An Approach to Measuring Kernel Energy in Software Applications

Mehiar Dabbagh[1], Hazem Hajj[1], Wassim El Hajj[2]
American University of Beirut
[1]Department of Electrical and Computer Engineering
[2]Department of Computer Science
E-mails: {mmd43, hh63, we07}@aub.edu.lb

Abstract- The large widespread of mobile devices and the increasing demand for high performance has made energy an important constraint in computer's technology. In this paper, we propose a simple software methodology that can be adapted by researchers to estimate the energy cost of kernels. Kernels are operations that can be implemented in hardware and are executed frequently in algorithms. This energy estimation helps in exploiting opportunities for energy optimizations in various algorithms. As a case study, we apply our methodology on back propagation (BP) algorithm and we estimate the energy cost of its kernels on two different architectures using: a) Simulation tools for the RISC architecture b) Physical measurement of the current and voltage using special boards for the CISC architecture. Our experiments show that the produced energy costs using our method have 5.14% error with respect to the costs of these kernels in real codes, which proves the high accuracy of our methodology. The results also show that different architectures exhibit different energy consumption when executing the same kernel which proves that alternative kernel implementations have impact on energy saving. Based on the estimated energy costs of the kernels, we further propose an energy optimization technique by using lookup tables (LUT). The proposed technique targets the exponential kernel because of its high energy consumption and replaces it with a LUT. The experimental results show a significant reduction of 99.97% in the energy consumed by the exponential kernel.

Keywords: Energy Aware; Back Propagation Algorithm.

I. INTRODUCTION

The large widespread of mobile devices has made energy an increasing importance in today's technology. Mobile devices are battery dependent and are required to stay charged for a long time, while battery technology is developing at a slow pace. Energy is an issue that is not only limited to mobile devices. In fact it has become a major constraint, named the power wall [1], in improving computer's performance. Towards the goal of reducing energy, the different computer layers, including application, compiler, operating system (OS), computer architecture, and OS lower-level circuit techniques, must all play a major role in creating power-efficient computer systems.

We present in this paper a methodology to estimate the energy cost of the kernel operations. A kernel is defined as an operation that can be implemented in hardware, and is executed frequently in the algorithm. Our work helps in exploring possible optimization in the application layer. Take for example data mining algorithms; if we are able to achieve a small saving in one of the operations that are executed repeatedly in each iteration (for instance, division), then we end up with a high energy saving of executing the whole algorithm. In fact, the energy cost of the kernel operations along with the asymptotic analysis of the studied algorithm help in knowing the energy contribution of each kernel to the overall energy of the algorithm. Based on knowing the energy contribution of each kernel, the kernels with the high contribution become our target for energy optimization since they have the highest energy reduction on running the whole algorithm.

Given a certain algorithm, for example Back Propagation Neural Network (BP) which we analyze in this paper, our work aims to do the following:

1. identify the kernels in the BP algorithm i.e. the operations that are executed the most
2. estimate, with high accuracy, the energy cost of each kernel
3. reduce the energy cost of the whole algorithm by reducing the energy cost of the kernels that consume the most energy.

In this paper, we suggest solutions for the second and third stages. The first stage was addressed in [2]. To address the second stage, we designed a simple yet accurate algorithm that estimates the energy cost of a given kernel. The kernel can be a simple operation such as addition or multiplication or a complex method composed of many operations. Our methods provides a mechanism to estimate the energy cost of any algorithm's kernel by only updating a set of variables that correspond to the range of the input data i.e. the min and max values in the input data. For example, to measure the addition (+) kernel in BP, all we need to do is to change these variables according to the BP training data. To measure the kernels of another algorithm such as Support Vector Machine (SVM), again the same variables of our method can be changed based on the [min, max] values of the SVM training data. Our method produces the time needed to execute the kernel. Another mechanism is still needed to calculate the power consumed by the CPU during this time. Multiplying the resultant time and power values produces the CPU energy consumed by the kernel. To calculate the CPU power consumption, we used two methods:

1. Simulation using SimpleScalar and WATTCH tools.
2. Physical measurement of the current and voltage of the major components of the computer architecture using special boards and acquisition system.

The major difference between these two approaches is that the simulation is based on the RISC architecture, whereas the physical measurements are based on the CISC architecture.

To test the accuracy of our proposed method, we considered the BP algorithm as a case study. We started by identifying its kernels, which are: addition, multiplication, subtraction, division, and exponential. Then, we ran the BP algorithm directly on the physical board (CISC architecture) and calculated the energy cost of these kernels. Hence, these costs present the exact kernels' cost. Finally, we ran our proposed method on the same physical board and calculated the energy cost of every kernel. We found out that our proposed method produces energy costs with only 5.14% error with respect to the costs of the kernels in codes of the algorithm. Such

978-1-4673-0465-8/11/$26.00 ©2011 IEEE

experiment was conducted to highlight the accuracy of our methodology.

Having proved the accuracy of our method using accurate physical measurements, any researcher can now adapt our method to estimate the energy cost of the kernels of any algorithm. If the user does not have the physical equipment, any simulation tool can be used, was it a tool that simulates a CISC architecture or a RISC one.

In fact, by applying our method on both CISC and RISC architectures, one can decide quickly on which architecture to use based on the produced energy costs. As will be seen later in section IV, we found out that the RISC architecture has lower energy costs for certain kernels compared to the CISC architecture. We also study how the different ranges of the numbers for the kernel operation affect its energy cost. An important consequence of this finding is that it provides evidence to support assumptions made in previous work [2] when exploring alternative kernel implementations for energy saving.

Finally and since the exponential operation in BP consumed a lot of energy, we propose to use lookup tables (LUT) instead of calculating the exponential kernel. Every exponential operation was replaced with fetching an element from a preprocessed LUT. An exceptional 99.97% energy saving was achieved to the exponential kernel when a LUT is used.

The rest of the paper is organized as follows: In section II, we summarize the research work that has been done to optimize energy in the different computer layers. In section III, we introduce our methodology to estimate the energy cost of the kernel operations of any algorithm. In section IV, we implement our methodology on back propagation NN algorithm and we use simulation and physical tools to estimate these costs for the RISC and CISC architecture. We prove in this section the accuracy of our method and show the savings of energy that can be achieved by using lookup tables. Finally in section V, we conclude the paper and present our future work.

II. RELATED WORK

We summarize in this section the previous research that was done to optimize energy in the different layers,

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

In the compiler layer, compiler optimization techniques are usually used to improve performance by reducing the number of cycles that are required to execute a program. These techniques can be also used to save energy. In [3], [4] [5] and [6], different compiler optimization techniques (e.g. loop unrolling, instruction rescheduling) were applied on benchmarks in order to save energy. A main limitation of these techniques is that their effect on both performance and energy varies from code to another. Therefore, heuristic methods are used to determine the combination of techniques with the highest energy saving/highest performance for a certain code. An attempt to overcome this was presented in [7] and [8], where the authors used machine learning and genetic algorithms to predict the best combination and the best sequence of compiler optimizations such that the energy or the delay for running a given code are 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 they 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 the energy required to wake the device up is high. Therefore, the device is switched into a lower power state if only we predict that it will remain idle for a long time that is enough to compensate for the transition energy. Additional information about how these predictive techniques and algorithms work is provided in [9] and [10]. A comparison between the power policies in windows 7 and windows vista was presented in [11]. 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 which the operating voltage and frequency for executing a task are reduced such that the task meets its deadline with the lowest possible frequency and voltage, which leads into a great saving of energy but affects performance [12].

In the application layer, the optimization methods that were proposed can be classified according to [13] into three main categories: computational efficiency, contextual awareness and data efficiency.

The basic idea behind computational efficiency is using the maximum available performance for the different devices so that the work is done early, which allows us to switch the devices into the idle mode to save energy. Examples of computational efficiency techniques include using parallel programming to reduce the execution time or any other technique that enhances performance. The authors in [14] implemented a framework called PowerPack for profiling the energy of different applications. The authors used parallel programming and examined the number of required parallel nodes such that the overall energy for running different benchmarks is minimized.

In contextual awareness, applications change their behavior based on the available energy. A dynamic compilation that adapts battery changes is proposed in [15], where precompiled parts of the code that have low energy are used when the battery is low.

In data efficiency techniques, data is stored, accessed, and transferred in an energy efficient way. For instance, in [16] the authors showed that considerable energy savings can be achieved by using SSD instead of HDD as a storage device.

Although the work that attempts to optimize energy consumption is very important, estimating the consumed energy is as important. All the previously mentioned work used one of two ways to estimate the energy required to run a program on a given architecture:

a. Physically measuring the current, frequency and voltage of the different components using ammeters and special acquisition systems. This method has high accuracy but is expensive since it requires special equipments.
b. Simulating the behavior of running the program on the architecture: this method is inexpensive and it allows us to analyze the performance and power behavior with a cycle level of granularity for the different components. However, it is less accurate than the physical measurements but it is a good solution when the special equipments and boards of a certain architecture aren't available.

Researchers have also proposed different frameworks for providing energy and thermal profiles [17] or to reduce the time that is required to estimate the energy cost of a long code[18]. But none of the previous work, to the best of our knowledge, has targeted estimating the energy of the kernels for the algorithms.

In this paper, we propose a methodology for estimating the energy cost of kernels for any algorithm. Kernels are operations implemented in hardware and are executed frequently in algorithms. This energy estimation helps in exploiting opportunities for energy optimization for the different algorithms in the

application layer. We apply this measuring methodology on back propagation algorithm and show its accuracy when run on a CISC architecture. We also analyzed the method when a RISC architecture is used. Hence, the choice of which architecture to use is very important in saving energy as will be shown in section IV. Moreover, we propose an energy optimization technique using lookup tables and calculate the energy savings that can be achieved.

III. METHODOLOGY TO ESTIMATE THE ENERGY OF THE KERNELS

The first step that we need to do is determining the kernels of the studied algorithm.

Once we determine the kernels of the algorithm, we need to know which kernel we should target in order to achieve the highest saving of energy. Asymptotic analysis here is needed in order to determine how frequent each kernel is executed based on the studied algorithm and the dataset properties. Asymptotic analysis i.e. the number of times a kernel is executed, along with the corresponding kernel energy cost help in determining possible targets for energy optimizations. Further details about how to determine the kernels of the algorithms and how to know which kernel we should target for the highest energy optimizations are provided in our previous work [2].

Our focus in this paper is to find an accurate yet efficient way to measure the energy consumed by the kernels. The main idea behind our methodology is running a code in which the kernel operation is executed a number of times N and estimating its energy cost. Since the code includes not only the kernel operation but also the initialization of the variables, we remove the kernel operation from the previous code and we estimate its energy cost without the kernel operation, then we subtract the energy cost of the code without the kernel from the code with the kernel and divide the obtained result by N so that we end up with getting the energy cost of executing one kernel operation.

The pseudo code shown in figure1 is used to determine the energy of the kernel operations that are executed on numbers in the range between Min_range and Max_range. For example if we are trying to measure the energy cost of the exponential function kernel, exp(x), this energy cost is dependent on the value of x. If we know that x in the algorithm is between 0 and 1000, then we need to estimate the energy cost of calculating the exponential function for the numbers in the range [0,1000] where Min_range=0 and Max_range=1000. We declare variable x as an array (line 4) and store in it the numbers between 0 and 1000 inclusive. Since there are infinite numbers in that range, we use a variable "step" to control the granularity between consecutive numbers. For example, if step is equal to 1 (difference between each two consecutive values is 1), then we calculate the energy cost of calculating the exponential function of the numbers (0, 1, 2…., 1000).

```
1)   Print time;
2)   int array_size = (Max_range - Min_range) / step + 1;
3)   //Declare the variables for a certain range
4)   float x[array_size]; float y[array_size];
5)   //Give the variables values in the studied range
6)   For(int i=0; i<array_size;i++) {
7)        x[i]=Min_range + i*step;
8)        y]i]=Min_range + i*step;  }
9)   For (int k=0; k<10000;k++)
10)       For (i=0; i<array_size;i++)
11)            For (j=0;j<array_size;j++)
12)                 Kernel_operation (x[i] ,y[j]);
13)  Print time;
```

Figure 1. the code that is used to determine the energy cost of the kernels.

Lines 1 and 13 are used to print the time at the beginning and at the end of the code respectively. This will help us in knowing the execution time of the code. In line 4 we declare the variables that will be needed to execute the kernel operation. Here we only declared two variables x and y. There could be more depending on the kernel. The variables are declared as an array where the size of the array (array_size) is equal to the range of the variables that are being studied. In lines 6, 7 and 8, the declared variables are assigned numerical values. In line 12 the kernel operation is executed for all the possible combinations of the values in the studied range using the "for" loops in lines 10 and 11. The "for" loop in line 9 is necessary when measuring the voltage and current of the different components physically because without it, executing the code takes a very short time that is not sufficient to collect power measurements. The larger the number of iterations, the more accurate the results but the more time is required for measuring.

The code in figure 1 was executed on RISC and CISC architectures. For physical measurements, the average CPU power of running the code was estimated by physically measuring the current and voltage of the major components on the board of the studied architecture. The CPU energy cost of executing the code in figure 1 was determined by multiplying the average power consumed by the execution time; refer to this energy as E1.

Since our objective is to determine the energy cost of the kernel operation, we need to exclude the cost of the lines of the code other than the kernel operation. So we run the same code in figure1 after substituting line 12 by ";" and we estimate its energy (refer to it as E2). Now the energy cost of one kernel operation can be easily calculated by subtracting E2 from E1 and dividing it by the number of times the kernel operation was executed, refer to it as N. N is dependent on the number of iterations of the "for" loops in lines 9, 10 and 11.

IV. EXPERIMENTS AND RESULTS

We implemented our methodology to estimate the kernels' cost of the back propagation (BP) algorithm. We chose to study this algorithm as an example of a computationally intensive algorithm that is widely used in many domains and as an extension to our previous work where BP was analyzed using relative energy costs. In the first part of this section, we prove the accuracy of our methodology by comparing the results obtained from our methodology to the costs of the kernels in real codes of the algorithms. Then we analyze the effect of using different architectures. To be more detailed: (1) we use the physical board (CISC architecture) and apply our methodology to estimate the energy consumption of the BP kernels. We then compare the kernels'costs obtained by our methodology to the costs of these kernels in a real BP code to get a sense of the method's accuracy. (2) We use Simplescalar and WATTCH simulation tools to represent a different (RISC) architecture. Our aim in this experiment is to analyze the kernels' cost when different architectures are used. In the second part of this section we show how the energy cost changes when executing the kernel operation for different data values. The objective of this study is to show that the energy cost of executing the kernel operations is dependent on the range of the numbers. We show how these costs vary as the range of numbers increases. Finally in the last part of this section, we propose an optimization technique for saving energy using lookup tables and estimate the energy savings that can be achieved using this technique.

1. Energy cost of BP kernels

By observing the most frequent operations that are executed

repeatedly in the training phase, we determine the kernels of BP algorithm which are: addition, subtraction, multiplication, division and exponential function.

For the BP kernels, we estimate the energy cost of the kernels in the range [0, 1000].Therefore Max_range in the code of figure 1 is substituted by 1000. This range was chosen based on the values of the BP training data where all the kernels operations are executed on numbers in that range.

We also substitute line 12 in the code of figure 1 by the kernel operation. For example, to estimate the energy of the addition kernel we substitute line 12 by "x+y;" similar substitutions can be done for the other kernels.

a) Physical Measurements

We implemented our methodology to estimate the energy of BP kernels physically on the CISC architecture. 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. The current and voltage were collected for the major components of the board: CPU, memory, and total power of the platform. PACS program is installed on the monitoring computer in order to show how the collected measurements are varying over time and to calculate the average and peak power of the different components of the system over a period of time.

Windows 7 is installed on the studied platform. We fix the power plan on the balanced mode for all the experiments (a power plan is a collection of system and hardware settings that manage how power is consumed in the computer). While measuring the cost of the studied codes, we stop any unnecessary program using the task manager and keep only the basic processes that the O.S. requires.

TABLE 1
Estimated Energy costs for BP kernels

Code	Platform average power (Watt)	CPU average power (Watt)	Time (Secs)	Platform energy (Joule)	CPU energy (Joule)
No operation	23.555	15.249	21	494.662	320.231
+	25.019	16.612	23	575.435	382.082
-	25.292	16.835	23	581.717	387.215
×	24.911	16.546	22	548.049	364.0158
÷	22.541	14.602	71	1600.400	1036.749
Exp	24.891	16.224	2029	50502.989	32919.153

We also change the configurations in the power management by preventing any device from turning into a lower power state after a period of idleness. All of these changes are made in order to make sure that the conditions are the same when we estimate the energy cost of the different kernels.

Table 1 shows the obtained physical measurements. Each row contains the collected measurements for codes of figure 1 after substituting line 12 with the appropriate kernel operation. The "no operation" row represents the code of figure 1 after substituting line 12 by ";". This is used to determine the energy cost of the program without the kernel operation. The second column shows the average power of the whole platform for executing the codes. The whole platform average power contains all the platform components in addition to the communication between these components. The third column represents the average power of the CPU for executing the codes. The fourth column in table 1 represents the time that was required to execute the code and is determined by the "print time" commands in figure1 at the beginning and end of the code. The fifth and sixth lines represent the platform energy and the CPU energy respectively and are obtained by multiplying the average power by the time.

We estimate the energy cost of each kernel by subtracting the energy cost of the "no operation" code from the energy cost of the kernel codes, then we divide the obtained result by (10^{10}) which is the number of times the kernel operation was executed. The Platform energy costs and the CPU energy costs of executing one kernel operation are shown in table 2.

TABLE 2
Platform and CPU energy costs of BP kernels using physical measurements.

	×	+	-	÷	**Exp**
Platform Energy $(\times 10^{-10}$joule)	53.39	80.77	87.06	1105.74	50008.33
CPU Energy $(\times 10^{-10}$joule)	43.79	61.85	66.98	716.52	32598.92

To examine the accuracy of our measurements, we run the code of the BP algorithm and estimated its energy by calculating the time required to execute the code and by collecting the average power using the special boards. Then we estimated the energy cost of this code after removing one addition operation that was executed on numbers in the range between 0 and 1000. We calculated the energy cost of the addition kernel by subtracting the energy of the BP code without the addition operation from the energy of the original BP code. Results showed that the energy cost of the addition kernel in the BP code was 80.804×10^{-10} which was very close to the estimated cost of the addition kernel using our proposed methodology 80.77×10^{-10} . We did the same experiments for the other kernels and the results in Table 3 showed that the energy cost of the kernels in the BP code were very close to the estimated costs using our methodology. These results proved that our methodology estimates well the energy of the kernels in the algorithms' codes. Based on Table 3, the estimated energy cost of the kernels using the method had a maximum error of 5.14%.

TABLE 3
Comparison between the platform energy costs of real BP code and the estimated energy costs using our proposed methodology.

	×	+	-	÷	Exp
Real BP code $(\times 10^{-10}$joule)	56.28	80.81	91.48	1148.12	49284.41
Our methodology $(\times 10^{-10}$joule)	53.39	80.77	87.06	1105.74	50008.33

b) Impact of alternative implementations for the same kernels on algorithms energy

We also ran these codes on Simplescalar and WATTCH simulation tools. Simplescalar simulates the execution of the code on the RISC architecture and WATTCH is used to give relative energy numbers that determine the energy cost of the different units of the architecture when this code is executed. In our experiments, we used the default configurations of Simplescalar and WATTCH simulation tools. Details of these configurations are found in [19]. For the BP kernel the relative CPU energy costs that were obtained by simulation after using the methodology (explained in section III)

are shown in figure 2 where the y-axis represents relative numbers obtained by WATTCH for the CPU energy of executing one kernel operation. We notice in figure 2 that the exponential kernel is the one with the highest CPU energy cost.

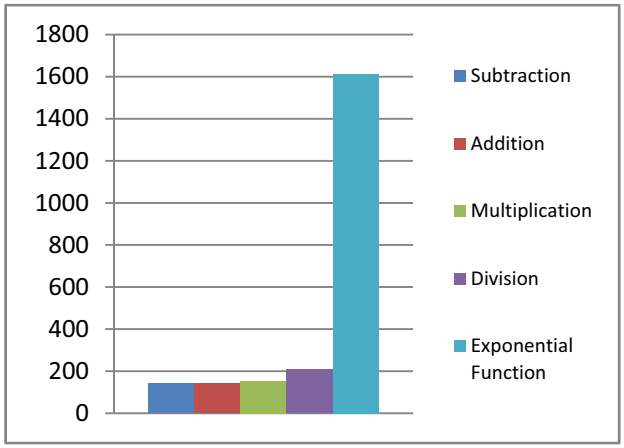

Figure 2. WATTCH results for the CPU energy cost of BP kernels on RISC architecture where the y axis shows the CPU relative energy costs of executing one kernel operation obtained by WATTCH for the RISC architecture.

We normalized the CPU simulation and the physical measurements by choosing the multiplication cost as a reference. This helped us understand the relative costs of the kernels. Table 4 shows the normalized simulation results and physical measurements. We don't expect to have similar results in the simulation and physical measurements since the simulation and physical measurements were performed on two different architectures. A RISC-based architecture was used for the simulator, and a CISC-based architecture was used for the physical measurements.

TABLE 4
Normalized simulation and physical measurements for the BP kernels.

	×	+	-	÷	Exp
Physical measurements	1	1.413	1.530	16.365	744.527
Simulation measurements	1	0.928	0.924	1.364	10.560

We note that the addition, subtraction and multiplication costs were relatively similar in both physical and simulation measurements. Hence, they have similar costs for the RISC and CISC architectures. However, the physical measurements showed that the exponential and division kernels had a much higher cost than simulation. This suggests that the CISC-based architecture had a higher cost for the division and exponential function kernels compared to the exponential and division in the RISC-based architecture. We conclude that if these two kernels were very common in the algorithm, then using the RISC-based architecture would have lower energy price compared to the CISC-based architecture used in this experiment. It is worth noting that other CISC-based architectures can consume much less energy. The important conclusion is not the fact that in this experiment the CISC-based platform performed worse, but rather that different kernel implementations on two different architectures yielded substantial differences. This conclusion opens doors for further research on kernel optimizations using different architectures.

2. The energy cost of the kernel operation for different ranges

In this part we investigated how the energy cost changes when we execute the kernel operation on different data ranges. We study the addition kernel as an example. We applied the methodology to estimate the platform and CPU energy costs for executing one addition operation for the different ranges.

The results in figure 3 showed that the energy cost of the kernel operation is dependent on the range of the numbers. As the range of numbers increases, the cost of addition tends to increase with some fluctuations. This conclusion provides another opportunity for further research on how the different ranges affect the cost for the different kernels and explain the reasons for the fluctuations.

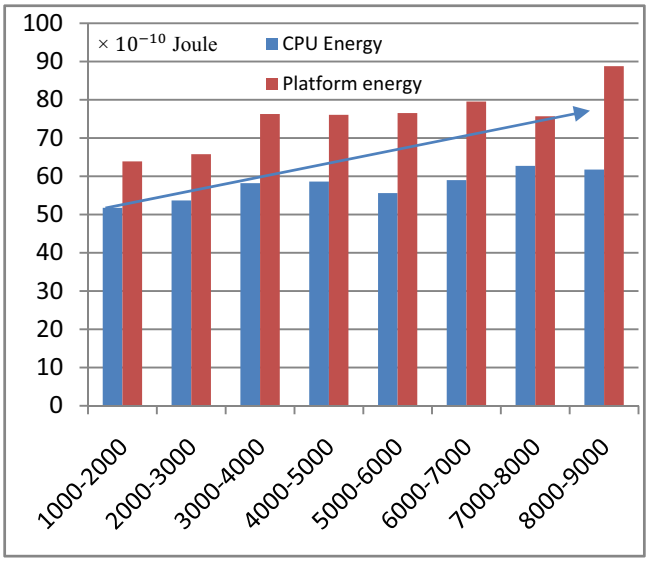

Figure 3. Platform and CPU energy of the addition kernel for different ranges. The arrow shows the tendency for the increase in energy as the range increses.

3. An optimization technique to save energy

Since our physical and simulation experiments showed that the exponential kernel has a high energy price, we propose an energy optimization technique using lookup table (LUT).

The basic idea behind our optimization technique is to calculate the exponential function of the numbers in the range 0 to 1000 with a step of 1 and store them in a LUT. Now in each iteration of the training phase (usually the number of iterations of BP algorithm is larger than 5000), instead of calculating the exponential function of the variables, we fetch the stored result from LUT that is maintained in the cache.

In order to evaluate our proposed method, we need to determine the energy cost of calculating the exponential function of the numbers in the range [0, 1000] and then compare the energy cost of fetching an element from LUT with the energy cost of calculating the exponential function.

The platform energy cost of calculating the exponential function of the numbers in the range [0, 1000] with a step of 1(refer to it as $E_{LUT\_initial}$) can be calculated based on the platform energy cost of (exp) kernel that was determined using our physical measurement as shown in equation (1):

$$E_{LUT\_initial} = 1000 \times E_{exp} = 1000 \times 50008.33 \times 10^{-10} \text{joule} \quad (1)$$

Where E_{exp} is the platform energy cost of one exponential function operation (Table 2). Similarly the CPU energy of calculating the exponential function of the numbers in the range [0,

1000] can be calculated by multiplying the CPU energy cost of executing one exponential kernel in Table 2 by 1000.

To estimate the energy cost of fetching an element from LUT we measure the energy cost of executing the pseudo code shown in figure 4 (refer to it as E1). Then we estimate the energy cost of executing the code without line 8 (refer to it as E2). The energy cost of fetching an element from LUT can be determined by subtracting E2 from E1 and then dividing the obtained result by the number of fetches that were executed in the code.

```
1) Print time;
2) //Calculate the exponential function of the numbers in the
   range [0,1000]
3) For(int i=0; i<1000;i++)
4)       LUT[i]=exp(i);
5) //Choose a random number and fetch the stored
   exponential function result from LUT
6) For(i=0; i<1000000;i++)
7)     { pick a random number k;
8)       Fetch the kth element in LUT; }
9) Print time;
```

Figure 4. Code used to determine the energy cost of fetching elements from LUT.

Our physical measurements showed that the platform energy cost of fetching an element from LUT is equal to (12.611×10^{-10}) joule and the CPU energy cost of fetching an element from LUT is equal to (8.765×10^{-10}) joule. These costs are much smaller than the platform energy cost and CPU energy cost for calculating the exponential function which are (50008.33×10^{-10}) and (32598.922×10^{-10}) respectively. In fact, fetching an element from LUT has 99,97% less platform and CPU cost than calculating the exponential function. We should note that using LUT is considered an approximation technique since we are only calculating the exponential function for the variables in the range [0, 1000] with a step of one. Therefore, if we want to fetch the exponential function of (1.3) we will fetch the exponential function of (1) since (1.3) isn't stored in the table. However this approximation causes a great saving of energy.

V. Conclusion and Future Work

In this paper we proposed a simple methodology that allows researchers to estimate the energy of algorithms' kernels using either simulation or physical measurements. This energy estimation helps in giving insight of which kernels should be targeted for energy optimizations in order to have a high energy saving for executing the whole algorithm. As a case study, we applied our methodology on BP algorithm and estimated the energy of its kernels using special boards and acquisition system for the CISC architecture and then using Simplescalar and WATTCH simulation tools for the RISC architecture. Our results showed that the energy cost of kernels can be quite different for different architecture providing support for research on alternative low energy implementations for kernels. We also proved that the kernels' costs obtained by our methodology estimate well the energy cost of the kernels in the real code of the algorithm since the produced energy costs are with only 5.14% error with respect to the cost of the kernels in a real BP code. Finally, since the exponential kernel had a high cost, we proposed an approximation technique by fetching elements from a lookup table. Our experiments showed that fetching an element from LUT consumes 99.97% less energy than calculating the exponential kernel. For future work, we plan to analyze other algorithms and estimate the energy cost of their kernels and examine other opportunities for reducing the energy cost of these kernels.

ACKNOWLEDGEMENT

This work was supported from the Intel-KACST MER research.

REFERENCES

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

[2] M. Dabbagh, H. Hajj, A. Chehab, W. El-Hajj, A. Kayssi, M. Mansour, "A Design Methodology for Energy-Aware Neural Network", in Wireless Communications and Mobile Computing Conference (IWCMC), 2011, pp. 1333-1340.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

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

[13] B. Steigerwald, R. Chabukswar, K. Krishnan, and J. D. Vega, "Creating Energy–Efficient Software", Intel white paper, 2008.

[14] 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.

[15] 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.

[16] D. Schall, V. Hudlet and T. Harder, "Enhancing Energy Efficiency of Database Applications Using SSDs", ACM Proceedings 2010.

[17] 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.

[18] C. Hu, D. A. Jimenez and U. Kremer, "Toward an evaluation infrastructure for power and energy optimizations," in Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, 2005, pp. 8 pp.

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