not modified to reduce their switching activity. This results in tests with arbitrary switching activity. In the case reported in the second part of Table III, the tests were modified to reduce their switching activity as much as possible (without the bound on switching activity based on functional broadside tests). Only the final diagnostic test sets are described, which can be compared with the final diagnostic test sets in Tables I and II.

It can be seen that, without a bound on switching activity, the diagnostic test set has either excessively high or excessively low-switching activity. Both situations are undesirable, and they are avoided by using functional broadside tests.

Instead of starting the generation of the diagnostic test set from $T_{arb.det}$, it is also possible to start it from the low-power test set generated in [9] based on functional broadside tests. Information about this test set is shown in the third part of Table III. It can be seen that the test set distinguishes significantly fewer fault pairs than the final diagnostic test set produced by the procedure described in this brief.

V. Conclusion

This brief described a low-power diagnostic test generation procedure based on functional broadside tests. In this procedure, the switching activity of functional broadside tests is used for bounding the switching activity allowed for nonfunctional broadside tests. It also prevents the switching activity of the tests from being unnecessarily low. Experimental results demonstrated the effectiveness of the procedure in producing low-power diagnostic test sets that detect all the detectable transition faults and leave small numbers of indistinguished fault pairs (in small equivalence classes).

References

[1] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, *et al.*, "A case study of IR-Drop in structured at-speed testing," in *Proc. Int. Test Conf.*, 2003, pp. 1098–1104.

[2] S. Sde-Paz and E. Salomon, "Frequency and power correlation between at-speed scan and functional tests," in *Proc. Int. Test Conf.*, 2008, pp. 1–9, Paper 13.3.

[3] V. R. Devanathan, C. P. Ravikumar, and V. Kamakoti, "On power-profiling and pattern generation for power-safe scan tests," in *Proc. Design, Autom. Test Eur. Conf.*, 2007, pp. 1–6.

[4] M.-F. Wu, H.-C. Pan, T.-H. Wang, J.-L. Huang, K.-H. Tsai, and W. T. Cheng, "Improved weight assignment for logic switching activity during at-speed test pattern generation," in *Proc. Asia South Pacific Design Autom. Conf.*, 2010, pp. 493–498.

[5] I. Pomeranz, "On the generation of scan-based test sets with reachable states for testing under functional operation conditions," in *Proc. Design Autom. Conf.*, 2004, pp. 928–933.

[6] Y.-C. Lin, F. Lu, K. Yang, and K.-T. Cheng, "Constraint extraction for pseudo-functional scan-based delay testing," in *Proc. Asia South Pacific Design Autom. Conf.*, 2005, pp. 166–171.

[7] I. Pomeranz and S. M. Reddy, "Definition and generation of partially-functional broadside tests," *IET Comput. Digital Tech.*, Jan. 2009, pp. 1–13.

[8] I. Pomeranz and S. M. Reddy, "On reset based functional broadside tests," in *Proc. Design Autom. Test Eur. Conf.*, 2010, pp. 1438–1443.

[9] I. Pomeranz, "Augmenting functional broadside tests for transition fault coverage with bounded switching activity," in *Proc. 17th Pacific Rim Int. Symp. Dependable Comput.*, Dec. 2011, pp. 38–44.

[10] T. Gruning, U. Mahlstedt, and H. Koopmeiners, "DIATEST: A fast diagnostic test pattern generator for combinational circuits," in *Proc. Int. Conf. Comput. Aided Design*, Nov. 1991, pp. 194–197.

[11] I. Pomeranz and S. M. Reddy, "A diagnostic test generation procedure based on test elimination by vector omission for synchronous sequential circuits," in *Proc. IEEE Trans. Comput. Aided Design*, May 2000, pp. 589–600.

An Algorithm-Centric Energy-Aware Design Methodology

Hazem Hajj, Wassim El-Hajj, Mehiar Dabbagh, and Tawfik R. Arabi

***Abstract*— The goal of this brief 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. There are two major contributions of this brief: 1) a method for identifying components with high energy consumption in compute-intensive applications. To this end, 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 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 two case studies using data mining algorithms: *K*-nearest neighbors and linear regression. The results highlight the contributions of kernels and memory energy to total energy.**

***Index Terms*— Data mining, energy profiling, energy-aware computing.**

I. Introduction

Energy efficiency is a major challenge for Exascale computing [1] and for a greener environment [2]. While many researchers and companies have driven extensive research at the hardware and architecture levels, fewer efforts, such as the one in this brief, have focused on starting from the application software layer, and examining a top-down energy reduction approach.

Toward the goal of reducing the energy consumed by software applications, we present a four-step top-down approach for identifying software components with high energy consumption, in particular frequently used operations called kernels, and measuring their energy consumption. The energy evaluation method involves isolating the kernel in a separate controlled piece of code with assembly injections, where the kernel energy can be measured in isolation. Unlike previous work targeting instruction level energy [3], we adopt the idea of using isolated code, but we use application-level code instead of instruction-level code and address the restriction on the program execution time. This brief enhances and complements our previous work [4] to make it generally applicable and more accurate. To make the approach more general, the method for energy assessment was expanded from basic kernels such as addition and subtraction to cover more general and aggregate forms of kernel equations such as Euclidean distance. To improve accuracy, assembly injections are introduced to account

Manuscript received December 10, 2012; revised June 24, 2013; accepted October 20, 2013. Date of publication November 21, 2013; date of current version October 21, 2014. This work was supported in part by the Intel-KACST Middle East Energy Efficiency Research Program and in part by the American University of Beirut University Research Board.

H. Hajj and M. Dabbagh are with the Electrical and Computer Engineering, American University of Beirut, Beirut 1107-2020, Lebanon (e-mail: hh63@aub.edu.lb; mmd43@aub.edu.lb).

W. El-Hajj is with the Department of Computer Science, American University of Beirut, Beirut 1107-2020, Lebanon (e-mail: we07@aub.edu.lb).

T. R. Arabi is with the Intel Architecture Group, Intel Corporation, Portland, OR 97124 USA (e-mail: tawfik.r.arabi@intel.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2289906

for cache misses and energy consumed by memory load and store operations. Furthermore, to ensure reliable results, physical energy measurements, conducted on specially instrumented circuit boards, are collected to provide actual and not just simulated measurements. The method is evaluated with data mining (DM) algorithms since they are good representatives of compute-intensive applications that can benefit from energy optimization. Experiments show that the improvement in accuracy is up to 39% compared to that in our previous work [4].

The rest of this brief is organized as follows. In Section II, we present a literature review on the topics of energy optimization and energy profiling. In Section III, we present the proposed methodologies. In Section IV, we present case studies and discuss the results. In Section V, we conclude this brief and present future work.

II. Related Work

Research on energy analysis can be classified into two main focuses: optimizing and profiling.

For optimization, researchers have explored changes at the lower layers of the system, such as hardware [5], and compiler [6] levels. There have also been some considerations at the higher levels. In the operating system layer, the primary ideas in past research were to determine policies that switch idle devices into lower power states by predicting when the full capacity of these devices is not needed [7]–[9]. Another technique that operating systems use to manage power is dynamic voltage and frequency scaling (DVFS) [10], where the operating voltage and frequency for executing a task are reduced to save the consumed energy. In the application layer, the focus has been on improvement methods for specific applications. Examples include energy optimization for scheduling algorithms [11], matrix multiplication [12], sensor network applications [13], and wireless communication [14]. The optimization methods at the application layer can be classified, according to [15], into three main categories: 1) use of contextual awareness such as low battery energy to determine which code to execute [16]; 2) data efficiency such as the use of Solid State Drive (SSD) [17]; and 3) computational efficiency such as the use of parallel programming [18] or the use of hardware acceleration with field-programmable gate array (FPGA) [12].

The focus of this brief is on energy profiling, which is often considered a critical step in identifying bottlenecks for energy optimization. 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 simulation. With physical measurements, the current and voltage of the different hardware components are collected using ammeters and special acquisition systems [19]. The main limitation of that work is that the methods are hardware-centric, and do not show how energy is consumed among the different parts of the application code. With simulation [20], while the methods may be inexpensive and allow users to analyze the performance and power behavior at a cycle level of granularity for the different platform components, they are less accurate than the methods that use physical measurements.

In summary, previous profiling efforts have focused on providing hardware-centric energy profiling for different applications. Very few targeted profiling how energy is consumed among the different parts of the application code based on accurate energy costs. In this brief, we suggest accurate methods for energy profiling that show the percentage of energy consumed by each part of the code. To achieve higher accuracy, we do not assume that the power is constant during the active state. Moreover, we account for the energy consumed by memory activities and conduct all energy analysis using physical boards.

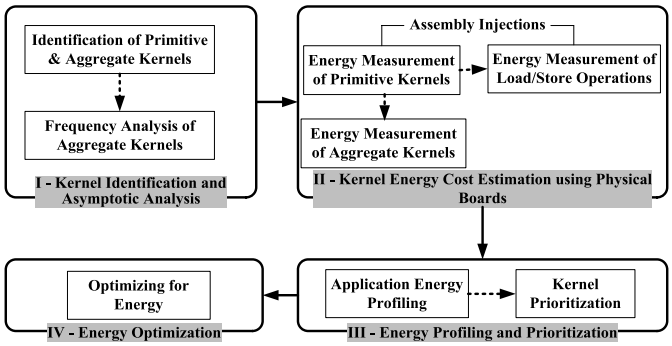

Fig. 1. Proposed four-step methodology for energy-aware algorithms.

III. Proposed Methods

This section describes our four-step methodology for developing energy-aware algorithms. Step 2 in the methodology represents the energy profiling method, and is detailed in Section III-B due to its importance and general applicability.

A. Methodology for Energy-Aware Algorithms

The purpose of the methodology is to analyze the program details and identify opportunities for reducing energy consumption. Fig. 1 shows the flow of the proposed four-step methodology.

Step 1: Kernel identification and kernel asymptotic analysis. Kernels are operations that are repeatedly executed in an algorithm, and are often found in "loops." We categorize kernels into primitive kernels and aggregate kernels. Primitive kernels are the basic operations such as addition, multiplication, division, subtraction, logarithm, and exponential. Aggregate kernels are used to represent any class of mathematical equations that combine multiple primitive kernels. Examples of aggregate kernels in data mining include information gain, Euclidean distance, and normalization. In this step of the methodology, we first identify the primitive and aggregate kernels by manually examining the algorithm. We then conduct an analytical study on every kernel to calculate its frequency of execution as a function of 1) the input data and 2) the parameters of the studied algorithm.

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, along with the energy for the load/store transactions that represent memory and cache interactions. In this brief, we use physical boards for accurate energy measurements. The setup consists of a specially instrumented board and data acquisition system (DAQ) that collects the current and voltage of the major platform components such as CPU and memory.

Step 3: Energy profiling and prioritizing kernels in terms of energy impac. 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 by multiplying the number of times a single kernel is executed (step 1) by the energy cost of executing the same kernel (step 2). The kernel that gives the highest overall energy reduction is the best target for energy reduction.

Step 4: Energy reduction. In this step, the code is modified to reduce the energy effects of kernels with high cost. The research on code optimization is extensive, and is beyond the scope of this brief. However, we mention here some optimization directions to close the loop on the energy-awareness process. Energy optimization methods that focus on the application layer may include data normalization and data reduction, or the use of kernel approximations that trade

```
(1)  Initialize kernel_variables
(2)  for i=1 to N {
(3)        Primitive_Kernel_execute(kernel_variables);
(4)  }
```

Fig. 2. Isolated code to determine primitive kernels energy.

```
Parameters:
  [Min, Max]: The range of values that the variables
              can take.
  N: number of iterations.
  Vx: The offset of variable "x" from the "esp" pointer.
  Vy: The offset of variable "y" from the "esp" pointer.
  Vz: The offset of variable "z" from the "esp" pointer.

Pseudo Code:
(1)  float x, y, z;
(2)  for i=1 to N {
(3)        x=Uniform(Min, Max);
(4)        y=Uniform(Min, Max);
(5)        asm("flds     Vx(%esp)");
(6)        asm("flds     Vy(%esp)");
(7)        asm("fadd     %st(1),%st(0)");
(8)        asm("fstps    Vz(%esp)");
(9)        asm("fstp     %st(0)");
(10) }
```

Fig. 3. Isolated code with assembly injection used to determine the energy cost of the addition primitive kernel (line 7).

high energy cost with lower accuracy. Other methods can benefit from alternative hardware implementations or the best set of compiler optimization techniques for those kernels with high energy cost.

B. Methodology to Estimate the Energy of the Kernels

The goal in this section is to detail the approach to accurately and efficiently 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. Finally, we explain the method used to measure the energy cost of the load and store operations.

1) Energy Cost of Primitive Kernels: 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 in isolation as shown in Fig. 2. In line 1, the variables are initialized and in line 3 the kernel operation is executed. The loop in line 2 is introduced to reduce the measurement variations that may result from the variables and the acquisition system's limited sampling rate. The values of the kernel variables are changed each time we execute the primitive kernel operation. With each iteration, the kernel variables take values from a certain range following a uniform distribution. Such methodology attempts to minimize the effect of the input values.

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. To focus the measurement on the primitive kernel, we propose to introduce assembly injection in the code to further isolate the energy cost of executing the primitive kernel without the impact of load/store operations. An example is shown in Fig. 3.

The figure shows the pseudocode that is used to estimate the energy cost of a 32-bit floating point addition. With every iteration, (lines 3 and 4) the kernel variables (x and y) take values from the range [Min, Max] following a uniform distribution. N indicates the number of times we repeat execution of the kernel operation. Vx, Vy, and Vz represent the offsets from the "esp" pointer where x, y, and z variables are placed, respectively. In lines 5 and 6, the x and y variables are loaded to the floating point stack. Line 7 adds the two elements that are on top of the stack, and saves the result on the top of the stack replacing the top of stack element [i.e., $\%st(0) = \%st(0) + \%st(1)$], where 0 and 1 determine the offset from the stack top. In line 8, the addition result is popped from the stack and stored in z. Finally, line 9 empties the stack by popping the last remaining element.

The energy cost of the addition operation is then calculated by subtracting the energy cost of executing the code without the assembly addition operation (line 7) from the code with the addition operation, and then dividing the result by the number of iterations N.

2) Energy Cost of Aggregate Kernels: Aggregate kernels are composed of multiple primitive kernels. For instance, Information gain is composed of logarithm, addition, and probability, which is in turn composed of division. We derive the energy cost of executing an aggregate kernel based on the energy costs of its primitive kernels constituents. 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.

3) Energy Cost of Load and Store Operations: When executing aggregate kernels such as the Euclidean distance, the compiled assembly code includes load and store operations in addition to the primitive kernel operations (subtraction, addition, etc.). Assembly injection can help isolate the impact of the kernel computation from the impact of the load/store operations. 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 to energy consumption. The load/store operations that we considered in our measurements include: 1) move value, stack; 2) move register, stack; 3) move stack, register; 4) move value, register; 5) move register, register; 6) flds; 7) fstp; and 8) fstps. The first five load/store operations are integer operations and are needed for array indices and loop counter manipulation. The last three are floating-point store/load operations.

The frequency of execution of the load and store operations is determined by examining the compiled assembly code and counting the number of times each type of these operations is executed. The energy cost of the load and store operations can then be calculated following the method described for estimating the energy of primitive kernels. The only difference is that only a load/store instruction is injected in the "for" loop where a different address is selected to be fetched in each iteration in order to have an average cost for a load operation whether a cache hit or a cache miss occurs. The isolated code is then executed with and without the injected load/store assembly operation. The average energy of the load or store operation is calculated by taking the average of the energy differences with and without the operation. The methodology can be equally applied to other floating point load/store operations not covered in this brief.

IV. Experiments and Results

In this section, we conduct experiments to demonstrate the effectiveness of the proposed methods for creating energy-aware algorithms, and to illustrate how to generate the energy profiles of primitive kernels, aggregate kernels, and load/store operations. In the first subsection, we explain the experimental setup for energy measurement. In the second subsection, we pick two DM algorithms [K-nearest neighbors (KNN) and linear regression (LR)] as case studies to validate the four-step methodology for energy awareness.

A. Experiment Setup for Energy Measurement

For this brief, physical measurements are used in order to provide accurate and realistic energy measurements, unlike the majority of other work that relies only on simulation tools. The studies were conducted on an X86-based system that uses a 32-bit instruction

set. However, the proposed methodology remains applicable in 64-bit execution mode. The platform had an Intel Core i7 CPU, 2.80 GHz with 2-GB RAM memory. Windows 7 was installed as the operating system for the board, with the power plan set to balanced mode. We used the <GCC> 4.5.2 compiler to compile the proposed codes as it supports assembly injection. This version of gcc was obtained via MinGW installation. When running experiments to collect energy, all unnecessary programs were stopped, except for the OS and the basic processes required. The digital acquisition DAQ, used to collect the current and voltage for the major components on the board, was the Fluke 2680 series. The sampling rate in our experiments was set to 20 samples/s, which is the highest rate that our DAQ supports. Special software was developed to show how the collected traces vary over time and to calculate the average power, peak power, and energy of the different components of the platform over a certain period. In our experiments, the number of times the primitive kernel operation was repeatedly executed in the isolated code of Fig. 2 was set to $N = 10^9$. The kernel variables ranged between 0 and 1000 and are assigned values based on a uniform distribution. In each experiment, measurements were repeated five times and the median value of the five measurements was chosen as the representative measurement. To ensure accuracy of the reported measurements, the central limit theorem was used to estimate the number of runs required for each simulation scenario, which indicated that five runs should be carried out for every experiment to obtain 90% confidence interval.

B. Case Studies for Energy-Aware Profiling and Computing

In this section, KNN and LR) algorithms were used as case studies for more in-depth analysis, and to demonstrate the applicability of the proposed methodology in creating energy awareness.

1) Energy-Aware Computing–KNN Case Study: Step 1: The methodology starts by identifying the primitive and aggregate kernels of the KNN algorithm. The primitive kernels were identified to be addition, subtraction, division, square, and square root. The aggregate kernels are normalization, Euclidean distance, and comparison. Frequency analysis was then conducted to determine the frequency of the aggregate kernels. The analysis shows that the frequency of occurrence is $N_{\text{tuples}} * N_{\text{features}}$ for "Normalization," N_{tuples} for "Euclidean Distance," and N_{features} for "Compare," where N_{tuples} and N_{features} are the number of tuples and features in the dataset, respectively. For this experiment, the first nearest neighbor (1NN) was considered, with the dataset having $N_{\text{tuples}} = 150$, and $N_{\text{features}} = 5$.

Step 2: In this step, the physical energy measurements of the primitive kernels, aggregate kernels, and load/store operations were conducted as described in the proposed methodology. We also calculated the energy consumption of multiplication, logarithm, and exponential for reference and for later use in other algorithm analysis. We used 32-bit floating point operations. Fig. 4 shows the energy measurements in logarithmic scale. As shown in the figure, the cost of square is slightly higher than multiplication, although their costs should intuitively be the same given that the Intel Core i7 we used has no dedicated square instruction. However, the C++ function "pow(x,2)" was used to simulate square which incurs some overhead on the stack and on handling the variables.

The energy measurements for the aggregate kernels for KNN (normalization, Euclidean distance, and comparisons) were derived based on the energy cost of the primitive kernels. The energy results for KNN aggregate kernels are shown in Fig. 5. Multiplying the frequencies derived in the first step by the energy costs of Fig. 4 results in the energy profiling of the presented aggregate kernels.

Step 3: The third step shows the algorithm energy profile and the contributions of the different kernels to the overall algorithm. This step consists of calculating the total energy of the algorithm, which

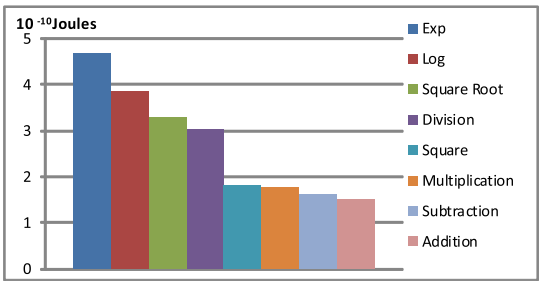

Fig. 4. Energy cost of the primitive kernels using physical measurements (shown in logarithmic scale).

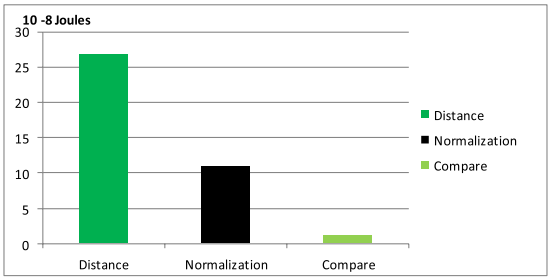

Fig. 5. Energy cost of KNN aggregate kernels.

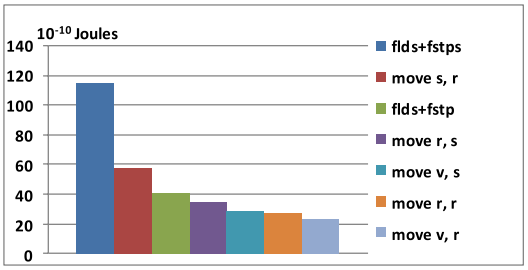

Fig. 6. Energy cost of move operations (load/store) using physical measurements. In the figure, v stands for value, r for register, and s for stack. Note that, flds + fstps and flds + fstp are normalized per operation.

was implemented in C++. The energy contribution of each aggregate kernel was calculated by multiplying the frequency of the aggregate kernel by its energy cost. Similarly, to find the contribution of the load/store operations (Data Move), we multiply the energy cost of each move operation (costs reported in Fig. 6) by its frequency in KNN.

Fig. 7 shows a pie chart for the energy distribution of the different aggregate kernels relative to the total energy. The chart provides optimization hints toward 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 (Fig. 5), the frequency of normalization is higher than the distance, and thus the energy contribution of normalization is higher than the distance. The Data Move operations account for 12% of the total energy consumed by KNN. 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.

Step 4: In this step, we study the effect of optimization and the potential savings to the total energy that can be achieved by reducing kernel's energy. The largest overall energy reduction is obtained by reducing the cost of normalization followed by Euclidean distance. In this step, reduction in kernel energy can be achieved using

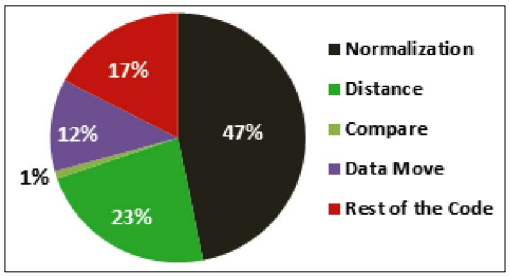

Fig. 7. Energy contribution of KNN aggregate kernels to the total energy.

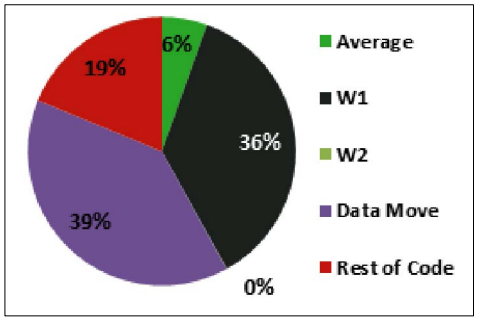

Fig. 8. Energy contribution of LR aggregate kernels to the total energy.

one of the approaches mentioned in Section II, e.g., implementing compiler optimization techniques on expensive kernels or using kernel approximations instead of exact values where accuracy was traded for considerable reduction in energy [4].

2) Energy-Aware Computing—LR Case Study: This experiment targeted LR. The LR aggregate kernels were determined to be the average of the data, and the regression coefficients. The profiling results are shown in Fig. 8. W2 has a 0.05% overall energy contribution and is indicated in the chart as 0%. The results 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 operations constituted a large percentage of the total energy.

V. Conclusion

This brief has introduced two methods for developing an energy-aware algorithm. The first method is a four-step process for energy analysis and profiling with particular emphasis on kernel-based evaluation, which are commonly found in compute-intensive applications. The second method provides a more in-depth methodology for estimating the energy of primitive kernels and load/store operations by using isolated code with assembly injection. Energy for aggregate kernels is extrapolated from the energy of primitive kernels. The experiments showed that different algorithms can have different kernels as top energy consumers. For future work, the ultimate goal is to develop a compiler that can automatically transform the code of energy-expensive kernels into more efficient assembly codes with lower costs.

References

[1] P. Kogge, "The tops in flops," *IEEE Spectrum Mag.*, vol. 48, no. 2, pp. 48–54, Feb. 2011.

[2] C. Pettey, (26 Apr. 2007). *Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO_2 Emission* [Online]. Available: http://www.gartner.com/it/page.jsp?id=503867

[3] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 437–445, Dec. 1994.

[4] M. Dabbagh, H. Hajj, A. Chehab, W. El-Hajj, A. Kayssi, and M. Mansour, "A design methodology for energy aware neural networks," in *Proc. 7th Int. Wireless Commun. Mobile Comput. Conf.*, 2011, pp. 1333–1340.

[5] C. W. Huang and S. Tsao, "Minimizing energy consumption of embedded systems via optimal code layout," *IEEE Trans. Comput.*, vol. 61, no. 8, pp. 1127–1139, Aug. 2012.

[6] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini, "Code transformations for energy-efficient device management," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 974–987, Aug. 2004.

[7] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 1, pp. 42–55, Mar. 1996.

[8] S. Albers, "Energy-efficient algorithms," *Commun. ACM*, vol. 53, no. 5, pp. 86–96, May 2010.

[9] Y. Lu and G. De Micheli, "Comparing system level power management policies," *IEEE Design Test Comput.*, vol. 18, no. 2, pp. 10–19, Mar./Apr. 2001.

[10] V. Raghunathan, C. L. Pereira, M. B. Srivastava, and R. K. Gupta, "Energy-aware wireless systems with adaptive power-fidelity tradeoffs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 2, pp. 211–225, Feb. 2005.

[11] Y. Lee, J. Kim, and C.-M. Kyung, "Energy-aware video encoding for image quality improvement in battery-operated surveillance camera," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 2, pp. 310–318, Feb. 2012.

[12] J.-W. Jang, S. B. Choi, and V. K. Prasanna, "Energy- and time-efficient matrix multiplication on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 11, pp. 1305–1319, Nov. 2005.

[13] P. K. Dutta and D. E. Culler, "System software techniques for low-power operation in wireless sensor networks," in *Proc. IEEE/ACM ICCAD*, Nov. 2005, pp. 925–932.

[14] N. Abbas, H. Hajj, and A. Yassine, "Optimal WiMAX frame packing for minimum energy consumption," in *Proc. 7th IWCMC*, 2011, pp. 1321–1326.

[15] P. Larson, "Energy-efficient software guidelines," Intel Software Solution Group, Tsukuba, Japan, Tech. Rep. 23, 2008.

[16] P. Unnikrishnan, G. Chen, M. Kandemir, and D. R. Mudgett, "Dynamic compilation for energy adaptation," in *Proc. IEEE/ACM ICCAD*, Nov. 2002, pp. 158–163.

[17] D. Schall, V. Hudlet, and T. Harder, "Enhancing energy efficiency of database applications using SSDs," in *Proc. ACM*, May 2010, pp. 1–9.

[18] 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," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, May 2010.

[19] H. Kothuru, G. Virupakshaiah, and S. Jadhav, "Component-wise energy breakdown of laptop," in *Proc. 6th Annu. GRASP Symp.*, 2010, pp. 1–54.

[20] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *SIGMETRICS Perform Eval. Rev.*, vol. 36, no. 2, pp. 26–31, 2008.