Processor Energy Estimation Method using Cycle-approximate Simulator

Woo-Hong Byun, Kyungsu Kang and Chong-Min Kyung
Department of EECS
Korea Advanced Institute of Science and Technology
Daejeon, 305-701, Korea

Abstract— **In existing methods for processor energy estimation, the information on the internal state of processor architecture, i.e., (de)activation of specific modules such as pipeline stages and the cache, is available via a cycle-accurate simulator. Despite good accuracy, above methods are too slow due to the complexity of cycle-accurate simulation. This paper describes a new technique for estimating the energy consumption of processors by predicting the internal information of processor architecture through a cycle-approximate simulator. We have evaluated our approach in comparison with an existing energy estimation method using a cycle-accurate ARM v5 architecture simulator. Experimental results shows that the proposed method achieves 97% accuracy and 28 times speed-up on average.**

Index Terms— **processor energy estimation, software energy estimation, cycle-approximate simulator**

I. Introduction

Reducing power consumption in battery-operated devices such as PDAs, laptop computers, mobile phones and UMPCs is a very critical issue. In order to optimize software in the aspect of energy consumption, the energy model of target hardware is essential. Various attempts to estimate the energy consumption using a cycle-accurate simulator in system level including cache were proposed in [1] [2] [3] [4] [5], where the energy consumption is estimated by using the profiling information provided by a cycle-accurate simulator. The cycle-accurate simulator provides information such as pipeline stages of processor, read/write miss of cache at each cycle when the application is executed. Such information is directly related to the power state of the processor and the cache. However, such low-level energy estimation as cycle-accurate simulation is difficult to be employed in the process of the software development due to its low speed. As the complexity of software gradually increases, the estimation of energy consumption at a high level becomes more important in order to raise the estimation speed.

Previous attempts to estimate energy consumption using fast cycle-accurate HDL simulators was not quite successful [6]. In an attempt to estimate the energy consumption at system level using an instruction level power analysis method [7], one calculates the total energy consumption in a processor as the sum of base energy and inter-instruction energy of instructions executed in the processor. This method quickly estimates the energy consumption of the processor using functional ISS (Instruction Set Simulator). However, this model does not consider energy consumption of the cache.

On the other hand, C-code static analysis method were introduced [8] [9] which estimate the energy consumption without functional simulation. After dividing the C-code into many constructs, each construct is compiled many times. Based on this result, the number and the type of related instructions are inferred when a certain construct is executed. Then the energy consumption is calculated using this inference. However the estimation accuracy of these methods is very low because the probability of branch operations being taken is assumed to be constant and the pipeline stall is not considered.

In this paper, we propose an energy estimation method via a cycle- approximate simulator. To our best knowledge, it is the first approach to speed-up the energy estimation by predicting the internal information of processor through a cycle-approximate simulator. We demonstrated in our experimental results that the proposed estimation method yields 97% accuracy and 28 times speed-up on average, in comparison with an existing energy estimation method, using the cycle-accurate simulator. This results show that our proposed method is more accurate than C-code analysis method while much faster than the cycle-accurate simulator.

This paper is organized as follows. Section II gives the knowledge of our energy model for ARM1176JZF-S [10], the target processor. Section III explains the prediction method of internal information, which is needed for the energy estimation, through the cycle-approximate simulator. Experimental results are presented in Section IV.

II. Energy model using cycle-approximate simulator

A. Core

The core of ARM1176JZF-S has three pipes classified as ALU, MAC, and LS (Load/Store) pipes as shown in Fig. 1. The ALU pipe executes most of the ALU operations, while the MAC pipe executes all the enhanced multiply and multiply-accumulate (MAC) instructions. the LS pipe executes all load and store instructions.

The energy model of the core is based on instruction-level energy characterization and presented as follows.

$$E_{core} = \sum_{i=1}^{M} n_i \cdot \overline{E_i} + n_{bt} \cdot E_{bt} + C_s \cdot E_s \qquad (1)$$

where M is the number of instruction groups, n_i is the number of executed instructions in each instruction group,

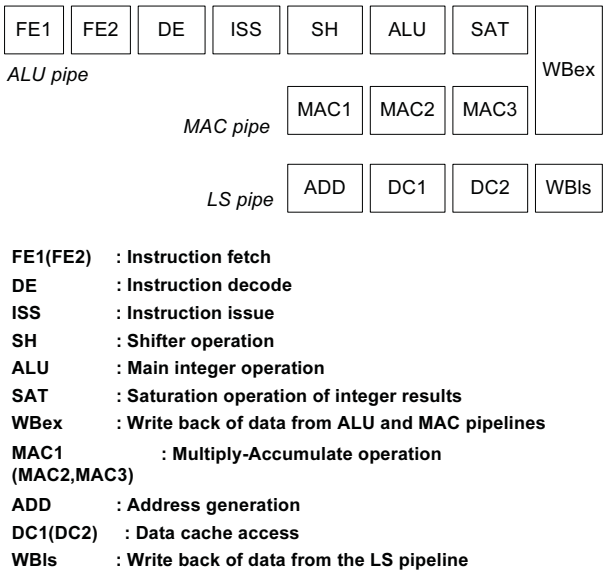

Fig. 1. ARM1176JZF-S pipeline architecture

$\overline{E_i}$ is the average energy consumption of the instructions in each instruction group, n_{bt} is the number of branch-taken instructions, E_{bt} is the energy consumption due to branch-taken instructions, C_s is the number of clock cycles wasted due to the pipeline stall and E_s is the energy consumption per clock cycle due to the pipeline stall.

The instructions are grouped according to energy cost of each instruction. This paper divided the instructions into seven groups which are ALU, MAC1, MAC2, MAC3, MAC4, Load and Store group respectively. The average energy consumption of the instructions in each group and their variances are shown in Table I.

TABLE I
AVERAGE ENERGY CONSUMPTION OF INSTRUCTIONS AND VARIANCE IN EACH INSTRUCTION GROUP

Instruction group	Average energy consumption (nJ)	Variance
ALU	0.4195	0.018
MAC1	0.7917	0.010
MAC2	0.5244	0.012
MAC3	1.2607	0.045
MAC4	1.5032	0.004
Load	0.5493	0.051
Store	1.9739	0.010

The ARM1176JZF-S architecture largely divided the instruction group into three, according to their pipeline path where each instruction is executed, because instructions using the same hardware resources take a ply of having similar energy consumption and execution clock cycles. The instructions executed in MAC pipe can be deeply classified into four instruction groups according to the size of register for MAC operation. The register size for each MAC group is shown in Table II. Also, the instructions using LS pipeline path are split into load and store groups due to their different execution clock cycles.

In the energy estimation, some energy overhead, which is not directly modeled by the instruction-level energy characterization, can be generated due to the branch-taken operation or the pipeline stall. When the processor executes branch-taken instructions, instructions prefetched in the pipeline are flushed and that leads to some energy consumption overhead. Therefore, the energy consumption due to the branch-taken operation and the pipeline stall are included in our energy model for accurate estimation.

TABLE II
THE REGISTER SIZE OF EACH MAC INSTRUCTION GROUP

MAC group	Register size
MAC1	32 bit operation, 32 bit result
MAC2	16 X 32 or 16 X 16 operation, 32 bit result
MAC3	32 bit operation, 64 bit result
MAC4	64 bit operation, 64 bit result

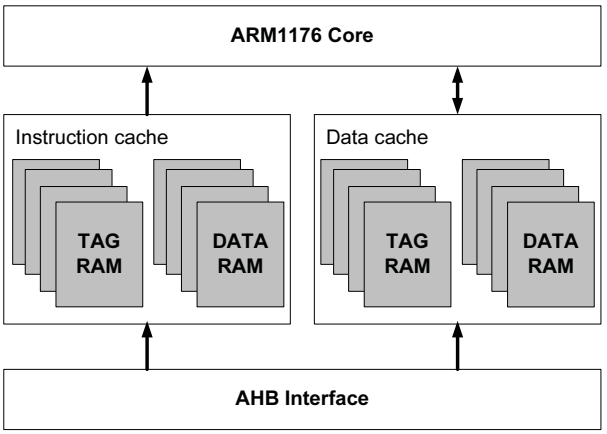

Fig. 2. ARM1176JZF-S cache architecture

B. Cache

The cache of ARM1176JZF-S is a Harvard architecture as shown in Figure 2. It is four-way set associative with a line length of eight words per each cache line and the size of the caches can be from 4KB to 128KB. Both the instruction cache and the data cache can provide two words per cycle for all requests.

The energy model of cache is based on the energy states of the cache and presented as follows.

$$E_{cache} = \sum_{j=1}^{K} n_j \cdot E_j + C_{idle} \cdot E_{idle} \quad (2)$$

where K is the number of energy states in cache, n_j is the number of accesses to each energy stage, E_j is the energy consumption of each energy state, C_{idle} is the number of clock cycles when the cache is not accessed and E_{idle} is the energy consumption of cache when cache is in idle state.

This paper divides the energy states of the cache into six which are instruction cache hit, instruction cache miss, data cache read hit data cache read miss, data cache write hit and data cache write miss. The criterion for this division is the type of operations of the cache and it is reasonable because, in each operation of the cache, it uses different amount of hardware resources respectively. For example, a cache miss results in the

requests required to do the line fill being made to the level two interface and it consumes more energy than a cache hit. For the cache which is not accessed during the cycle, their static energy values are added to cache energy model by E_{idle}.

III. Parameter Estimation Method

In this section, we explain the prediction method of parameters such as the number of cache hit/miss, which are needed to estimate energy consumption of a processor but are not available in the cycle-approximate simulator.

A. Pipeline stall

The pipeline stall occurs due to the reasons such as data dependency and cache miss. This can be calculated simply by subtracting ideal clock cycle counts of each instruction executed in the processor from the total execution clock cycle counts of the processor. The pipeline stall is presented as follows.

$$C_s = C_{total} - (\sum_{i=1}^{M} n_i \cdot \overline{C_i} + n_{bt} \cdot \overline{C_{bt}}) \quad (3)$$

where C_{total} is the number of total execution clock cycles, $\overline{C_i}$ is the average clock cycles of instructions in each instruction group, $\overline{C_{bt}}$ is the average clock cycle overhead of branch-taken instructions.

As shown in equation 3, the estimated number of pipeline stalls does not include pipeline stalls due to branch-taken operations because the energy consumption of pipeline stall due to branch-taken instructions is already modeled in equation 1 as energy consumption overhead E_{bt}.

For characterizing $\overline{C_i}$, the ideal number of clock cycles of each instruction is extracted by using ARM1176JZF-S Technical Reference Manual [10]. $\overline{C_{bt}}$ is characterized by the cycle-accurate simulation and the number of average clock cycles is eight which is the same as the number of pipeline stages.

B. Instruction cache and data cache read/write miss

In Figure 2, most of the memory accesses of the core are not shown in AHB interface because of the excellence of cache (i.e.,practical use of data locality in cache). However, the core requests required data from external memory and the request signal is visible in AHB interface when a cache miss occurs. Therefore, the number of cache miss can be calculated by monitoring the AHB interface and which is explained in Figure 3.

C. Instruction cache read hit

As mentioned above, the caches of ARM1176 can provide only two words (i.e.,instructions) per a clock cycle for all requests and all instructions executed in the processor are fetched only from the instruction cache. That means the total number of executed instructions is proportional to twice the number of cache accesses and presented as follows.

$$n_{ins} = 2 \cdot (n_{ins\_rh} + n_{ins\_rm}) - n_{bt} \cdot \overline{n_{ins\_flush}} \quad (4)$$

where n_{ins} is the number of instructions executed on the processor, n_{ins_rh} is the number of instruction cache read hit, n_{ins} is the number of instruction cache read miss and $\overline{n_{ins\_flush}}$ is the average number of flushed instructions due to branch-taken operations.

When the processor executes a branch-taken instruction, the instructions which have already been fetched before branch instruction being executed in the pipeline are flushed. Therefore, for calculating the total number of executed instructions, we subtract flushed instructions due to branch-taken operations from the number of instructions prefetched from the instruction cache. The number of flushed instructions which is characterized through cycle-accurate simulation is eight, which is the same as the number of pipeline stage. Also, the number of executed instructions can be known through the cycle-approximate simulations and the equation for estimating the read hit of instruction cache read is rewritten as follows.

$$n_{ins\_rh} = \frac{n_{ins} + n_{bt} \cdot \overline{n_{ins\_flush}}}{2} - n_{ins\_rm} \quad (5)$$

D. Data cache read/write hit

The number of data cache read hit can be calculated by subtracting the number of data cache read miss counts from the number of load operations executed in the processor. The ARM1176JZF-S processor has three kinds of load operations which are load, multiple load and swap instructions. The load instruction copies data from memory to a register. The multiple load instruction copies multiple data from memory to registers and the swap instruction copies data from some memory line to another memory line. The formula for calculating the number of data cache read hit is shown as follows.

$$n_{data\_rh} = n_{ldr} + n_{ldm} \cdot [\frac{n_{ldm\_regs} + 1}{2}] + n_{swp} - n_{data\_rm} \quad (6)$$

where n_{data_rh} is the number of data cache read hit, n_{ldr} is the number of load instructions executed in the processor, n_{ldm} is the number of multiple load instructions executed in the processor, n_{ldm_regs} is the number of registers storing data resulted from each multiple load instruction, n_{swp} is the number of swap instructions executed in the processor, n_{data_rm} is the number of data cache read misses and [] is the gauss notation.

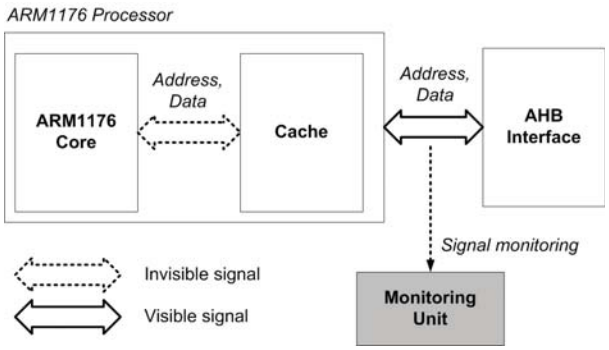

Fig. 3. ARM1176JZF-S cache architecture

TABLE III
THE ACCURACY OF PROPOSED ENERGY ESTIMATION METHOD

Benchmark	Proposed (nJ)	CaEst (nJ)	Error (%)
Susan	1584700.05	1507471.88	5.12
Qsort	35383544.53	34738031.43	1.86
FFT	430812.83	423492.11	1.73
StringSearch	483820.82	477878.27	1.24
H.264	110971235.79	108569597.00	2.21

A multiple load instruction takes different number of data cache accesses depending on the number of target registers, n_{ldm_resg}. We use the gauss notation for calculating the number of data cache accesses due to the multiple load instructions because the caches can provide two words per a clock cycle, as mentioned above.

The number of data cache write hit can also be calculated in a similar way as the method of calculating the number of data cache read hit and the equation is presented as follows.

$$n_{data\_wh} = n_{str} + n_{stm} \cdot [\frac{n_{stm\_regs} + 1}{2}] + n_{swp} - n_{data\_wm} \quad (7)$$

where n_{data_wh} is the number of data cache write hit, n_{str} is the number of store instructions executed in the processor, n_{stm} is the number of multiple store instructions executed in the processor, n_{stm_regs} is the number of source registers used by each multiple store instruction and n_{data_wm} is the number of data cache write misses.

IV. EXPERIMENTAL RESULTS

To validate the accuracy of our energy estimation method, we compared our energy estimation results to an existing energy estimation method [12] which uses a cycle-accurate simulator (later referred as CaEst). We built our platform using the SoC Designer 7.0 [11] and tested our estimation method with the MiBench [13] and H.264 [14] applications.

Table III shows the result of our estimation method. As a result, the proposed energy estimation method showed 2.43% error on average and 5.12% at most for the applications.

Table IV shows the accuracy of parameter estimation method proposed in Section III. We did not report the result of the instruction cache and data cache read/write miss. Because the miss counts are not extracted by estimating some values but by monitoring AHB bus, we took the miss counts perfectly(i.e.,the accuracy of cache miss counts is 100%). The origin of the estimation error comes from the variance of average values such as average clock cycles of instructions in each instruction group, average clock cycle overhead of the branch-taken instructions and the number of flushed instructions due to the branch-taken prediction. In Section III, to estimate data cache read/write hit, we assumed that one cache access is resulted from one load operation. However, in reality, the cache has capability to provide two words in a cycle. That means two load operations can be processed by one cache access if the loaded data are in the same cache line and it is another factor of error for estimating parameters.

Table V shows the speed-up of the proposed estimation method. This result shows that the proposed estimation method is 28.47 times faster than the CaEst on average.

TABLE IV
VALIDATION ON THE PARAMETER ESTIMATION METHOD

Benchmark	Error (%)			
	Pipeline stall	I-$ hit	D-$ read hit	D-$ write hit
Susan	4.16	4.25	1.12	0.19
Qsort	6.34	2.78	3.79	5.65
FFT	7.46	4.29	5.14	2.53
StringSearch	0.59	6.53	5.28	5.10
H.264	6.35	1.94	2.24	4.78
Average (%)	4.98	3.96	3.51	3.65

TABLE V
THE SPEED-UP OF PROPOSED ENERGY ESTIMATION METHOD

Benchmark	Proposed (second)	CaEst (second)	speed-up
Susan	51.60	1658.96	32.15
Qsort	2015.86	54276.22	26.92
FFT	19.66	517.56	26.33
StringSearch	22.57	579.10	25.66
H.264	4296.43	134526.41	31.31

V. CONCLUSION

In this paper, the energy estimation method using the cycle-approximate simulator has been proposed for the first time. We proposed the energy model and the parameter estimation method by using the cycle-approximate simulator. The proposed method can estimate the energy consumption of the processor core and cache faster than the existing energy estimation method using the cycle-accurate simulator. According to the experiment, our energy estimation method yields 28 times speed-up while having 97% average accuracy on average.

REFERENCES

[1] T. Simunic, L. Benini, G. De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. DAC, 1999.
[2] L. Benini, D. Bruni, M. Chinosi, C. Silvano, V. Zaccaria, R. Zafalon. A Power Modeling and Estimation Framework for VLIW-based Embedded Systems. PATMOS, 2001.
[3] ACS Beck, JCB Mattos, FR Wagner, L Carro. CACO-PS: a general purpose cycle accurate configurable power simulator. SBCCI, 2003.
[4] Hyunsuk Kim, Seokhoon Kim, Ikhwan Lee, Sungjoo Yoo, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, Soo-Kwan Eo. An Industrial Case Study of the ARM926EJ-S Power Modeling. ISOCC, 2005.
[5] CH Chang, D Marculescu. Design and Analysis of a Low Power VLIW DSP Core. ISVLSI, 2006.
[6] P. Landman. High-Level Power Modeling, Estimation, and Optimization. ISLPED, 1996.
[7] V Tiwari, S Malik, A Wolfe. Power analysis of embedded software: a first step towards software power minimization. IEEE Transactions on VLSI Systems, 1994.
[8] I Kadayif, M Kandemir, G Chen, N Vijaykrishnan, M. J. Irwin, A. Sivasubramaniam. Compiler-directed high-level energy estimation and optimization. ACM TECS, 2005.
[9] I Kadayif, M Kandemir, N Vijaykrishnan, M. J. Irwin, A. Sivasubramaniam. EAC: a compiler framework for high-level energy estimation and optimization. DATE, 2002.
[10] ARM Ltd. Arm1176JZF-S Technical Reference Manual. http://www.arm.com.
[11] ARM SoC Designer. http://www.arm.com/product/DevTools/MaxSim.html
[12] H. Kim, S. Kim, I. Lee, Y. Sungjoo, C. Eui-Young,K.-M. Choi, J.-T. Kong, and S.-K. Eo. An industrial case study of the arm926ej-s power modeling. ISOCC, 2005.
[13] Guthaus M.R., Ringenberg J.S., Ernst D., Austin T.M., Mudge T., Brown R.B. MiBench: A free, commercially representative embedded benchmark suite Workload Characterization, 2001
[14] ISO/IEC 14496-10 and ITU-T Rec. H.264, Advanced Video Coding, 2003