Developing an Environment for Embedded Software Energy Estimation

S. Nikolaidis, A. Chatzigeorgiou[1], T. Laopoulos

Department of Physics,Aristotle University of Thessaloniki,
[1]Department of Applied Informatics, University of Macedonia,
54124, Thessaloniki, Greece

Abstract

The instruction-level energy consumption modeling method proposed in [6] is evaluated and completed. According to the proposed method the base and inter-instruction energy costs of the ARM7 embedded processor as well as the energy cost due to different values in the instruction parameters are modeled. These models can be used in the estimation of the energy consumed by the processor to execute real software programs. A software tool has been developed for this estimation. In this article the energy models derived for the instructions of ARM7 embedded processor are analyzed and the energy estimation framework is presented.

1. Introduction

A large number of embedded computing applications are power or energy critical, that is power constraints form an important part of the design specification. Early work on processor analysis had focused on performance improvement without determining the power-performance tradeoffs. Recently, significant research in low power design and power estimation and analysis has been developed.

Embedded software power modeling techniques are distinguished into two main categories: a) physical *measurement-based* and b) *simulation-based* ones. In *simulation-based* methods, energy consumed by software is estimated by calculating the energy consumption of various components in the target processor through simulations. The main drawback of these simulation-based techniques is the need of information about the circuit level design of the processor which is not usually available. In *measurement-based* approaches [1-3], the energy consumption of software is characterized by data obtained from real hardware. The advantage of measurement-based approaches is that the resulting energy model proves close to the actual energy behavior of the processor.

In measurement techniques, a common practice is to associate instructions running on the processor with their corresponding energy cost. A measuring environment has been proposed by the authors in [7] for the measurement of the instantaneous current of the processor during the execution of the instructions and an instruction-level energy consumption modeling methodology has been proposed [8] aiming in the creation of highly accurate models. In this paper the results of our experiments are presented and the achieved accuracy of our method is given. Also, the way of the implementation of a software tool for the estimation of the energy consumed for the execution of programs is described.

2. Instruction-level energy modeling

The energy consumed during the execution of instructions can be distinguished in two amounts. The base cost, which is the energy amount needed for the execution of the operations which are imposed by the instructions, and the inter-instruction cost which corresponds to an energy overhead due to the changes in the state of the processor provoked by the successive execution of different instructions. Measurements for determining these two energy amounts for each instruction of the ARM7TDMI processor were taken and presented in [4]. However the base costs in [4] were for specific operand and address values (zero operand and immediate values and specific address values to minimize the effect of 1s). This base cost is called *pure* base cost.

We have observed in our measurements that there is a strong dependency of the energy consumption of the instructions on the values of their parameters (operand values, addresses). To create accurate models this dependency has to be determined. Additional measurements were taken to satisfy this necessity. By incorporating these effects in our models the proposed method keeps its promised accuracy while it becomes very attractive since it can be easily implemented in software as an estimation tool.

Making some appropriate experiments we observed that the effect of each energy-sensitive factor on the energy cost of the instruction is independent of the effect of the other factors [4]. The distortion of our results from this conclusion is, most of the time, less than 2-3% and only in some marginal cases becomes more than 7%. According to this conclusion, the effect of the energy-sensitive factors can simply be summed to give the total energy amount.

Other sources of energy consumption are conditions of the processor, which lead to an overhead in clock cycles because of the appearance of idle cycles. This is the case of the appearance of pipeline stalls. The effect of such cases on the energy consumption was measured and modeled.

According to the above, the energy, E_i, consumed during the execution of the i instruction can be modeled as:

$$E_i = b_i + \sum_i a_{i,j} N_{i,j} \qquad (1)$$

where b_i is the pure base cost of the i instruction, $a_{i,j}$ and $N_{i,j}$ is the coefficient and the number of 1s of the j energy-sensitive factor of the i instruction, respectively.

Having modeled the energy cost of the instructions, the energy consumed for running a program of n instructions can be estimated:

$$PE = \sum_1^n E_i + \sum_1^{n-1} O_{i,i+1} + \sum \varepsilon \qquad (2)$$

where $O_{I,j}$ is the inter-instruction cost of the instructions i and j, and ϵ is the cost of a pipeline stall.

3. Pure base cost and inter-instruction cost models - results

The completed models for the instruction-level energy consumption of the ARM7TDMI created according to the proposed methodology can be found in [5]. Thousand experiments corresponding to the execution of loops of instruction instances on the processor to realize the appropriate processor conditions referred in [5] have been performed. For the measurement of the instantaneous current of the processor the measuring environment proposed in [7] was employed. Pure base costs of all the instructions and for all the addressing modes are given. Since the number of the possible instruction pairs (taking into account the addressing modes) is enormous, groups of instructions and groups of addressing modes according to the resources they utilize, have been formed and inter-instruction costs have been given only for representatives from these groups. In this way we keep the size of the required model values reasonable without significant degradation of the accuracy (less than 5% in the inter-instruction cost by using only representative instructions).

Most of the values of the pure base costs present a difference less than 20% in the energy of the instructions which are executed in the same number of cycles. Most of the values of the inter-instruction costs have negative sign as it was expected. The contribution of the inter-instruction costs, as they are calculated according the proposed method, remains small. As it can be observed by our models most of the inter-instruction costs are less than 5% of the corresponding pure base costs while almost all the cases are covered by an 15% percentage.

To determine the accuracy of the method a number of programs with various instructions were created. In these instructions the effect of energy sensitive factors wasn't taken into account. The error was found to be up to 1.5%.

4. Energy dependency on instruction level parameters – results

The dependency of the energy of the instructions on the values of the instruction parameters and the operands, called energy sensitive factors, was also studied. Energy depends on the number of 1s in the word structures of these entities. The energy-sensitive factors are the register numbers, the register values, the immediate values, the operand values, the operand addresses and the fetch addresses of the instructions.

This energy dependency can be approximated with sufficient accuracy by linear functions. Coefficients should be derived for each instruction for every energy sensitive factor. However, appropriate grouping of the instructions is used to keep reasonable the number of required coefficients to increase the applicability of the method without significant loss in the accuracy.

The grouping of the instructions for the derivation of the coefficients and the corresponding measurements are presented in [5]. According to the results the linear dependency mentioned above is obvious. Some results are presented here. In Figure 1 the effect of the register number for data-processing instructions in immediate addressing mode is presented. The actual physical measurements versus estimated energy values for the ADC instruction in scaled register offset addressing mode is shown in Table 1 where the achieved for the selected coefficient accuracy resulted is also given. The error is less than 3%. Such error values characterize all the selected coefficients.

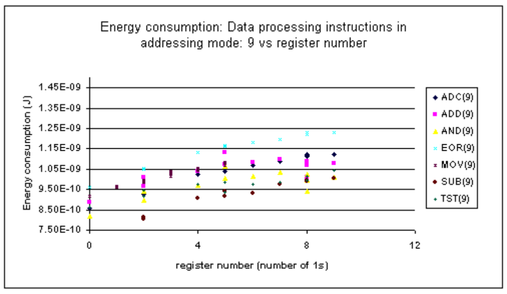

Fig. 1 The effect of register number for data-processing instructions

Table 1: Actual physical measurements versus estimated energy values (nj) for the ADC instruction

#1s	Estim.	Meas.	% error
0	0.874	0.855	2.20
2	0.936	0.929	0.74
2	0.936	0.924	1.23
5	1.028	1.040	1.19
6	1.059	1.067	0.77
8	1.121	1.119	0.16
8	1.121	1.124	0.28
9	1.151	1.124	2.47
7	1.090	1.088	0.17
5	1.028	1.054	2.46
8	1.121	1.114	0.61
4	0.997	1.023	2.51

To evaluate the absolute accuracy of our modeling approach, real programs were used as benchmarks. The corresponding assembly listings were extracted from C programs by utilizing the facilities of the *armcc* tool, shipped with the ARM ADS software distribution. The energy consumption at each clock cycle is measured and estimations for the instructions are produced based on the derived models. The overall energy dissipation is calculated by equation (2) in order to sum up all the individual contributions that relate to variations in the energy consumption at the instruction level. According to our results the error of our approach in real life programs was found to be less than 5%.

5. Software energy estimation framework

This section describes the main features of the software power estimation framework that has been developed for the ARM7TDMI processor. The software (hereafter "Energy Profiler") employs the instruction-level power models that have been presented, enabling exploration of various alternatives of a given program, in order to optimize its power consumption. The Energy Profiler receives as input the trace file of executed assembly instructions and estimates the base and inter-instruction energy cost of the program.

5.1 *Kernel of the Program*

The kernel of the program, which is written in the ANSI *C* programming language [9], hard-codes the derived instruction-level energy models. An overview of the energy estimation process is shown in Figure 2.

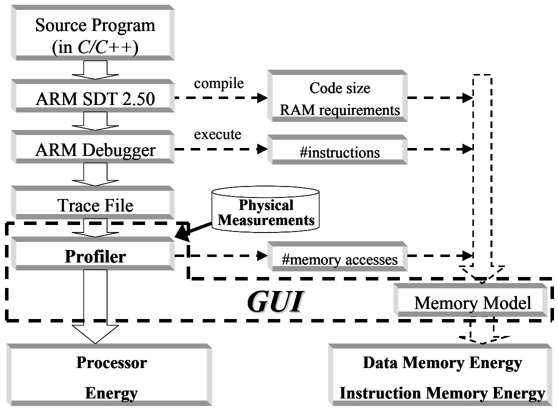

Fig. 2: Energy estimation process

Energy estimation is initiated by compiling the source file of the program under study with one of the available compilers of the ARM processor family [10]. Compiling the program provides both the code size and the minimum RAM requirements for the data memory. Next, the execution of the code using the Debugger generates the trace file and provides the number of executed assembly instructions.

The trace file which is then parsed serially by the Energy Profiler, contains two kind of entries: *Memory lines* that indicate an access to the data or instruction memory for fetching data or opcodes, respectively, and *Instruction lines* which indicate the conditional execution of an assembly instruction.

Each time the profiler identifies an instruction line it calculates the energy consumption by proceeding to the following main steps:

Step 1: Identification of instruction category
Step 2: Identification of specific instruction
Step 3: Identification of addressing mode
Step 4: Base Energy Cost calculation
Step 5: Inter-instruction Energy Cost calculation
Step 6: Modification of base cost according to Energy Sensitive Factors

Concerning the first step, the profiler looks for specific patterns in the operation code bits. To facilitate the organization and retrieval of information, the search for the specific instruction category, is performed by traversing a binary tree, in which each node corresponds to one bit of the opcode. Each node has two descendants: the path to the left subtree is followed if the corresponding bit of the parent node is 0, while the right subtree is followed otherwise. The leaves of the tree contain the information that is being sought. For example, the binary tree for deciding the instruction category based on the values of several bits is shown in Figure 3.

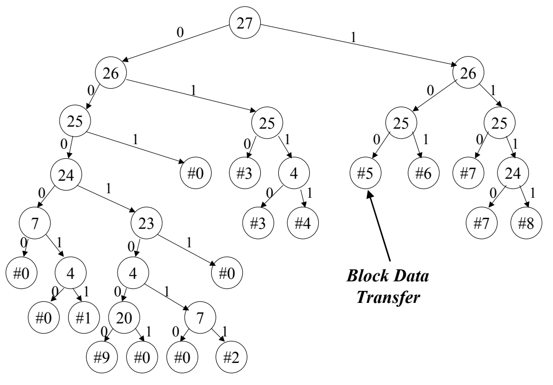

Fig. 3: Binary Tree for Instruction Categories

Concerning the implementation of the binary trees in *C*, each node is a structure (*struct*) [9] with four fields: a) *data* which holds the node's data, b) *bit* which holds the value of the corresponding bit in the opcode, c) *two pointers* to the left and right subtrees.

Constructing the trees is not a trivial task, since in many cases the trees consists of a few tens of nodes, each of which should receive the correct values for data and bit variables. Hard-coding the information by hand is an error-prone process. For this reason, the proposed approach separates the process of implementing the trees from entering data into them. In this way, the process can partly be automated and data can be easily verified since information is not 'hidden' into the code.

The information that is to be organized in each binary tree is stored in a *string*, which represents the *pre-order traversal* (root-left-right) of the tree. For example, the string that corresponds to the binary tree of Figure 2 is: *"1B1A191807#0004#00#01170414#09#0007#00#02#00#0019#0304#03#041A19#05#0619#0718#07#08$"*

Each node is represented in the string by two hexadecimal digits, which correspond to the *bit* variable of the node. If the digits are preceded by the symbol '#', the corresponding node is a leaf node. In that case, the information represented by the two digits is stored to the *data* variable of the node. The symbol '$' indicates the end of the string and ends the construction of the tree. The following recursive pseudocode constructs a binary tree from the information stored in a string. Essentially, the algorithm traverses in pre-order a binary tree that is symbolically represented in the string and constructs in parallel the binary tree as linked nodes.

```
Function createTree
Input: string
Output: root node of the binary tree
{
   Read next two digits (HEX_INFO)
   if $, exit

   Create node x
   if x is leaf_node
      place HEX_INFO to x.data
   else
      place -1 to x.data
      place HEX_INFO to x.bit
      createTree(rest_of_string)
      place returned node to
         x.left_child

      createTree(rest_of_string)
      place returned node to
         x.right_child
   end_if

   return x
}
```

Once the specific instruction category (represented by a unique number) is found, it is used as index in an array of pointers to trees, and the selected tree is employed to obtain the specific instruction that has been executed. Next, a third binary tree (specific for each instruction) is traversed, in order to extract the addressing mode of the executed assembly instruction.

Finally, the number corresponding to the addressing mode and the instruction category are used as indices in a two-dimensional table that contains the physical measurements for the base energy costs, in nJ.

Concerning the inter-instruction energy cost calculation, which is associated with the execution of adjacent assembly instructions, the groups that have been determined during the derivation of the corresponding models have been employed in the profiler. The inter-instruction costs have been placed in two-dimensional arrays, while the information extracted during base cost calculation (instruction category, type, addressing mode) has been used both for selecting the appropriate array as well as for the indices that specify an element. To be able to compare two adjacent instructions, after parsing each instruction line of the trace file, information is placed on a temporary buffer.

The final step consists of the modification of the pure base cost of each instruction according to the energy sensitive factors described earlier. The factors that have been implemented are: register numbers, immediate values, operand addresses and instruction fetch addresses. A separate function for each energy sensitive factor receives as input the number of 1s of the word space and returns the amount of energy that has to be added, considering the corresponding coefficient. The kernel sums up the results for all energy factors and modifies accordingly the pure base cost for each instruction line.

In parallel, the number of executed instructions and the code size are used as input to a memory power model (developed by an industrial vendor) in order to calculate the energy consumption of the instruction memory (Fig. 2). In the same way, the number of data memory accesses and the minimum RAM size are used to compute the energy consumption of the data memory. These calculations are performed once the complete trace file has been parsed.

5.2 *Graphical User Interface*

The program includes a Graphical User Interface (GUI) that has been developed in Java. The GUI provides access to the kernel of the program for parsing an input trace file and displays the generated results in multiple tables and graphs. Among others, profiling results are displayed as dissipated energy distribution among system components, instruction categories and main instruction types. The GUI is also capable of comparing results for two or more trace files, thus enabling a comparative analysis of several programs, which aids in exploring the optimum solution of the design space.

The interface between the GUI and the kernel of the program is built upon an XML (Extensible Markup Language) document that is generated by the parser and is being read by the GUI. The XML format provides the possibility of using tags and attributes for delimiting pieces of data, leaving interpretation to the client application. XML data are not written on any temporary file; rather data are being transferred on the fly from the kernel of the program to the GUI, via the standard input and output streams. A sample screen of the GUI with comparative results for two programs is shown in Figure 4.

Conclusions

In this article our embedded software energy consumption estimation methodology was evaluated. All the factors which affect the energy consumed for the execution of a software structure were taken into account. An error up to 5% was found. A software energy estimation framework has been developed based on the derived models.

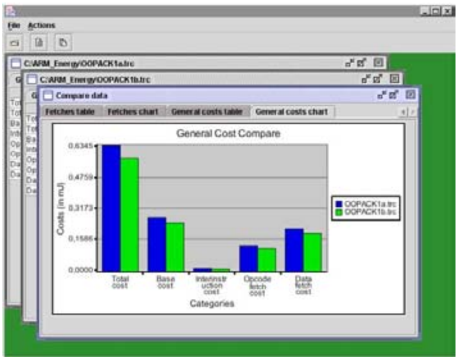

Fig. 4: Sample screen of the GUI

References

[1] V. Tiwari, S. Malik and A. Wolfe, "Power Analysis of Embedded software: A First Step Towards Software Power Minimization", *IEEE Trans. on VLSI Systems*, pp. 437-445, Dec. 1994

[2] S. Steinke, M. Knauer, L. Wehmeyer, P. Marwedel, "An Accurate and Fine Grain Instruction-Level Energy Model supporting Software Optimizations," in Proc. of PATMOS, Switzerland, Sept 2001.

[3] N. Chang, K. Kim, and H. G. Lee, "Cycle-Accurate Energy Consumption Measurement and Analysis: Case Study of ARM7TDMI," *IEEE Trans. on VLSI Systems*, pp. 146-154, Apr. 2002.

[4] S. Nikolaidis, N. Kavvadias, P. Neofotistos, "Instruction level power measurements and analysis", IST-2000-30093/EASY Project, Deliverable D15, Sept 2002.

[5] S. Nikolaidis, N. Kavvadias, P. Neofotistos, "Instruction level power models for embedded processors", IST-2000-30093/EASY Project, Deleverable D21, Dec 2002. Web site: easy.intranet.gr

[6] S. Nikolaidis, Th. Laopoulos, "Instruction-level Power Consumption Estimation of Embedded Processors for Low Power Applications," Proc. of IDAACS, pp.139-142, Foros, Ukraine, 2001.

[7] T. Laopoulos, P. Neofotistos, K. Kosmatopoulos, S. Nikolaidis, "Measurement of Current Variations for the Estimation of Software-related Power Consumption," accepted in IEEE Trans. on Instrumentation and Measurement.

[8] S. Nikolaidis et all, "Instrumentation set-up for Instruction level Power Modeling," in Proc. of PATMOS, Sevilla, Sept.2002

[9] Kernighan B., Ritchie D.M., *The C Programming Language*, Prentice-Hall, Upper Saddle River, NJ, 1988

[10]ARM Developer Suite, http://www.arm.com/devtools/ads?OpenDocument