Abstract

Objective: The power factor parameter plays an imperative role in design specification of embedded systems as a large number of applications that run on these systems is power critical. To develop a power efficient system, it is essential to know how power is consumed throughout the system. Methods: A proper power analysis tool is designed by partitioned the application and measures the power consumption in each partitioned block. Findings: This tool can be very helpful in the verification process of any embedded systems applications to meet the power constraints. Applications: This tool can be applied in mobile cloud computing environment to determine the power consumed task to get offloaded onto the cloud.

Keywords: Control Flow Graph, Embedded System, Power Optimization, Power Profiling

1. Introduction

The recent years has shown a tremendous increase in the use of embedded systems. Recent statistics show that the number of embedded computer systems in the USA is more than the population of the country. The use of embedded systems has become infinite, because every day new products are being introduced into the market which uses the embedded computers. These days, microprocessors, microcontrollers are available at low costs than they used to be a decade earlier. It is hence easier to buy the hardware and write our own custom software for it. Complications and issues arise when these systems are used by the user. The major issue being the design issue. Proper design goals are to be set and it has to be made sure that the embedded systems are up to the standards. Basically, an embedded system can be able to run dedicated logics by taking into consideration the energy and memory consumption. Majorly the energy consumption is the important aspect which needs to be focused. These days’ devices are battery operated. To meet the low power requirements, researchers and scholars have been putting efforts on power estimation procedures. As a result, there are many techniques which have been proposed to help designers to get an early insight of system’s power dissipation behavior of the embedded computers. This paper deals with the development and validation of instruction level power profiling tool for x86 based processor. The profiling tool can be used in determining the power cost of embedded software. The power analysis tool takes assembly file as input to carry out the static instruction level power modelling.

Power profiling gives knowledge about the behaviour of computer applications based on energy consumption and paves the way to improve their energy efficiency. Power Profiling can be classified into two broad categories i. Measurement based profiling and ii. Model based profiling. In measurement based method the profiling are performed using device circuits such as oscilloscope etc., The Model-Based Profiling has function tables which has predefined instructions associated with individual CPU power consumption. Further the model-based profiling can be divided into low-level models and high-level models.
2. Methodology and Approach

2.1 Assembly Code Extraction

The code which runs in the CPU is the machine code. These codes are patterns of 0’s and 1’s in the words of the memory. Regardless of the means of achieving this machine code, it has to be placed in the right address for the dispatch unit of the CPU to run the instructions. Normally, human coders find it difficult to write this machine code directly as it is difficult to read and understand. So, people use the assembly language. They are comparatively easy to read and understand. They correspond precisely to the instructions which the CPU executes, and can be translated into such instructions. Assembler is the one which takes care of this conversion of assembly code to the machine code. A code written in a high level language has to be compiled before execution. The compiler produces either an assembly language programme as an intermediate step or else the machine code directly. The proposed work requires assembly code for profiling purpose. The code written in a high level language, java for instance, is taken. Figure 1 shows a sample java code and assembly code generated as a result of execution of objdump.

The Figure 2 shows the extracted and necessary information obtained from the assembly code. They are in the order of address: opcode: operands.

2.2 Splitting into Blocks

Now as the assembly code has been successfully extracted, the next step involves splitting up of the code into blocks. The second column in Figure 2b is the block number column. The blocks are numbered as 1, 2, etc. as shown in Figure 2b.

The algorithm involved in splitting of the code into blocks is as follows:

Algorithm Block Separation

\[
\text{BlockNumber} \leftarrow 0 \\
\text{visited}[\cdot] \leftarrow \text{size of assembly code} \\
\text{Loop until all statements are visited} \\
\text{currentIndex} \leftarrow 0 \\
\text{Loop from currentIndex to length of assembly code} \\
\text{CurrentStatement} \leftarrow \text{read a statement} \\
\text{if currentStatement is branch statement} \\
\text{increment blockNumber} \\
\text{visited[currentIndex]} \leftarrow \text{blockNumber} \\
\text{currentIndex} \leftarrow \text{branchDestination} \\
\text{else} \\
\text{visited[currentIndex]} \leftarrow \text{blockNo}
\]

Figure 1. Sample Code written in C & assembly code.
2.3 Control Flow Graph Construction

A control flow graph is a graph in which each node in the graph represents a basic block. A basic block is something where there are no jumps or jump targets. A jump targets start a block and jumps end a block. After separating our code into blocks, we need to construct a control flow graph to know the control flow between the blocks. Figure 3 shows the Control Flow Graph (CFG) for the code. There are totally four blocks. The algorithm for flow graph construction is:

\[
\text{BlockNumber} \leftarrow \text{visited}[0] \\
\text{block} \leftarrow \text{create a new block object} \\
\text{block.start} \leftarrow 0 \\
\text{block.BlockNumber} \leftarrow \text{blockNumber} \\
\text{Loop from } i = 0 \text{ to } (\text{assembly code length}+1) \\
\]

If \( i == \) assembly code length

\[
\text{Block.end} \leftarrow i-1 \\
\text{Add block to blockList} \\
\]

Else if \( \text{visited}[i] \neq \text{currentIndex} \\
\text{Block.end} \leftarrow i-1; \\
\text{Add block to blockList} \\
\text{Block} \leftarrow \text{create a new block} \\
\text{Block.start} \leftarrow i \\
\text{BlockNumber} \leftarrow \text{visited}[i] \\
\text{block.BlockNumber} \leftarrow \text{blockNumber} \\
\]

Algorithm ConstructFlow

\[
\text{Loop from } i = 0 \text{ to length of Assembly code} \\
\text{CurrentStatement} \leftarrow \text{read a statement} \\
\text{if currentStatement is branch statement} \\
\text{blockNumber} \leftarrow \text{visited}[i] \\
\]

Figure 2a. Extracting information from assembly code. (b) Code divided into blocks.
currentBlock ← block with blockNumber from blockList
currentBlock.FlowList ← destination of current branch statement

2.4 Base Cost Estimation

Now as the CFG has been constructed, the base cost for each blocks has to be constructed. The base cost or the power consumed by each blocks can be calculated as follows:

\[ \text{Energy} = \text{Frequency} \times \text{Execution Time} \]

The CPU execution time is the product of number of instructions executed, average cycles per instruction and CPU clock cycle. A block consists of a number of instructions to be executed. This constitutes the \( I \) value. The average instruction executed takes a number of cycles per instruction (CPI) to be completed. CPU has a fixed clock cycle time, \( C = 1/\text{Clock Rate} \) which is measured in seconds/cycle. The clock rate is assumed to be 1.995.

\[ T = I \times CPI \times C \]

Where, \( T = \text{CPU Execution Time in seconds} \); \( I = \text{Number of Instructions Executed} \)

\( CPI = \text{Average Cycles per Instruction} \); \( C = \text{CPU Clock Cycle} \)

The energy consumed is the product of CPU execution time and the power value for 1 core. With respect to the frequency value, i.e the clock rate value, the value for 1 core turns out to be 68.23. The CPU execution time is obtained as a result of previous calculation. Thus, the energy consumption is calculated as \( E = P \times T \). The power consumed by each block is calculated. The power consumption values for the considered example are shown in Table 1.

### Table 1. Power Consumption Values

| Block | Adjacency List | Power Consumption |
|-------|----------------|-------------------|
| 1     | 2, 3           | 1.1286165413533835E–6 |
| 3     | 4              | 1.026015037593985E–7 |
| 2     | 4              | 5.130075187969925E–8 |
| 4     |                | 1.026015037593985E–7 |

3. Conclusions and Future Works

Power profiling is an important factor to be considered while designing an embedded system. The power profiling is hence done for the processors having x86 architectures. The assembly code is extracted for every high level code which is compiled and split into blocks to find the energy consumption of the block. This profiling paves way for several other techniques for power optimization, scheduling etc. Further this tool can be applied in mobile cloud computing environment to determine the power consumed task to get offload onto the cloud.

4. References

1. Flinn J, Satyanarayanan M. Powerscope: a tool for profiling the energy usage of mobile applications. Mobile computing systems and applications, Proceedings, WMCSA’99 Second IEEE workshop on IEEE. 1999; p. 2–10.
2. Sun Y, Zhu T, Zhong Z, He T. Energy profiling for mPlatform. Proceedings of the 7th ACM conference on embedded networked sensor systems, ACM. 2009; p. 407–8.
3. Brooks D, Tiwari V, Martonosi M. Vancouver: Watch: a framework for architectural-level power analysis and optimizations. Appears in the proceeding of 27th International Conference on Computer Architecture. 2000; p. 83-94.
4. Burger D, Austin TM. The Simple Scalar tool set, version 2.0. ACM SIGARCH Computer Architecture News. 1997; 25(3):13–25.
5. Tiwari V, Malik S, Wolfe A. Power analysis of embedded software: a first step towards software power minimization. IEEE Trans Very Large Scale Integrating (VLSI) System. 1994; 2(4):437–45.
6. Qu G, Kawabe N, Usami K, Potkonjak M. Function-level power estimation methodology for microprocessors. Los Angeles, California: Proceedings of the 37th annual design automation conference, ACM. 2000; p. 810–3.
7. Valvano J. Cengage Learning: Embedded microcomputer systems: real time interfacing, 3rd (edn). 2010; p. 240.
9. Chen DR, Chiang KF. Cloud-based power estimation and power-aware scheduling for embedded systems, Computer and Electrical Engineering. 2015; 47:204-21.
10. Blagodurov S, Fedorova A. User-level scheduling on NUMA multicore systems under Linux. Linux symposium. 2011; p. 81-92.
11. Rogers R, Lombardo J, Mednieks Z, Meike B. O’Reilly Media, Inc.: Android application development: programming with the Google SDK, 1st (edn). 2009.
12. Smith A. Washington DC: Pew Research Centre: Smartphone ownership. 2013.
13. Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-the-art and research challenges. Journal of Internet Services Application. 2010; 1(1):7–18.
14. Blume H, Becker D, Rotenberg L, Botteck M, Brakensiek J, Noll TG. Hybrid functional-and instruction-level power modeling for embedded and heterogeneous processor architectures. Journal of System Architecture. 2007; 53(10):689–702.
15. Chung YF, Lin CY, King CT. Aneprof: energy profiling for android java virtual machine and applications. IEEE 17th international conference on parallel and distributed systems (ICPADS), IEEE. 2011; p. 372–9.
16. Jung W, Kang C, Yoon C, Kim D, Cha H. DevScope: a non-intrusive and online power analysis tool for smart phone hardware components. Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software co-design and system synthesis, ACM. 2012; p. 353–62.
17. Shye A, Scholbrock B, Memik G. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. Proceedings of the 42nd annual IEEE/ACM international symposium on micro-architecture, ACM. 2002; p. 168–78.
18. Srinath SL, Pillai AS. Adaptive Interplay of DVS and DPM for Power Consumption Reduction in Real-Time Embedded Processors. Indian Journal of Science and Technology. 2016; 9(30):1-6.
19. Devasena CL. IPv6 Low Power Wireless Personal Area Network (6LoWPAN) for Networking Internet of Things (IoT) – Analyzing its Suitability for IoT. Indian Journal of Science and Technology. 2016; 9(30):6–11.