An Improved Instruction-Level Power Model for ARM11 Microprocessor

Wei Wang
Electronics & Electrical Engineering Research Group
Electronics & Computer Science
University of Southampton
Southampton SO17 1BJ, UK
ww5g09@ecs.soton.ac.uk

Mark Zwolinski
Electronics & Electrical Engineering Research Group
Electronics & Computer Science
University of Southampton
Southampton SO17 1BJ, UK
mz@ecs.soton.ac.uk

ABSTRACT

The power and energy consumed by a chip has become the primary design constraint for embedded systems, which has led to a lot of work in hardware design techniques such as clock gating and power gating. The software can also affect the power usage of a chip, hence good software design can be used to reduce the power further. In this paper we present an instruction-level power model based on an ARM1176JZF-S processor to predict the power of software applications. Our model takes substantially less input data than existing high accuracy models and does not need to consider each instruction individually. We show that the power is related to both the distribution of instruction types and the operations per clock cycle (OPC) of the program. Our model does not need to consider the effect of two adjacent instructions, which saves a lot of calculation and measurements. Pipeline stall effects are also considered by OPC instead of cache miss, because there are a lot of other reasons that can cause the pipeline to stall. The model shows good performance with a maximum estimation error of -8.28% and an average absolute estimation error is 4.88% over six benchmarks. Finally, we prove that energy per operation (EPO) decreases with increasing operations per clock cycle, and we confirm the relationship empirically.

Categories and Subject Descriptors

B.7 [**INTEGRATED CIRCUITS**]: Design Aids; C.0 [**Computer Systems Organization**]: GENERAL

Keywords

Power modeling, Energy modeling, Energy estimation

1. INTRODUCTION

Low power consumption is becoming more and more important, because a lot of embedded systems use batteries as energy sources. To make a low power processor, a number of hardware techniques have been developed, such as clock gating and power gating. Software also affects the power consumption, but there is a gap between software and hardware that makes it hard to predict which code consumes the least power or energy. Instruction-level power analysis is one method to suggest how to write a power-efficient software applications [17].

This paper presents a new instruction-level power model to estimate the power usage of a program on a single core processor: ARM1176JZF-S. Instead of studying the instructions individually, we consider the average power of a program. We present a concise model with good performance – the maximum error is less than 9% across six benchmarks.

The second contribution is to consider pipeline stalls in the model by using operations per cycle (OPC) instead of cache miss as the metric and this makes the model both concise and accurate. Pipeline stalls have not always been well considered in previous models; some models consider too many conditions under which the pipeline might stall. Such models are hard to generate and use. Models that do not consider it sufficiently lose accuracy. We use OPC to analyze the effect of the pipeline to be both concise and accurate. Furthermore, we consider conditions under which increasing operations per cycle (OPC) leads to decreased energy per operation (EPO), both theoretically and empirically. Therefore, making the instruction pre-fetch unit and branch predictor work more efficiently can reduce the energy of a program.

This paper is organized as follows: The background is presented in section 2. In section 3, the key parameters of the processor and the experimental method are introduced. The power usage and analysis of the basic tests and the instruction-level power model are shown in sections 4 and 5 respectively. The results from some standard benchmarks are presented in section 6. Section 7 is divided into two parts. The first part is a comparison between our model and some other work. In the second part, we analyze the relationship between EPO and OPC and present some suggestions for writing energy efficient software.

2. BACKGROUND

Instruction-level power and energy models have been studied over many years but the original model is still the basis of a much work. It expresses the average energy as [17]:

$$E = \sum_i (B_i \times N_i) + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k, \quad (1)$$

where, E is the total energy consumed by the program and B_i is the basic energy of instruction i. For example the instructions MOV and LOAD may consume different energies, therefore, they have different basic costs. N_i is the number of times of this instruction appears in the program. $O_{i,j}$ is the overhead energy consumed by a pair of instructions i, j. For example, if ADD consumes 1 nJ and SUB consumes 2 nJ, the sum of ADD and SUB is 3nJ but if we run ADD and SUB together, the energy consumed might be 3.5 nJ. This 0.5 nJ difference is called overhead energy. $N_{i,j}$ is the number of different pairs. E_k is any other additional energy, such as cache misses, pipeline stall penalties, etc. This basic model has been used to study the ARM7TDMI [10], and it has been extended to Leon3 [12] and PowerPC 603e microprocessor [1].

When analyzing the instruction-level power consumption, some research shows that the power may be affected by the value of the operand [14, 12], and a new model which considers the operand was created to address this: the data dependency model. The following is an example of this kind of model [14]:

$$\begin{aligned} power_{average} &= P_{data} + P_j + P_{i,j} \\ &= K_1 \cdot n_1 + \cdots + K_n \cdot n_n + K_0 + C_{i,j}, \end{aligned} \quad (2)$$

where K_i and n_i are, respectively, the weights and the elements which can influence the power consumption. For example, K_0 is the minimum cost for the particular instruction and this is the same idea as the basic cost in the previous model. $C_{i,j}$ is the additional cost of changing instructions from i to j, like the overhead energy above.

However, neither of the previous two models considers pipeline stalls much. Moreover, it is not convenient to measure all of the overheads caused by all possible instruction pairs. For example a model based on DSP 56K needs 1176 measurements in total [11] because there are 49 instructions in ISA. Bazzaz et al created a new energy model based on these two models [3]. The benefit of this model is that it considers the factors which make the pipeline stall and addresses the problem of overhead energy. But there are 38 parameters in this model which makes it difficult to use.

Functional-level power analysis address these problems. The main idea is to split the processor into different parts and generate power-sensitive parameters for each block. For example, the IMU power may be affected by the rate of instruction dispatching and the CPU stall rate [9]. The total power consumption is the sum of the blocks. The benefit of this model is that it shows how software affects the power in detail and software engineers can use this to write low power/energy software. However, there are many variables to consider and the functional level-model is hard to create because it requires detailed understanding of the architecture of the processor.

To evaluate power and energy consumption of programs, some simulators have been developed such as SimplePower[18] and Wattch[4]. However, these tools have some disadvantages, one of which is that the simulator only supports limited RTL level processor models, and it is difficult to add commercial modern processor models because of the commercial secrets. For example, SimplePower simulates an in-order 5-stage pipeline, and some advanced technologies are not used, such as branch prediction and instruction prefetch. On the other hand, low power technologies are not used in these simulators such as clock-gating and dynamic voltage and frequency scaling (DVFS). Therefore, the power simulator cannot fully replace the real measurement.

Sometimes, we are more concerned about the energy usage of a program than the power. However, an energy model is hard to create because there are a lot of factors that affect the energy of a program. For example, the types of instruction, the Hamming distance of two adjacent instructions and pipeline stalls can all affect the energy. To use an energy model, the target program must also be analyzed, which is usually achieved by simulation to get the input parameters of the model, such as the cache miss rate and distribution of each type of instruction in the target program. An average power model can be generated much more easily than an energy model and the parameters of the power model are also easier to obtain.

Instead of establishing the energy model directly, it is easier to formulate the energy of a program in two steps: 1) create the power model, and 2) measure the runtime. The run time of the program can be measured easily using program counter, for example, and simulated by instruction set simulators, such as gem5 [5]. Therefore, the power model can also be extended to model the energy.

Recently, a new instruction-level energy model was created by Shao and Brooks. They used the energy per instruction, EPI, as the parameter, [15]. They use runtime instruction counts and EPI to compute the energy of the program.

This paper presents a new instruction-level model based on the ARM11 which is convenient to create and use and takes pipeline stall factors into account.

3. EXPERIMENTAL SETUP

A Mini6410 development board was chosen because it uses the ARM1176JZF-S as the CPU [13], as part of a Samsung S3c6410A processor. Moreover, it supports DVFS and also has interfaces for low power memory. Table 1 shows the key parameters of the CPU [2].

Figure 1 shows the power supply and the test platform. To make the necessary power measurements, a 0.51Ω series resistor was included between the power supply and the CPU. A digitizing oscilloscope with a sample rate of 2GHz was used to measure the instantaneous power as tests were carried out. The instantaneous power, average power and the energy are calculated by the following three equations:

$$\begin{aligned} P(t) &= I(t)V(t) \\ &= \frac{V_1(t) - V_2(t)}{R} \times V_2(t) \\ &= \frac{V_1(t) - V_2(t)}{0.51} \times V_2(t) \end{aligned} \quad (3)$$

Table 1: **Samsung S3c6410A features**

Technology	65nm
Vdd	1.1V
Frequency	533MHz
Prefetch Unit	uses both static and dynamic branch prediction
Branch target address cache	128-entry
L1 cache	write-through cache with 16KB, 4-way, 2 words per cycle for all requesting sources,
L1Dcache	write-through cache witch 16KB, 4-way, 2 words per cycle for all requesting sources

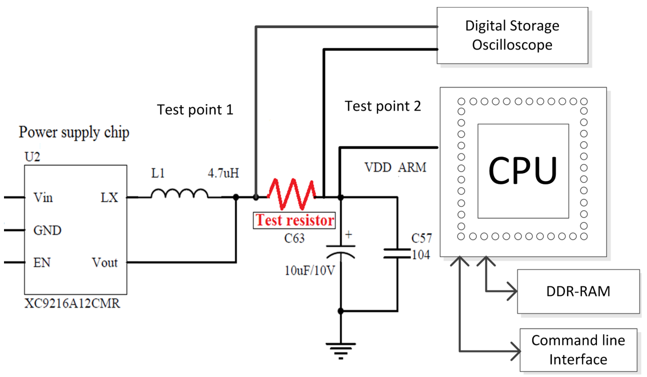

Figure 1: Power supply schematic diagram.

$$\begin{aligned} P_{average} &= \int_0^T P(t)dt/T \\ &= \int_0^T \frac{V_1(t) - V_2(t)}{0.51} \times V_2(t)dt/T \end{aligned} \tag{4}$$

$$E = P_{average} \times T \tag{5}$$

where, $V_1(t)$ and $V_2(t)$ are the instantaneous voltages at test points 1 and 2 in Figure 1. Linux is used as the operating system. The run time of the experiments and benchmark applications can be extracted directly.

4. BASIC POWER CONSUMPTION OF DIFFERENT INSTRUCTIONS

As described above, the most significant component of a model is the basic instruction power consumption. Therefore, we wrote different tests for different target instructions. The main body of each test is a loop with a number of instances of the same opcode in each loop. In order to avoid cache misses, we chose 8KB (2000 instructions) as the loop size. All of these tests can be fully cached, because both the L1 data and instruction caches are 16KB.

Figure 2 shows the power consumption of arithmetic logic functions in different addressing modes: multiply, load and store. In order to distinguish between addressing modes, we have put "i" or "r" at the end of the test name, for immediate or register respectively. The following conclusions can be drawn:

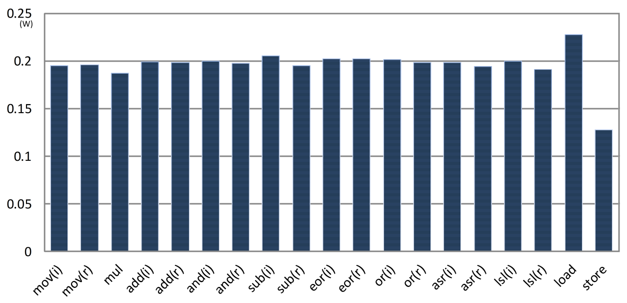

Figure 2: The basic power consumption of ARM11.

• For different arithmetic logic instructions, the processor power consumption is similar. The opcode does not affect the power much because all of the arithmetic logic instructions use the same pipeline stage. (See also [16].) The instruction "MOV(i)" consumes the least power, 0.1947W and "SUB(i)" consumes the most power, 0.2052W. Thus, the maximum difference is 5.45%. Furthermore, the standard deviation (σ=0.00304) divided by the average power ($\overline{P_{AL}}$=0.1990) is 1.53%, so the basic power of different arithmetic logic instructions is very similar.

• The addressing mode does not affect the power very much. For example, the minimum difference, between EOR(i) and EOR(r), is -0.03% and the maximum difference is 5.02%, from "SUB(i)" and "SUB(r)".

• Load consumes the most power. There are not any instruction or data cache misses in the load test because all the target operand addresses are the same. Therefore, the load test runs faster and consumes more power than arithmetic logic functions.

• Store consumes the least power because the instructions per clock cycle (IPC) of store is 0.04. Furthermore, the fact that it takes 25 cycles to finish one store instruction means pipeline stalls happen often. Writing data back to the main memory takes a relatively long time and the cache write buffer has to ensure the coherence of the data cache and the main memory. Consequently, although the processor tries to keep writing data to memory, the pipeline may stall and has to wait until the buffer is empty before writing new data. Moreover, the cache write buffer is only 1-2 words in ARM11 and is easy to fill. Hence, the processor spends most of the time waiting for the cache write buffer and so the power of a store instruction is the lowest.

From this analysis, in order to simplify the model, we assume all arithmetic and logic instructions consume the same basic power in all addressing modes.

On the other hand, cache misses can affect the power and speed of a processor. In order to study how cache misses affect the power consumption, we increased the loop body size in different tests and measured how the power changes with the cache miss rate. Figure 3 shows those results, where both the L1 data and instruction caches are 16KB.

As Figure 3 shows, when the loop body is larger than 16KB,

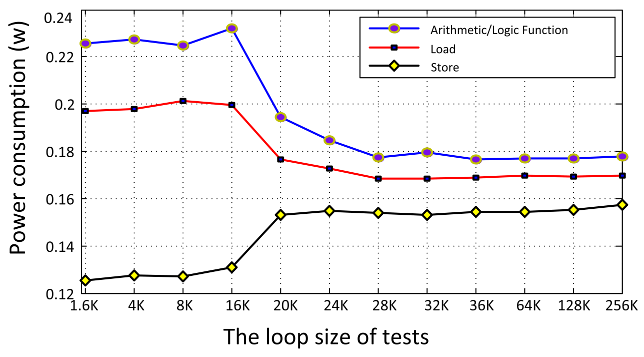

Figure 3: Power consumption versus cache miss rate.

the instruction cache is not sufficiently large to contain it which causes an increase in the cache miss rate, and hence arithmetic/logic and load instructions consume less power. This is due to the processor having nothing to do while then instruction fetch unit reads from the instruction memory. Therefore, the power consumption is lower, but the energy per instruction is higher. As the cache miss rate increases, the processor spends more and more time waiting and pipeline stalls happen more often. The power consumption keeps decreasing until 32K, because all of the old instructions have been moved out of the cache after one loop in the program and each new instruction has to be fetched from main memory. The lowest speed of the tests is determined by the main memory speed. Thus, even though the loop size increases to more than 32KB, the power does not change. The power for an arithmetic/logic instruction is about 0.170W and for load 0.177W.

The behavior of store is quite different. For a cache miss, the power consumption becomes greater than for a cache hit. The reason is that for a cache hit, because of the write buffer, the pipeline stalls often and the IPC is low (0.04). However, for a cache miss, the IPC does not change very much (0.028) and the processor has to fetch new instructions from main memory. Hence, the communication rate with main memory is higher and more data goes though the IO ports. Consequently, a cache miss consumes more power than a cache hit.

It is clear that cache misses can affect the power consumption of a processor. Therefore, the power model has to consider both the effects by instruction types and cache misses.

5. INSTRUCTION LEVEL POWER ANALYSIS AND MODELING

In order to study how the instructions affect the power in combination, a more realistic program environment was created. Figure 4 shows the components and IPC of each basic test. All of the programs are coded manually, allowing to understand the distribution of the program detailed and change it easily. We choose 25% as a step size and the percentage of the arithmetic logic instruction decreases from test 1 to test 9. For example, all of the instructions in test 1 are arithmetic/logic but only 25% of instructions come from arithmetic/logic in test 7, test 8 and test 9. In contrast, the

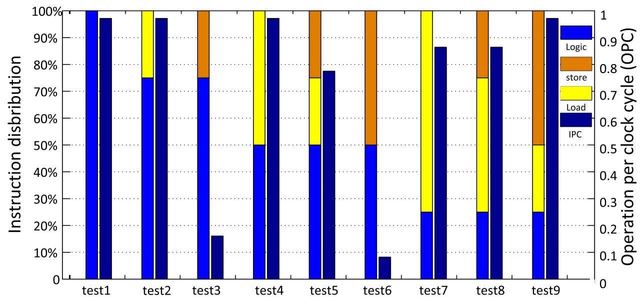

Figure 4: Instruction distribution and OPC of each basic test

load and store instruction percentages increase. Finally, figure 4 shows all of the possibilities. We ignore the 0% logic case because it is unlikely that a program does not have any logic instructions.

The second column shows the IPC for each test. There are a lot of different reasons why the pipeline stalls, such as data dependencies and cache misses. But in all cases, the IPC becomes low and the processor spends more time waiting. Thus, IPC can be used as a parameter to estimate how smoothly a program runs and to reflect the effect of the cache miss rate and pipeline stall rate. There are some complex instructions in ARM assembly which need more than one cycle to finish. When these complex instructions are fetched into the processor, they will be decoded into more operations. For example, a complex instruction may need three clock cycles to finish, therefore the IPC is equal to 0.33, but the processor actually spends time working not waiting. Therefore, we will use operation per clock cycle (OPC) to describe the speed of an application instead of IPC henceforth. For these tests, the number of instructions roughly equals the number of operations: IPC equals OPC.

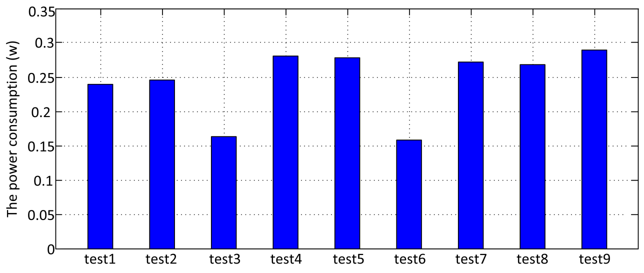

Figure 5: The power consumption of each basic test

Figure 5 shows the power consumption of each basic test. Test3 and test6 consume the least power because the OPC (IPC) is the lowest in these tests and the processor spends more time waiting. Test 9 consumes the most power because the IPC is very high (more than 0.968). It means the processor is extremely busy calculating and has few pipeline stalls. For the tests with similar IPCs, if the test has more logic instructions, it will consume less power, as in test 1 and test 2. Therefore, we use linear regression to generate a model [6] to describe how these factors determine the power.

$$P_{average} = 0.1882 - 0.0601 \times P_{logic} + 0.0081 \times P_{store} + 0.1251 \times OPC, \quad (6)$$

where the $P_{average}$ is the average power consumption of the program, and P_{logic}, P_{store} and OPC are the logic instruction percentage, store percentage and OPC of the program respectively. We assume that all of the instructions come from these three cases, thus after linear regression, the P_{load} is replaced by the equation $P_{logic} + P_{store} + P_{load} = 100\%$. Figure 6 shows the difference error percentage between the

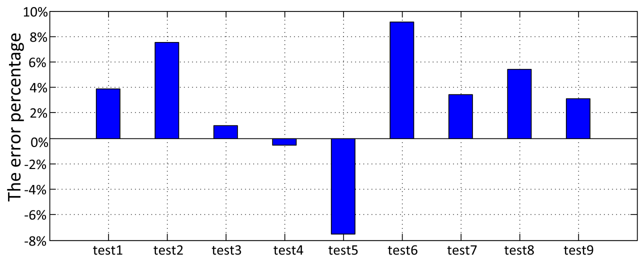

Figure 6: The estimation results for each basic test

model and the measured results. It is clear that all of the tests are estimated accurately, with errors less than 10%.

6. VALIDATION

Six standard benchmarks: Bitcount, Fibonacci, Tak, FIR filter, Quicksort and Tower of Hanoi were used to test the performance of the model. The components of each test are shown in Figure 7. The distribution was generated by the instruction simulator tool gem5 [5]. We do not need a cycle-accurate mode and the basic mode can generate all of the required information quickly. Moreover, the ARM performance counter also can supply this information. Therefore, the distribution is fast and easy to get.

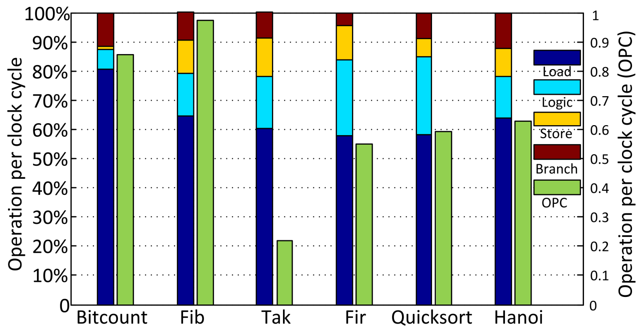

Figure 7: The components of each standard benchmark test program

From Figure 7, the lowest OPC is for Tak (0.22), while for Fibonacci it is more than 0.95. The components of the different tests are also very different. For example, the logic percentage of Fir is less than 60% but it is more than 80% in Bitcount.

Figure 8 shows the measured power (column 1), estimated power from our model (column 2) and the difference percent-

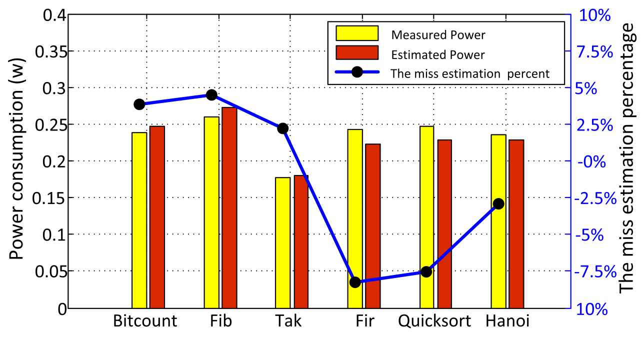

Figure 8: The power consumption of measurement and estimation

age (curve). Tak consumes the least power, 0.1765W, because it has the lowest number of operations compared with the others. Although Bitcount, Fir, Quicksort and Hanoi have different OPC and opcode percentages, they consume similar power. Moreover, the power estimation is very accurate and the estimation errors are less than 9% with an average absolute error of 4.88%.

7. ENERGY MODEL

As discussed above, our model is easy to extend for studying energy. The runtime of the program is measured from the OS, thus the error in estimating the energy consumption of a program is the same as the power estimation error.

7.1 Comparison with Previous Work

Table 2: Comparison with previous work

	Our method	[10]	[12]	[1]	[8]
Fibonacci	4.47%	—	15.58%	—	—
Fir	8.288%	-4.05%	—	11.52%	—
Quicksort	-7.51%	—	11.41%	—	8.98

Table 2 compares previous work and our method. For the Fir test, the method reported in [10] gives a better estimate because it considers the overhead energy. But for Fibonacci and Quicksort, our model gives a better prediction.

Compared with the other models, one benefit of ours is its simplicity because the overhead power of adjacent instructions is not considered. If a model considers the overhead energy, the measurement times will be proportional to the square of the number of the instructions in the instruction set architecture (ISA). There are 51 instructions in ARM assembly language, it would need at least 1300 measurements to cover every potential pair.

Another benefit of our model is that it is easy to create. In order to generate the power model, we use just nine training tests to achieve minimum and maximum errors of -0.56% and 9.15% respectively. However, the energy model for the MeP processor requires sophisticated training tests and considers the standard deviations of every parameter values [6]. The minimum and maximum error of that model are 2% and 16%. Another energy model for the ARM7TDMI uses 60 specialized tests to estimate the coefficients of each energy sensitive factor. On top of this, there are 35 parameters for the model including: the ARM7 instruction set, register bank bit flip, instruction word Hamming distance etc. [3].

We also consider the effect of cache misses and pipeline stalls to some extent. Furthermore, cache misses and data dependency can make the pipeline stall, hence will affect the power and energy consumption of a program. We take OPC into account in terms of these factors and this approach gives an effective method for analyzing them. However, these factors are not considered very much in [10, 12].

7.2 Discussion: Low Energy Software

Power has a close relationship with energy, thus the power model can be extended to study the energy usage of a program. In this subsection, we will consider how the power model might be applied to writing low energy software.

Energy per instruction (EPI) or Energy per operation (EPO) describes the energy efficiency of a microprocessor [7]. Instead of IPC, we use EPO to estimate the energy efficiency as follows:

$$\begin{aligned} EPO &= \frac{Energy}{N} = \frac{P \times T}{N} = \frac{P}{N/T} \\ &= \frac{P}{N/(Cycles \times (1/F))} \\ &= \frac{P}{OPC \times F} \end{aligned} \tag{7}$$

where, N, P and F are the total number of operations in the program, the average power and the frequency of the processor, which is 533MHz in this case, respectively. Combining equations (6) and (7), we get:

$$\begin{aligned} EPO =& \frac{P}{OPC \times F} \\ =& \frac{0.1882 - 0.0601 \times P_{logic} + 0.0081 \times P_{store} +}{OPC \times F} \\ & \frac{0.1251 \times OPC}{OPC \times F} \\ =& \frac{C_1}{OPC \times F} + \frac{0.1251}{F} \end{aligned} \tag{8}$$

where $C_1$is $0.1882 - 0.0601 \times P_{logic} + 0.0081 \times P_{store}$. It is clear that the EPO is inversely proportional to OPC. Therefore, the bigger the OPC is, the less energy is consumed by each operation, if the programs have similar instruction distribution.

Figure 9 shows the energy per operation and OPC of each test, demonstrating the conclusion proposed by equation (8). Furthermore, if pipeline stalls can be reduced, by reducing, for example, the cache miss rate, the energy usage will be better. Consequently, it is important to make the pre-fetch unit and branch predictor run more efficiently to reduce the pipeline stalls. The following ideas may help to write energy efficient code:

• Try to use arrays and simpler loops as much as possible. Lists, and trees may have complex structure which can confuse the pre-fetcher.

• Avoid long jumps. Pre-fetchers detect jumps only in a certain range because detecting longer jumps require more hardware storage.

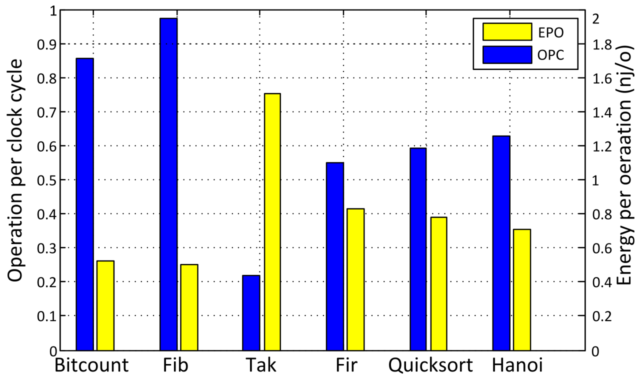

Figure 9: The energy per operation VS operation per clock cycle

• Make loop and jump more easily predictable by the branch predictor.

• Try to re-use memory locations because these data has a high chance of being in he data cache.

However, the pre-fetch is not always helpful because if the pre-fetch unit chooses a wrong address it will have two problems: (1) it wastes the memory bandwidth in fetching useless data, (2) if new data is fetched into the cache, some useful data may evicted because of the limitation of cache size. Thus, the actions of the pre-fetch and branch predictor should be considered more in software design.

8. CONCLUSIONS

This paper presents an instruction-level power model for a single core, RISC processor ARM1176JZF-S. Firstly, we do not distinguish the different types of arithmetic and logic instructions because the power consumption of them is nearly the same. Furthermore, the instructions are divided into three clusters based on their power and behaviors: arithmetic/logic, load, and store. Secondly, the speed of a program can affect the power. Even though two different programs have the same type and number of instructions, the power and energy will be different if they have different OPCs. Thus, we take the OPC into consideration and it reflects the factors which can make the pipeline stall, such as cache miss and data dependency. Finally, we present a model that takes both the components and the OPC of a program into account. This model is very concise because it does not consider the effect of two adjacent instructions. The maximum error is -8.28% and the average absolute error of all tests is 4.88%. Compared with previous models that consider the effect of adjacent instructions, our model requires far less input data while retaining a high degree of accuracy. Finally, we show that energy per operation (EPO) is inversely proportional to operation per clock cycle (OPC).

9. REFERENCES

[1] J. M. Acevedo Pati O. and C. A. A.J. Static simulation: A method for power and energy estimation in embedded microprocessors. In *Circuits and Systems*

(MWSCAS), 2010 53rd IEEE International Midwest Symposium on, pages 41 –44, aug. 2010.

[2] ARM. Arm1176jzf-s technical reference manual, 2009.

[3] M. Bazzaz, M. Salehi, and A. Ejlali. An accurate instruction-level energy estimation model and tool for embedded systems. *Instrumentation and Measurement, IEEE Transactions on*, 62(7):1927–1934, 2013.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pages 83 –94, june 2000.

[5] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli. Accuracy evaluation of gem5 simulator system. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–7, 2012.

[6] L. Gauthier and T. Ishihara. Processor energy characterization for compiler-assisted software energy reduction. *JECE*, 2012:8:8–8:8, Jan. 2012.

[7] E. Grochowski and M. Annavaram. Energy per instruction trends in intel microprocessors. *Technology@ Intel Magazine*, 4(3):1–8, 2006.

[8] S. Hao, D. Li, W. G. Halfond, and R. Govindan. Estimating android applications' cpu energy usage via bytecode profiling. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 1–7. IEEE, 2012.

[9] M. E. A. Ibrahim, M. Rupp, and H. A. H. Fahmy. A precise high-level power consumption model for embedded systems software. *EURASIP J. Embedded Syst.*, 2011:1:1–1:14, Jan. 2011.

[10] N. Kavvadias, P. Neofotistos, S. Nikolaidis, K. Kosmatopoulos, and T. Laopoulos. Measurements analysis of the software-related power consumption in microprocessors. In *Instrumentation and Measurement Technology Conference, 2003. IMTC '03. Proceedings of the 20th IEEE*, volume 2, pages 981 – 986 vol.2, may 2003.

[11] B. Klass, D. Thomas, H. Schmit, and D. Nagle. Modeling inter-instruction energy effects in a digital signal processor. In *Power Driven Microarchitecture Workshop in conjunction with the 25th International Symposium on Computer Architecture*, volume 2, pages 1094 –1097, Oct. 1998.

[12] S. Penolazzi, L. Bolognino, and A. Hemani. Energy and performance model of a sparc leon3 processor. In *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, pages 651 –656, Aug. 2009.

[13] Samsung. Samsung s3c6410 mobile pprocessor, Apr 2008.

[14] D. Sarta, D. Trifone, and G. Ascia. A data dependent approach to instruction level power estimation. In *Low-Power Design, 1999. Proceedings. IEEE Alessandro Volta Memorial Workshop on*, pages 182 –190, Mar. 1999.

[15] Y. S. Shao and D. Brooks. Energy characterization and instruction-level energy model of inteląŕs xeon phi processor. *International Symposium on Low Power Electronics and Design (ISLPED)*, 2013.

[16] S. Sultan and S. Masud. Rapid software power estimation of embedded pipelined processor through instruction level power model. In *Performance Evaluation of Computer Telecommunication Systems, 2009. SPECTS 2009. International Symposium on*, volume 41, pages 27 –34, July 2009.

[17] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):437–445, Dec. 1994.

[18] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 340–345, New York, NY, USA, 2000. ACM.