platform had an Intel® Core™ i7 CPU, 2.80GHz with 2 GB RAM memory. Windows 7 was installed as the operating system for the specially instrumented board. The power plan of Windows 7 was set to the balanced mode for all the experiments. A power plan is defined to be a collection of system and hardware settings that manage how power is consumed in the computer. There exist three power plan modes in Windows: power saver, balanced and high-performance modes. The balanced mode was chosen for our study since it is the default power plan mode that is used in Windows and since it does not sacrifice neither performance nor energy but tries to find the balance between these two constraints.

When running experiments to collect energy, all unnecessary programs are stopped, except for the OS and the basic processes it requires. Example of processes that must be running always in the background is the “Explorer.exe”, a user interface process that runs the windows graphical shell for the desktop, task bar, and Start menu.

The DAQ was collecting the current and voltage for the major components of the board. The DAQ was Fluke 2680 Series [40] data acquisition system. The sampling rate was set to the highest supported rate which is 20 samples/second.

The collected traces were passed through a hub from the data acquisition system to the monitoring computer. PACS program was installed on the monitoring computer. PACS is a software that shows how the collected traces are varying over time and that calculates the average power, peak power and energy of the different components of the platform over a certain period of time.

In both simulation and physical setups, measurements were repeated five times and the median value of the five measurements was taken to minimize noisy measurements.

4.2. Kernel Identification in Common Data Mining Algorithms

To illustrate the types of kernels commonly found in software, we test the methodology with data mining algorithms. The first step in the energy-aware method was used to derive the primitive and aggregate kernels for common Data Mining algorithms. The identified kernels are presented in Table 4.2. The aggregate kernels are composed of a collection of primitive kernels. As an example, the normalization aggregate kernel is composed of the two primitive kernels: subtraction and division. Based on the proposed method for estimating energy for aggregate kernels, the energy cost of the primitive kernels are sufficient to derive the energy cost of aggregate kernels.

Several observations can be concluded from the table, and that are relevant for the study of energy optimization for the DM field. It can be seen that the basic operations: addition, subtraction, multiplication and division are very common primitive kernels among the different algorithms compared to other less frequent primitive kernels such as the exponential function and the square root. Some of the aggregate kernels in Table 4 such as: Euclidean distance, average, probability and pattern matching are very common among a large set of algorithms. Other aggregate kernels are used specifically for a certain algorithm, but not in others. Example of such aggregate kernels is the kernel that is used to calculate the first and second regression coefficients in Linear Regression.

Table 4.2 - Primitive and aggregate kernels of top data mining algorithms based on the mathematical equations that are executed repeatedly in these algorithms.

DM Algorithm	Equation	Aggregate Kernels	Primitive Kernels
K-Nearest Neighbors	$\acute{v} = \frac{v - min}{range}$ $dist(X1, X2) = \sqrt{\sum_{i=1}^{N_f} (x_{1i} - x_{2i})^2}$ $dist_1 < dist_2?$	Normalization Distance Compare	Subtraction Division Square root Square Addition
Linear Regression	$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$ $w_1 = \frac{\sum_{i=1}^{N_{tuples}} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N_{tuples}} (x_i - \bar{x})^2}$ $w_2 = \bar{y} - w_1 \times \bar{x}$	Average Regression coefficient (w_1) Regression coefficient (w_2)	Addition Division Subtraction Multiplication Square
K-means Clustering	$dist(X1, X2) = \sqrt{\sum_{i=1}^{N_f} (x_{1i} - x_{2i})^2}$ $\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$ $dist_1 < dist_2?$	Distance Compare Average	Square root Square Subtraction Addition Division

Back Propagation Neural Network (BP NN)	$S_k^p = \sum_{\mathrm{j}} w_{j\mathrm{k}} y_{jk}^{p-1} + b_k$ $y^p = \frac{1}{1 + \mathrm{e}^{-\mathrm{S}^\mathrm{p}}}$ $e_j(n) = d_j(n) - y_j(n)$ $\Delta w_{jk} = \gamma \delta_k^p y_j^p$ $w_{jk}' = w_{jk} + \Delta w_{jk}$	Forward Stage Calculations Backward Stage Calculations	Multiplication Addition Division Exponential Subtraction
Decision trees - Id3 and C4.5	$info(D) = -\sum_{i=1}^{N_{class}} p_i \times \log(p_i)$ $info_A(D) = \sum_{i=1}^{N_{label}} \frac{\lvert D_i \rvert}{\lvert D \rvert} \times info(D_i)$ $Gain(A) = info(D) - info_A(D)$	Pattern Matching Probability Information	Multiplication Addition Logarithm Division Subtraction
Decision trees - Gini index	$gini(D) = 1 - \sum_{j=1}^{N_{class}} p_j^2$ $gini_A(D) = \frac{\lvert D_1 \rvert}{\lvert D \rvert} \times gini(D_1) + \frac{\lvert D_2 \rvert}{\lvert D \rvert} \times gini(D_2)$ $\Delta gini(A) = gini(D) - gini_A(D)$	Pattern Matching Probability Gini	Subtraction Addition Square Division Multiplication

Naïve Bayesian Classification	$P(C\|X) = P(X\|C) \times P(C)$ $P(X\|C_i) = \prod_{k=1}^{N} P(x_k\|C_i)$	Pattern Matching Probability Posteriori	Multiplication

4.3. Case Study for Energy-Aware Profiling and Computing with Focus on Primitive Kernels

BP algorithm was used as a case study for demonstrating the applicability of the four-step methodology in creating energy awareness, and the energy estimation method. The following notations are used to represent the data properties for the BP Neural Network:

IN: Number of input neurons. IN also represents the number of features in the dataset.

HN: Number of hidden neurons

ON: Number of output neurons

The training was repeated (N) times on different tuples from the dataset. To validate the theoretical findings and get actual energy measurement, a stand-alone code was developed in C to implement the BP algorithm, with the following typical choices of parameters: IN=2, HN=3, ON=2, N=10000. Here are the results of the method at each step as applied to BP algorithm.

Step 1: The kernels of the BP algorithm were first identified as described in Table 4.2. Asymptotic analysis was then conducted to determine the frequency of execution for the primitive kernels as a function of the dataset properties. The frequency of the kernel operations were based on the equations that are executed in the BP. Table 4.3 shows the result of the asymptotic analysis for the primitive kernels. Further details on how we determined the order of these kernels is available in [5].

Table 4.3 - Results of Asymptotic Analysis for of BP primitive kernels

Primitive Kernels	Order
$\times$	$\theta(N \times [2 \times \mathrm{IN} \times \mathrm{HN} + 3 \times \mathrm{ON} \times \mathrm{HN} + 5 \times \mathrm{HN} + \mathrm{ON} + \mathrm{IN}])$
$+$	$\theta(N \times [2 \times \mathrm{HN} \times \mathrm{IN} + \mathrm{HN} + 3 \times \mathrm{HN} \times \mathrm{ON} + \mathrm{ON}])$
$\div$	$\theta(N \times \mathrm{HN})$
-	$\theta(N \times [\mathrm{ON} + \mathrm{HN}])$
Exp	$\theta(N \times \mathrm{HN})$

Counters were placed in the code to validate the asymptotic analysis. The resulting frequencies of BP primitive kernels are shown in Figure 4.1. We observe that counter simulation results are consistent with the asymptotic analysis in Table 4.3 which proves the correctness of our asymptotic analysis.

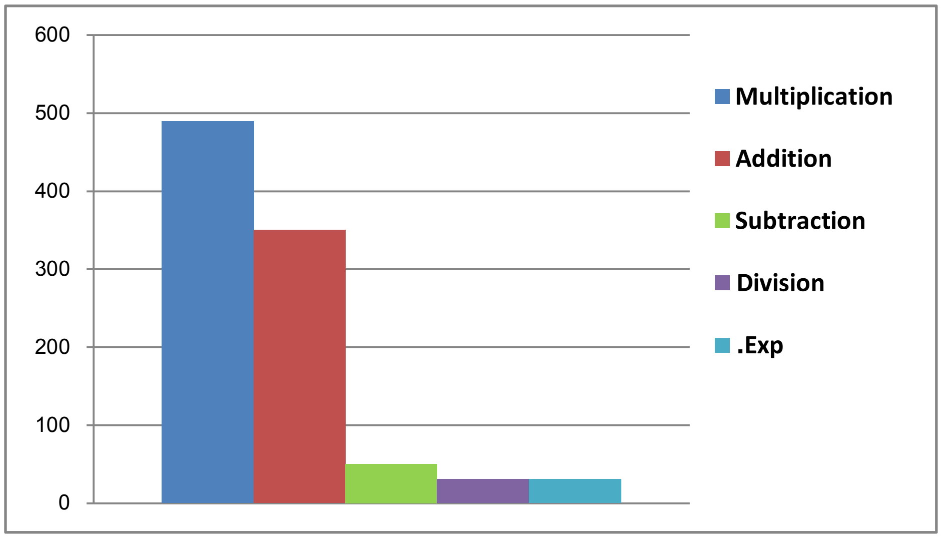

Figure 4.1 - Number of times BP primitive kernels are executed for IN=2, HN=3, ON=2 and N=10000.

The asymptotic analysis and the simulation for the studied BP NN consistently showed that the highest numbers of primitive kernels executed were multiplications followed by additions.

Step 2: For this step, the energy estimation method was applied per kernel as described in 3.2.1. The resulting energy costs are shown in Figure 4.2. The numbers were normalized (relative to subtraction) to show relative impact across kernels. Figure 4.2 shows that the exponential kernel has the highest energy cost.

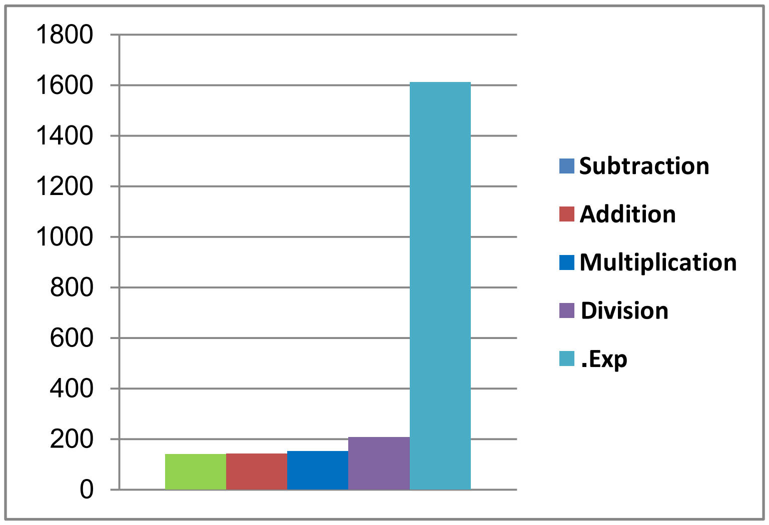

Figure 4.2 - WATTCH results for the energy cost of BP primitive kernels on RISC architecture

Step 3: For energy profiling, the overall energy was then calculated by multiplying the number of times each kernel is executed by the corresponding cost of each kernel. The overall energy of the algorithm based on Figure 4.1 and 4.2 was calculated as shown in equation (4).

$$overallenergy = 152.7\times 490 + 141.8 \times 350 + 141.1 \times 50 + 208.4 \times 30 + 1612.6\times 30 = 186138(\text{in thousands}) \quad (4)$$

Based on this calculation, we can determine the contribution of each kernel to the overall energy as shown in Figure 4.3.

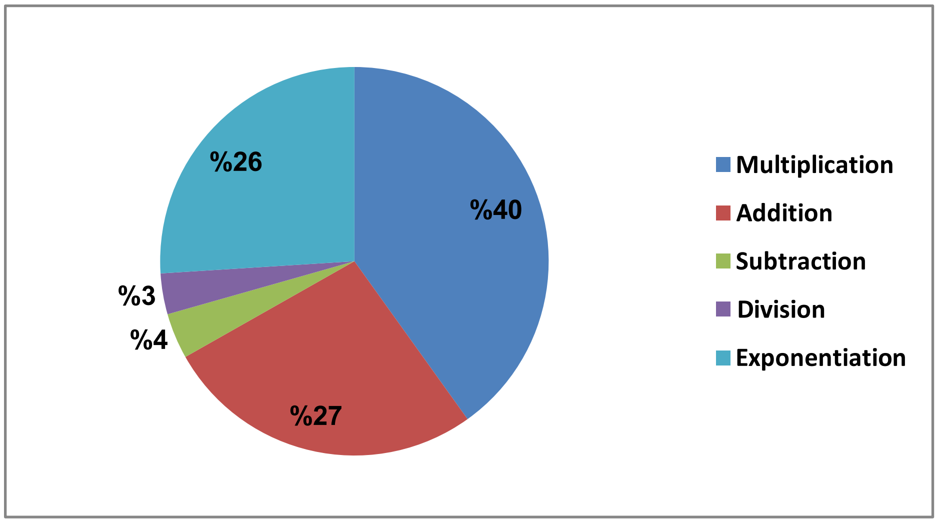

Figure 4.3 - Contribution of BP primitive kernels to the overall cost

The results show how the two proposed methods can collectively contribute to energy awareness and prioritization. It can be seen that although the exponential function was executed only a few times (as shown in Figure 4.1), it has a high contribution to the overall energy due to its high relative energy cost. It is also clear from Figure 4.3, that the multiplication kernel has the highest impact followed by addition and exponential. It is also noted that the energy contribution of the exponential kernel is higher than that of the division kernel despite the fact that there are more division computations than there are exponentials.

To further study the potential savings to the overall energy, we simulated the impact of reducing each kernel's energy cost by 25%. Figure 4.4 shows the overall energy cost of the algorithm as the number of iterations increases. Results confirm that the largest overall energy reduction is obtained by reducing the cost of multiplication followed by reducing the cost of the exponential function. Reducing in the cost of the other kernels (subtraction, division and addition) has smaller impact on energy reduction.

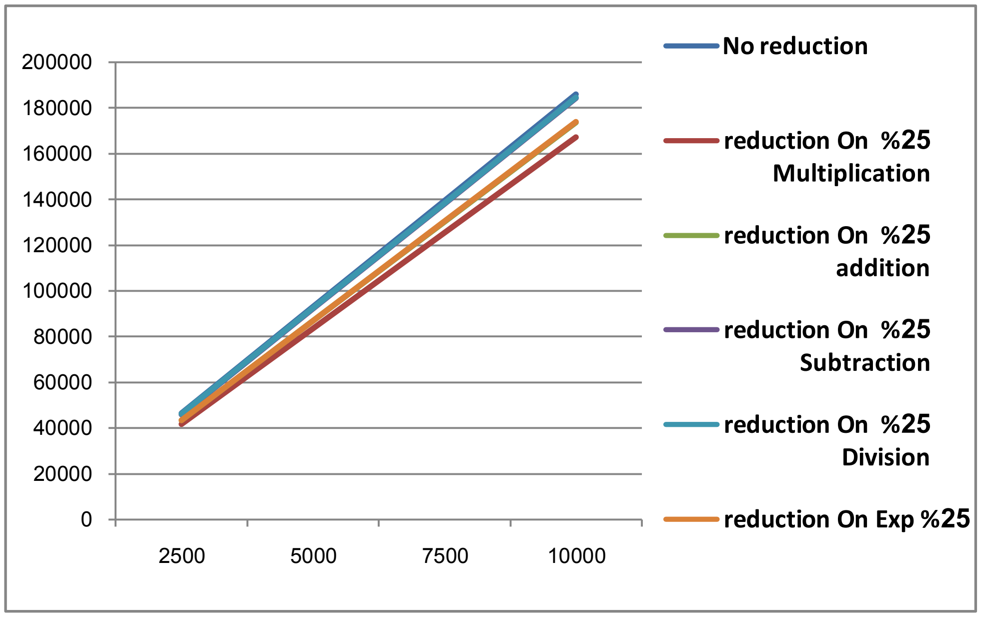

Figure 4.4 - The impact of reducing different BP primitive kernels by 25% on the overall algorithm for different iterations.

Step 4: The last step in the method consists of picking the most consuming kernels, reducing their energies, and then determining the overall impact. Although energy optimization is outside the scope of our work, we still wanted to illustrate the potentials of this step. We proposed two approaches for reducing the energy of BP algorithm. The first approach is based on using approximations to reduce energy but at the expense of accuracy. The second approach involves data pre-processing or transformation to enable faster convergence of the algorithm. These two approaches can be used for any algorithm, but we show their usefulness as applicable to BP.

Step 4A – Approximate LUT: From the results of step 3, exponential kernel was one of the top energy-consuming kernels. As a result, we considered alternatives to the high energy consuming exponential operation by using approximation techniques [34]. The basic idea is to pre-calculate the exponential function for the numbers in the range 0 to 1000 with a step of 1 and store the results in a Look-up Table (LUT). Then at each iteration of the training phase, instead of calculating the exponential function of the variables, the stored result is fetched from the LUT that is maintained in the cache. The LUT is considered an approximation technique since only a discrete number of values are calculated the exponential in the range [0, 1000] with a step of one. Therefore, the exponential function of (2.3) is approximated by the pre-calculated exponential value of (2 or 3) already stored in the LUT.

To evaluate the impact of using LUT as alternative to the exponential, the energy estimation method was used to determine the energy cost of fetching an element from LUT in addition to the overhead incurred once in populating the LUT. The energy cost of fetching an element from the LUT was 94.49% less than the cost of calculating the exponential function. This will reduce the energy of running the whole BP algorithm by 24.57% as the exponential function contributes 26% to the total energy of the algorithm. The energy cost of populating the LUT population was calculated based on the energy cost of (exp) kernel as shown in equation (5):

$$E_{LUT\_initial} = 1000 \times E_{exp} = 1000 \times 1612.62 \qquad (5)$$

Where E_{exp} is the cost of one exponential function operation based on Figure4.2.

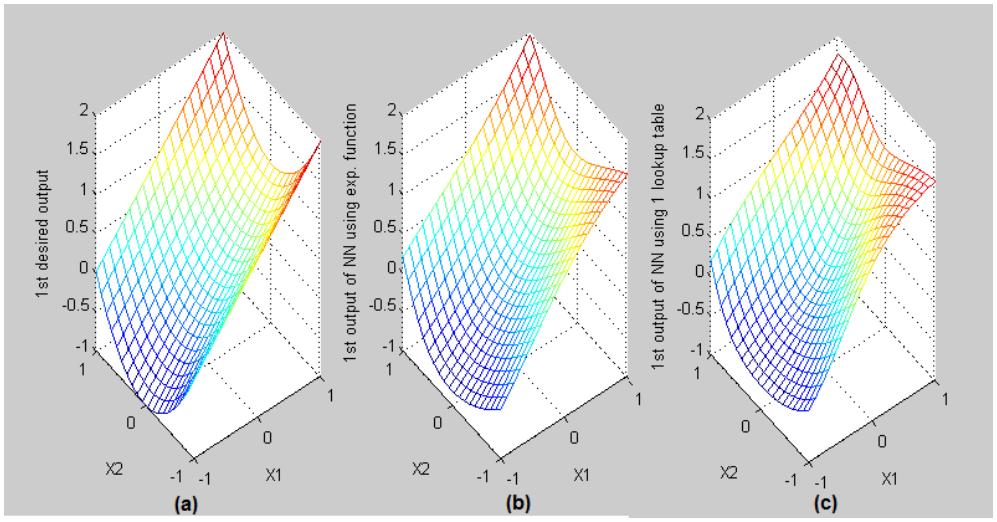

Figure 4.5 –A comparison between the first output and the approximated output (a) 1st desired output (b) 1st NN output using exp. (c) 1st NN output using lookup table approximation

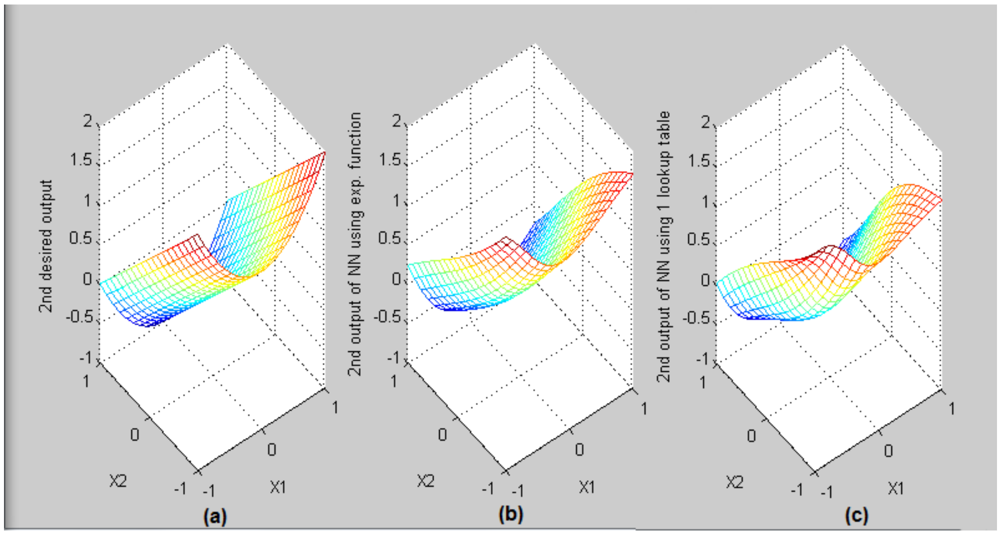

Figure 4.6 -A comparison between the second output and the approximated output (a) 2nd desired output (b) 2nd NN output using exp. (c) 2nd NN output using lookup tables approximation

The output of BP NN in figures 4.5 and 4.6 show the effect of using the LUT approximation versus the effect of the exponential function. Figure 4.5(a) represents the first desired output of NN. Figure 4.5(b) represents the first output of NN using normal exponentiation function and figure 4.5(c) represents the first output of NN using LUT. The same comparison is shown in Figure 4.6 for the second output of NN. Visually, it can be seen that the output of the NN with the LUT approximation technique is very close to the output of NN with no approximation, except for right upper edge of the hyper-plane.

The relative error in the approximation was also measured by taking difference between the two computed expoenential values= $|\exp(s) - \exp LUT(s)|$ and then divide it by exp(s) as shown in (6):

$$relative_{error\ (s)} = \frac{|exp(s) - \exp LUT(s)|}{|exp(s)|} \qquad (6)$$

The average of all relative errors was calculated to be 25.64%. This approach trades accuracy at the expense of energy. Significant energy savings can be gained if we tolerate this error.

Step 4B – Normalize for Faster Convergence: In the second approach for energy optimization, we examine ways that are specific to the algorithm beyond the general examination of kernels. In particular, we look at preprocessing the data to simplify algorithm computations for reducing energy impact.

Based o previous mathematical modeling of NN [3], the input values of the NN training phase can be normalized to the range of [0, 1] for faster convergence of the NN algorithm. To examine the impact of the normalization on energy efficiency, we examine the energy consumption of BP NN with and without data normalization. The energy cost of the normalization process is also added to the cost when normalizing.

Min-max Normalization was used to normalize the training set into [0, 1] as follows:

$$v' = \frac{v - min_a}{max_a - min_a}(new\_max_a - new\_min_a) + new_{min\ a} \qquad (7)$$

Where new_max=1 and new_min=0. It can be seen that normalization process requires two operations: subtraction and division. The relative costs of these operations were measured to be 141.1 and 208.4 respectively based on the relative energy costs from Figure 4.2. One subtraction operation and one division operation is performed for each attribute value in each training tuple. As a result, the total energy cost for normalization can be determined as shown in (8):

$$C_{nor} = (141.1 + 208.4) \times N_{tuples} \times N_{att} \qquad (8)$$

Where C_{nor} is the cost of normalization, N_{tuples} is the number of tuples in the training set, and N_{att} is the number of attributes in each tuple.

To demonstrate the savings in energy that are obtained by normalization, an experiment was conducted on a training set with 400 tuples. Each tuple contained two attributes with values between 0 and 2.8. The stopping criterion for the MSE was set to 0.8.

Table 4.4 shows the cost comparison for running the algorithm with and without normalization. It can be clearly seen that normalizing the training set has led to reducing the energy cost. There is an energy overhead for the normalization process. However, this overhead will be compensated as normalizing the training set before the learning phase makes the algorithm converge faster with fewer iterations, and hence resulting savings in energy.

Table 4.4 - A cost comparison with and without normalizing the training set

Without Normalization	With Normalization		
Overall Cost	Cost of Normalization	Training Cost	Overall Cost
6,775,400	279,600	2,773,500	3,053,100

Figure 4.7shows the number of iterations that was required to reach the stopping criterion with and without normalization. We can see that the larger the values of the input are, the more iterations were required to reach the MSE threshold.

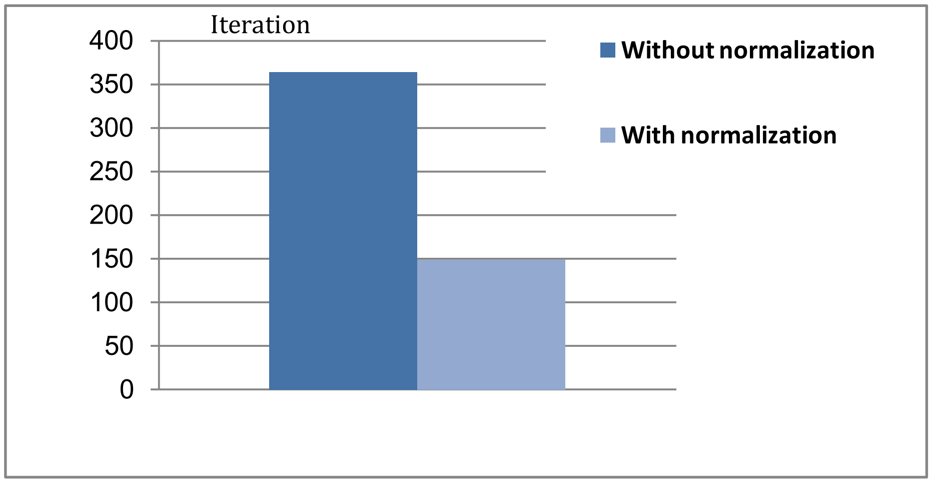

Figure 4.7 - Number of required iterations to reach MSE<0.8

4.4. Additional Case Studies with Considerations for Aggregate-level Kernels and Load/Store Energy

While the previous sub-section focused on assessing primitive kernels with the BP NN case study, this sub-section shows experiment results for aggregate kernels with KNN and LR case studies. The proposed methods are used to profile the energy contribution of the aggregate kernels in the algorithms. The measurements were conducted using instrumented boards that provided actual physical numbers rather than simulated evaluation using the setup described in section 4.1. We present our energy profiling results for KNN and LR algorithms and end this subsection with an analysis for the obtained results.

4.4.1.Energy-Aware Computing – KNN Case Study

This sub-section describes how the method can be used for other algorithms and derive an assessment of aggregate-level kernels. In the following experiments, physical measurements are collected using instrument boards for the CISC architecture instead of simulation tools in order to obtain energy estimates. Table 4.5 and Table 4.6 show the energy cost of primitive kernels and load and store operations based on the approach proposed in section 3.2. From Table 4.2, the identified aggregate kernels for KNN are normalization, Euclidean distance and comparisons.

As proposed in our four-step methodology explained in section 3.1, asymptotic analysis is first conducted to determine the frequency of the aggregate kernels. The results are shown in Table 4.7. N_{tuples} , and N_{features} are the number of tuples and the number of features in the dataset respectively.

For this experiment, 1st nearest neighbor ($K = 1$) is considered, with the dataset having $N_{\text{tuples}} = 150$, and $N_{\text{features}} = 5$. The resulting frequencies for the aggregate kernels are shown in Figure 4.8.

Table 4.5 - Energy cost of primitive kernels using physical measurements

	+	-	×	/	Square	Square Root	Log	Exp
Energy (10^{-10} Joule)	33.66	39.94	10.89	1058.62	64.78	2025.74	7347.46	49961.21

Table 4.6 - Energy cost of move operations (load/store) using physical measurements

Move Operation	Energy (10^{-10} Joule)
move value, stack	27.94
move register, stack	34.51
move stack, register	57.46
move value, register	22.96
move register, register	26.92

Table 4.7 – Order of KNN kernels

Aggregate Kernel	Order
Normalization	$N_{tuples} \times N_{features}$
Euclidean Distance	N_{tuples}
Compare	N_{tuples}

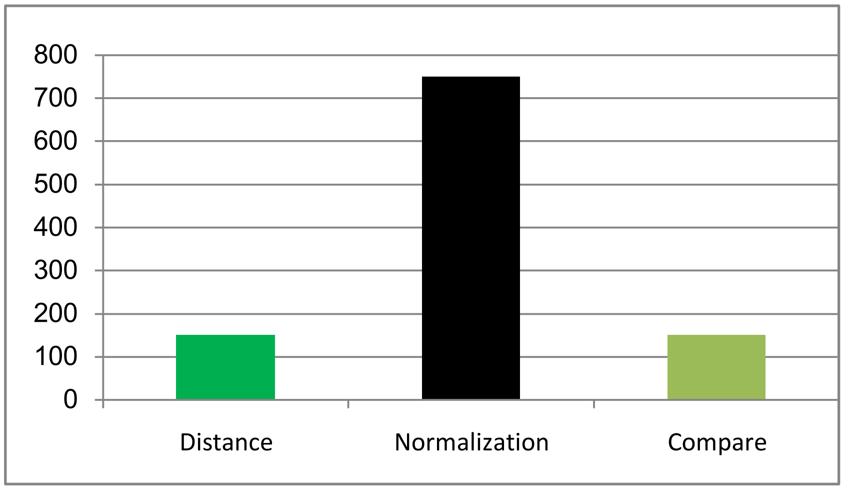

Figure 4.8 - Frequency of KNN aggregate kernels for the considered dataset.

The second step in the method consists of estimating the energy cost each aggregate kernel, which can be derived based on the energy cost of primitive kernels in Table 4.5 and by following the approach described in section 3.2.2. The resulting energy numbers for KNN aggregate kernels are shown in Figure 4.9.

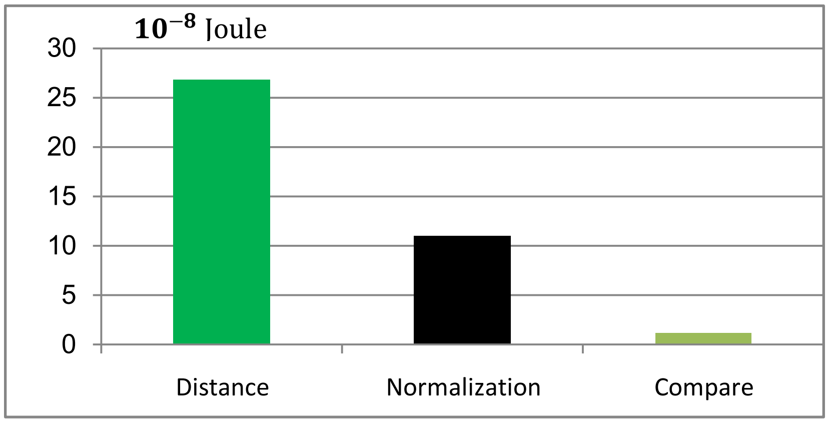

Figure 4.9 - Energy cost of KNN aggregate kernels for the considered dataset.

The third step consists of calculating the total energy of the algorithm, which was implemented in C++. The energy contribution of each aggregate kernel was calculated bymultiplying the frequency of kernels (Figure 4.8) by the energy cost of kernels (Figure 4.9).

Asymptotic analysis was also conducted to determine the frequencies of the load/store operations. Their energies were then estimated using the method proposed in3.2.3.

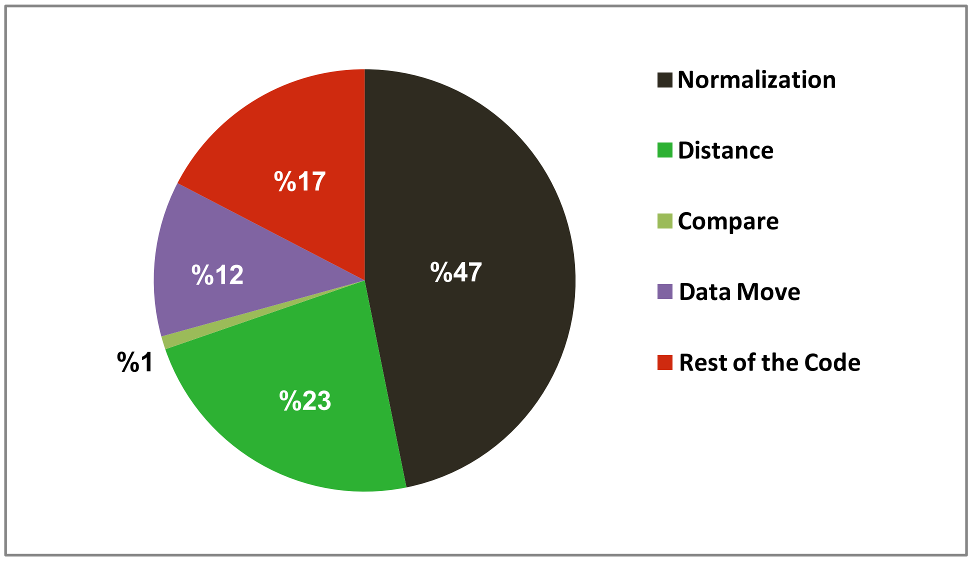

Figure 4.10 - Energy contribution of KNN aggregate kernels to the total energy.

Figure4.10 shows a pie chart for the energy distribution of the different aggregate kernels relative to the total energy. The chart provides interesting insight towards energy-aware computing. It can be seen that the normalization aggregate kernel has the highest energy contribution for the studied dataset. Although the Euclidean distance has higher cost than normalization (Figure 4.9), the frequency of normalization is higher than the distance (Figure 4.8) and thus the energy contribution of normalization is higher than the distance. The data move portion in Figure 4.10 represents the energy involved in the load and store operations in the algorithm. The rest of the code represents the energy involved in executing operations other than the aggregate kernels such as the overhead in the "for" loop.

Towards energy optimization, we also study the savings of the overall energy that can be achieved by reducing each kernel's cost by 25%. This helped in reflecting which kernels can be targeted for maximizing the reduction of energy. Figure 4.11 shows the overall energy cost of the algorithm after reducing each kernel's cost by 25%. Results indicate that the largest overall energy reduction is obtained by reducing the cost of normalization followed by Euclidean distance, which is consistent with the results of the pie chart in Figure 4.10. Figure 4.11 also gives an estimate of what would be the total energy of the algorithm when the energy of any of the kernels is reduced by 25%.

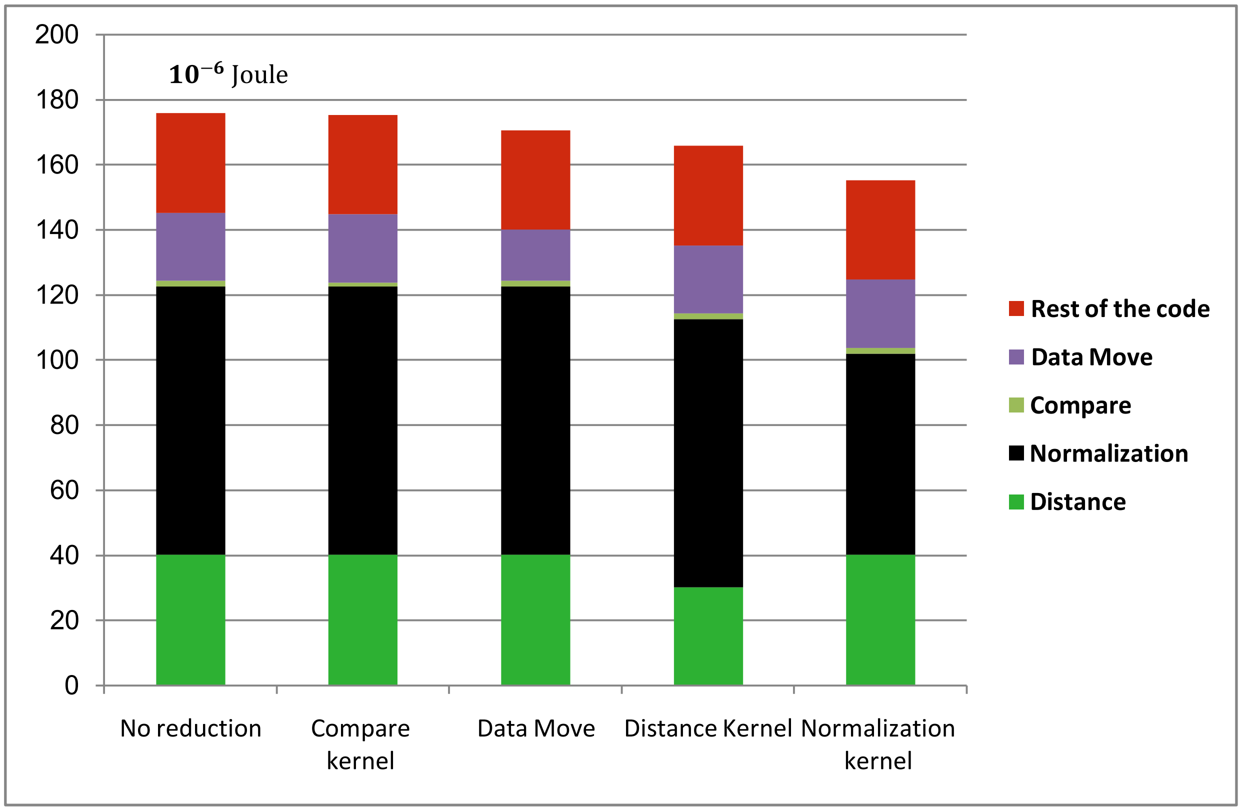

Figure 4.11 - Effect of reducing the energy of KNN aggregate kernels by 25%.

4.4.2.Energy-Aware Computing – LR Case Study

This sub-section covers an additional experiment on demonstrating the use of the method for energy profiling and developing energy-awareness for an algorithm, and to illustrate that the findings are algorithm-dependent. This experiment targeted LR. The LR kernels were determined in Table 4.2to be the average, the first regression coefficient (W_1), and the second regression coefficient (W_2). The algorithm was implemented using C++ code, and tested with a dataset that has $\mathrm{N_{tuples}} = 150$. The profiling results are shown in Figure 4.12, and it can be seen that a large percent of the algorithm's energy is spent in load and store operations. The findings indicate that significant energy is consumed by the memory and cache exchanges rather than by computations. This is due to the fact that LR does not involve intensive computational operations that have high energy cost such as exponential or square root and thus the energy cost of load and store operation constitutes a large percent of the total energy. Another insight from the results is that the first regression coefficient (W_1) has a high energy contribution as it involves many computations compared to average and to the second regression coefficient(W_2).

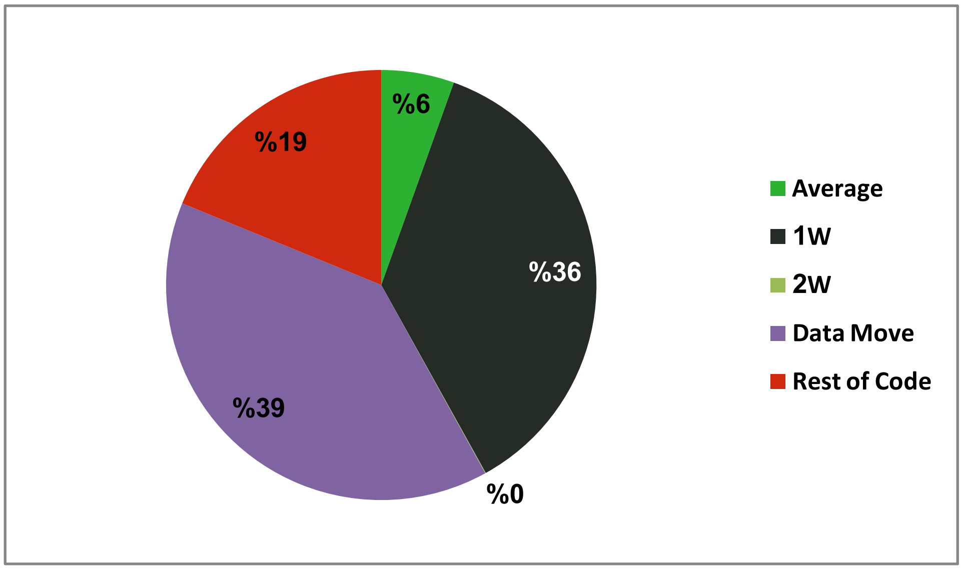

Figure 4.12 - Energy contribution of LR aggregate kernels to the total energy.

Here also, we examined the effect of reducing each kernel's cost by 25%. Figure 4.13shows the overall energy cost of the algorithm after reducing each kernel's cost by 25%. Results indicate that the largest overall energy reduction is obtained by reducing the cost of memory fetches, followed by reducing the energy cost of calculating the first regression coefficient. Figure 4.13 also gives an estimate of what would be the total energy of the algorithm after optimization.

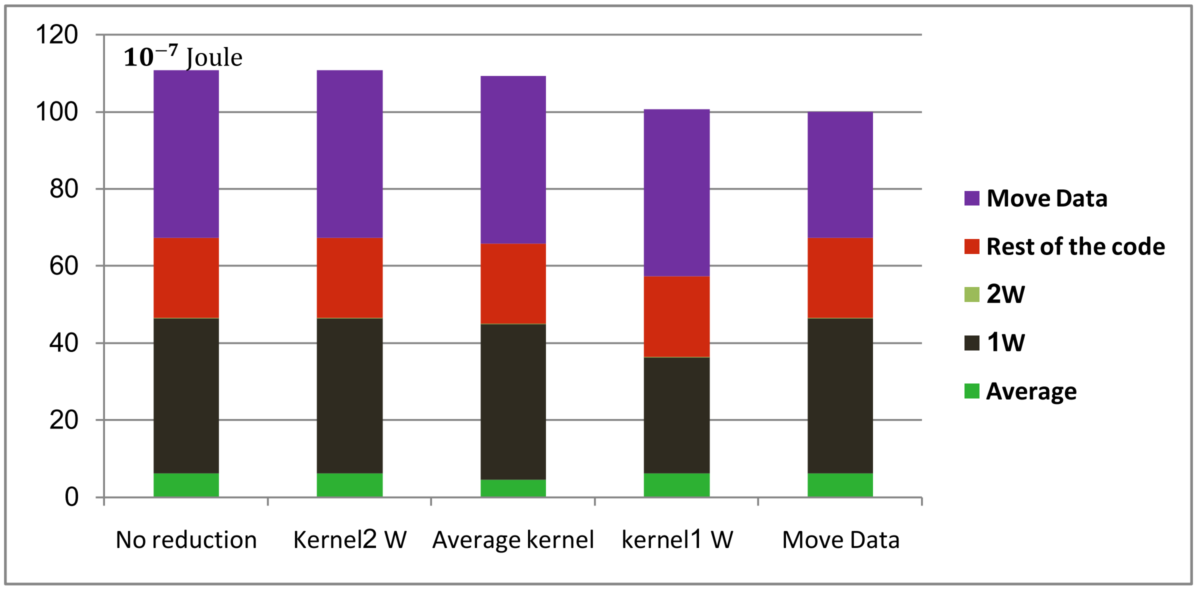

Figure 4.13 - Energy effect of reducing LR aggregate kernels by 25% on total energy.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Our work on energy profiling has introduced two methods for developing energy-aware algorithm. The first method is a four-step process for energy analysis and profiling with particular emphasis on kernel-based evaluation. The second method provides a methodology for kernel and load/store energy estimation. The energy estimation method shows how to get energy for primitive kernels based on assembly injection, and then derive energy for aggregate kernels from the primitive kernels.

Experiment results were conducted with accurate physical measurements instead of relying on simulation only. Specially instrumented boards were used for that purpose. Case studies were considered with three common data mining algorithms: BP NN, KNN and LR. The case studies show the successful use of the methods for energy profiling and developing energy awareness. The BP NN case study also showed two efficient approaches for energy optimization based on creating an approximate LUT for the kernel, or data preprocessing for faster convergence. In some cases, the energy saving came at the expense of accuracy. Furthermore, the experiments show that different algorithms have different kernels as top energy consumers. The experiments showed how to prioritize the highest cost kernels that can be targeted for energy optimization.

For future work, we plan to develop an automated tool that predicts the energy cost of any given code based on the energy cost of the primitive kernels that were already calculated in our work. We also plan to target the identified expensive kernels in the three profiled algorithms and find opportunities for energy optimization in the compiler layer. Our intention is to develop a compiler that focuses on transforming the codes of expensive kernels into more efficient assembly codes with lower energy price.

REFERENCES

[1] P. Kogge, “The Tops in Flops”, IEEE Spectrum magazine, Feb. 2011.

[2] C. Pettey, “Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emission,” *Gartner Press Release*, 26 April 2007. URL: http://www.gartner.com/it/page.jsp?id=503867.

[3] S. Haykin, “Neural Networks: A Comprehensive Foundation (2nd Edition),” *Prentice Hall*, July 1998.

[4] J.Han, M.Kamber, “Data Mining: Concepts and Techniques,” 2nd Edition, *Morgan Kaufmann*, January 2006.

[5] M. Dabbagh, H. Hajj, A. Chehab, W. El-Hajj, A. Kayssi and M. Mansour, "A design methodology for energy aware neural networks," *in Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, 2011, pp. 1333-1340.

[6] S. Daud, R. B. Ahmad and N. S. Murhty, "The effects of compiler optimizations on embedded system power consumption," *in Electronic Design*, 2008. ICED 2008. International Conference, 2008, pp. 1-6.

[7] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, pp. 94, 2000.

[8] G. Sinevriotis and T. Stouraitis, "A novel list-scheduling algorithm for the low-energy program execution," in Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium, 2002, pp. IV-97-IV-100 vol.4.

[9] S. Wiratunga and C. Gebotys, "Methodology for minimizing power with DSP code," *in Electrical and Computer Engineering*, 2000, pp. 293-296 vol.1.

[10] N. P. Desai, "A novel technique for orchestration of compiler optimization functions using branch and bound strategy," *in Advance Computing Conference, 2009. IACC 2009*, 2009, pp. 467-472.

[11] A. M. Malik, "Spatial based feature generation for machine learning based optimization compilation," *in Machine Learning and Applications (ICMLA), 2010 Ninth International Conference*, 2010, pp. 925-930.

[12] C. Dubach, T. M. Jones, E. V. Bonilla, G. Fursin and M. F. P. O'Boyle, "Portable compiler optimisation across embedded programs and microarchitectures using machine learning," *in Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium, 2009,* pp. 78-88.

[13] S. Hung, C. Tu, H. Lin and C. Chen, "An automatic compiler optimizations selection framework for embedded applications," *in Embedded Software and Systems, 2009. ICESS '09. International Conference, 2009*, pp. 381-387.

[14] K. D. Cooper, D. Subramanian, and L. Torczon, "Adaptive optimizing compilers for the 21st century," *in Proceedings of the 2001 Symposium of the Los Alamos Computer Science Institute*, October 2001.

[15] Y. Lu and G. De Micheli, "Comparing system level power management policies," *Design & Test of Computers, IEEE, vol. 18, pp. 10-19*, 2001.

[16] S. Albers, "Energy-Efficient Algorithms,"*Communications of the ACM, vol. 53, no. 5, pp. 86-96*, May 2010.

[17] Y. Lu and G. De Micheli, "Comparing System-Level Power Management Policies,"*IEEE Design & Test, v.18 n.2, p.10-19, March 2001*.

[18] B.P. John, A. Agrawal, B. Steigerwald, and E.B. John, "Impact of Operating System Behavior on Battery Life", *presented at J. Low Power Electronics*, 2010, pp.10-17.

[19] S. Kaxiras and M. Martonosi, "Computer Architecture Techniques for Power-Efficiency". *Morgan and Claypool*, 2008.

[20] P. K. Dutta, D. E. Culler, "System Software Techniques for Low-power Operation in Wireless Sensor Networks," *ICCAD '05 Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pp. 925- 932, 6-10 Nov. 2005.

[21] N. Abbas, H. Hajj and A. Yassine, "Optimal WiMAX frame packing for minimum energy consumption," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International,* 2011, pp. 1321-1326.

[22] T. H. Darwish, R. Chabukswar, "Intel Hardware Accelerated High Definition Video Playback Power Analysis," *Intel Software & Service Group,* 2009.

[23] S. Pawaskar, H. H. Ali, “A Dynamic Energy-aware Model for Scheduling Computationally Intensive Bioinformatics Applications,” *High Performance Computing and Simulation (HPCS),* 2010 pp.216-223, June 28 2010-July 2 2010.

[24] B. Steigerwald, R. Chabukswar, K. Krishnan, J. D. Vega, “Creating Energy – Efficient Software,” *Intel white paper*, 2008.

[25] P. Larson, “Energy-Efficient Software Guidelines,” *Intel Software Solution Group,* 2008.

[26] P. Unnikrishnan, G. Chen, M. Kandemir and D. R. Mudgett, "Dynamic compilation for energy adaptation," *in Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference,* 2002, pp. 158-163.

[27] D. Schall, V. Hudlet and T. Harder, “Enhancing Energy Efficiency of Database Applications Using SSDs”, *ACM Proceedings* 2010.

[28] R.Ge, X.Feng, S. Song, H.Chang, D. Li and K. W. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *Parallel and Distributed Systems, IEEE Transaction*, vol. 21, pp. 658-671, 2010.

[29] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, K. Olukotun, “Map-Reduce for Machine Learning on Multicore,” *Neural Information Processing Systems,* pp. 281—288, 2006.

[30] B. Negrevergne, A. Termier, J. Méhaut, T. Uno, “Discovering Closed Frequent Itemsets on Multicore: Parallelizing Computations and Optimizing Memory Accesses,” *High Performance Computing and Simulation (HPCS),* 2010 pp.521-528, June 28 2010-July 2 2010

[31] M. Marcu, D. Tudor, H. Moldovan, S. Fuicu and M. Popa, "Energy characterization of mobile devices and applications using power–thermal benchmarks," *Microelectron. Journal, vol. 40, pp. 1141-1153, 7,* 2009.

[32] C. Hu, D. A. Jimenez and U. Kremer, "Toward an evaluation infrastructure for power and energy optimizations,"*19th IEEE International Symposium on Parallel and Distributed Processing*, 2005.

[33] D. Burger , T. Austin, “The SimpleScalar tool set, version 2.0”, *ACM SIGARCH Computer Architecture News, v.25 n.3, p.13-25,* 1997.

[34] A. Yamamoto, "Computational Efficiencies of Approximated Exponential Functions for Transport Calculations of the Characteristics Method," *Annals of Nuclear Energy,* 31(9):1027- 1037, June 2004.

[35] F. Hamady, A.Chehab and A. Kayssi, "Energy Consumption Breakdown of a Modern Mobile Platform under Various Workloads", International Conference on Energy Aware Computing (ICEAC), November 30–December 2, 2011, Istanbul, Turkey.

[36] H. Kothuru, G. Virupakshaiah, S. Jadhav, "Component-wise Energy Breakdown of Laptop," *in Proceedings of the 6th Annual GRASP Symposium, Wichita State University, 2010.*

[37] A. Mahesri and V. Vardhan, "Power Consumption Breakdown on a Modern Laptop," *in Proceedings Workshop on Power Aware Computing Systems, December 2004.*

[38] Chinn, Desai, DiStefano, Ravichandran, and Thakkar, "Mobile PC Platforms Enabled with Intel Centrino," *Intel Technology Journal, May 2003.*

[39] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," SIGMETRICS Perform Eval. Rev., 36(2):26–31, 2008.

[40] Fluke 2680 Series Data Acquisition Systems, http://us.flukecal.com/products/data-acquisition-and-testinstruments/data-acquisition/2680-series-data-acquisitionsystems.