Presented at the *International Conference on VLSI Design*, Banglore, India, January 1996.

Instruction Level Power Analysis and Optimization of Software

Vivek Tiwari, Sharad Malik, Andrew Wolfe
Dept. of Electrical Engineering
Princeton Univ.

Mike Tien-Chien Lee
Fujitsu Labs of America
San Jose, CA

Abstract

Software constitutes a major component of today's systems, and its role is projected to continue to grow. This motivates the need for analyzing power consumption from the point of view of software - something that circuit and gate level power analysis tools are inadequate for. This paper describes an alternative, measurement based instruction level power analysis approach that provides an accurate and practical way of quantifying this cost. This technique has been applied to three commercial, architecturally different microprocessors. This paper presents an overview of the salient results of these analyses. The ability to evaluate software in terms of power consumption makes it feasible to develop tools and techniques for low power software. Several ideas in this regard as suggested by the power analysis of the subject microprocessors are also summarized.

1 Motivation

The increasing popularity of power constrained mobile computers and embedded computing applications drives the need for analyzing and optimizing power in all components of a system. Software constitutes a major component of systems where power is a constraint. In light of this, there is a clear need for considering the power consumption in systems from the point of view of software. Software impacts the system power consumption at various levels of the design. However, in order to systematically analyze and quantify this impact, it is important to start at the most fundamental level - the instruction level. Just as logic gates are the fundamental units of computation in digital hardware circuits, instructions can be thought of as the fundamental unit of software. Accurate modelling and analysis at this level is the essential capability needed to quantify the power costs of higher abstractions of software.

Traditional power analysis tools are not suited for power analysis of software. Accurate tools exist only for the lower most levels of the design, but they can be slow and cumbersome to use, and often these cannot even be applied, since the lower level information is not available to the software or embedded system designer.

These problems can be overcome if the current being drawn by the CPU during the execution of a program is physically measured. An instruction level power analysis technique based on this idea has recently been developed [5]. This technique helps in formulating instruction level power models that provide the fundamental information needed to evaluate the power cost of entire programs. This technique has so far been applied to three commercial, architecturally different microprocessors:

- Intel 486DX2-S Series, 40MHz, 3.3V. A CISC processor based on the x86 architecture. It is widely used in mobile and desktop PCs [5].
- Fujitsu SPARCliteMB86934, 20MHz, 3.3V. A 32-bit, SPARC based RISC processor, specially designed for embedded applications [3].
- Fujitsu proprietary DSP, 40MHz, 3.3V. A new implementation of a Fujitsu DSP architecture, used in several embedded DSP applications [2].

The purpose of this paper is to provide an overview of the salient results of the power analysis of these processors. The power models for the individual processors, as well as comparisons between them provide useful insights into power consumption in microprocessors. In addition, the information provided by the instruction level analysis can guide higher-level software design decisions. It can also be directly used by automated tools like compilers, code generators and code schedulers for generating code targeted towards low power.

2 Analysis Methodology

The average power, P, consumed by a microprocessor while running a program is given by: $P = I \times V_{CC}$, where I is the average current and V_{CC} is the supply voltage. The energy, E, consumed by a program is further given by: $E = P \times N \times \tau$, where N is the number of clock cycles taken by the program and τ is the clock period. Thus, the ability to measure the current drawn by the CPU during the execution of the program is essential for measuring its power/energy cost.

A simple and practical technique has been developed to measure this current. The power supply connections to the CPU are isolated from the rest of the system. The program under consideration is put in an infinite loop and executed on the CPU. The current waveform will now be periodic and an average value can be visually obtained from a standard off the shelf digital ammeter. This technique was directly applied in the case of the 486DX2 and the DSP, where the CPUs were part of actual evaluation boards. A variation of the technique was used for the '934, for which a suitable board was not available. The programs under consideration were simulated on a VERILOG model of the CPU to obtain test vectors. These were applied to the pins of a stand-alone CPU using an IC tester. The tester reported the average current drawn by the power pins of the CPU.

While the above technique is simple in nature, it provides a wealth of data that can be used to conduct a comprehensive power analysis of a given processor. It should also be stressed that the main idea behind instruction level power analysis is independent of the actual method used to measure average current. In addition, this analysis can be conducted even for un-fabricated CPUs. Instead of physical current measurements, current estimates can be obtained through simulations on low level design models of the CPU.

3 Instruction Level Power Modelling

The current measurement method described above can be used to assign power costs to individual instructions and inter-instruction effects. The costs are obtained through experiments which involve the creation of specific programs and measurement of the current drawn during their execution. These costs are the basic parameters that define the instruction level power model of a given processor. This approach has been empirically validated for all the subject microprocessors. The basic components of the power model are briefly discussed below.

Instruction Base Costs: The primary component of the power models is the set of *base costs* of instructions. The base cost of an instruction can be thought of as the cost associated with the basic processing needed to execute the instruction. It is obtained by executing a loop consisting of several instances of the given instruction. The product of the average current drawn during the execution of this loop and V_{CC} is the *base power cost* of the instruction. The base power cost multiplied by the number of non-overlapped cycles needed to execute the instruction is its *base energy cost.* For example, a register ADD instruction on the 486DX2 costs $314mA$ and takes 1 cycle leading to an energy cost of $213nJ$. A similar instruction on the '936 costs $199mA$ and 1 cycle, for an energy cost of $541nJ$.

The definition of base costs follows the convention that the base costs of instructions should not reflect the power contribution of effects like stalls and cache misses, which are modelled separately. The base cost of an instruction can vary with the value and address of its operands. While appropriate measurement experiments can give the exact cost for any given operand and address value, in practice it is seen that this variation in cost is only a secondary effect. The DSP, the smallest of the three processors, exhibited the maximum variation. But even this was less than 10% for most instructions.

Effect of Circuit State: Given an instruction sequence, using base costs of the individual instructions to model the cost of the sequence tends to underestimate the cost. This is because base costs do not account for the effect of the change in circuit state when two different instructions are executed consecutively. The concept of *circuit state overhead* for a pair of instructions is used to deal with this effect. This quantity is defined as the difference between the actual cost of an instruction pair and the average of the base costs of the individual instructions. Adding in the circuit state overhead for each pair of consecutive instructions leads to a much closer estimate for the cost of a sequence.

The above effect was observed for all the subject processors, but it varied in its impact. In the case of the 486DX2 and the '934, the circuit state overhead varied in a restricted range and had a limited impact. For the 486DX2 it varied in the range 0-30mA, while most programs varied in the range of 300-420mA. In the case of the '934, these ranges are 0-20mA and 250-400mA, respectively. However, in the case of the DSP, where the average current for most programs was in the range 20-60mA, the overhead for some pairs was significantly large - up to $25mA$.

The explanation for this difference may lie in the fact that in large complex processors like the 486DX2 and '934, a major part of the circuit activity is common to all instructions, e.g., the clocks, instruction prefetch, memory management, pipeline control, etc. While circuit state can result in significant variation within certain control and data path modules, its impact on the net power consumption of the processor is masked by the much larger common cost. However, in the DSP, which is a smaller, more basic CPU, the instruction control and data path constitute a larger fraction of silicon, and thus, the impact of circuit state is more visible.

Other Inter-Instruction Effects: The final component of the power model is the power cost of other inter-instruction effects that can occur in real programs, e.g., pipeline stalls and cache misses. Current measurement experiments are conducted to isolate the power costs of these effects. Multiplying the power cost of each kind of stall or cache miss by the number of cycles taken for each, gives the energy cost of these effects.

Overall Instruction Level Power Model: The parameters described above form the basis of estimating the energy cost of entire programs. The overall energy cost, E of a program is given by:

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

where for each instruction i, B_i is the base cost, and N_i is the number of times it will be executed, and for each pair of consecutive instructions (i, j), $O_{i,j}$ is the circuit state overhead, and $N_{i,j}$ is the number of times the pair will be executed. E_k is the energy contribution of the other inter instruction effects, k (stalls and cache misses), that would occur during the execution of the program. A software power/energy estimation framework based on the this idea is described in [5].

Given that the circuit state overhead varies in a limited range in the case of the 486DX2 and the '934, it suggests that a constant value could be used for all instruction pairs. This is a more efficient and yet fairly accurate way of modelling this effect. A value of $15mA$ and $18mA$ in the case of the 486DX2 and the '934, respectively has been found to be suitable. However, for the DSP, this effect has a greater variation and impact. A table of pairwise overhead values leads to a better estimate in this case [2].

4 Software Energy Optimization

The potential for energy reduction through modification of software has largely been ignored due to the lack of practical analysis techniques. The instruction level analysis technique described in the previous sections overcomes this deficiency. Application of this technique provides the fundamental information that can help guide higher-level design decisions like hardware-software partitioning and choice of algorithm. It also helps in the identification of sources of energy reduction that can then be exploited by software development tools like code generators and schedulers. Several ideas in this regard are described below.

Reducing Memory Accesses: An inspection of energy costs for the three processors reveals that instructions that involve memory accesses are much more expensive than instructions that involve just register accesses. For example, for the 486DX2, instructions that use register operands cost in the vicinity of $300mA$ per cycle. In contrast, memory reads and writes cost upwards of $400mA$ and $530mA$, respectively, even in the case of cache hits. For cache misses the cost is several times higher due to multiple cache-fill cycles, and the energy consumption in the external memory system.

Thus, large energy savings can be attained through a reduction in the number of memory accesses. This motivates the need for development of optimizations to achieve this reduction at all levels of the software design process. During compilation, the most effective way of reducing memory operands is through better utilization of registers. The potential of this idea has been demonstrated through some experiments in the case of the 486DX2 [4]. For example, for a certain *heapsort* program, an aggressive use of register allocation led to a 40% decrease in the CPU and memory energy cost. The results of our study show that the extensive research in register allocation also has an immediate application in compilation for low energy.

Energy Cost Driven Code Generation: The idea behind energy cost driven code generation is to select instructions based on their energy costs instead of the traditional metrics of code size or running time. An energy based code generator was created for the 486DX2 using this idea [4]. Interestingly, it was found that the energy and running time based code generators produced very similar code.

This observation provides evidence for a general trend that was observed for all the subject processors. This is that energy and running times of programs track each other closely. The difference in average current for sequences that perform the same function is usually not large enough to compensate for any difference in the number of cycles. This is an important observation since it can be considered as empirical justification for a powerful guideline for software energy reduction for today's processors - as a first step towards energy reduction, do what needs to be done to improve performance.

It bears mentioning that there may be certain application specific processors where this observation may not hold in general. Aggressive use of power management may also lead to situations where the fastest code may not always be the least energy code. In such cases, code generation based on energy costs will be useful in its own right, and the analysis technique described in the previous section can provide these energy costs.

Instruction Reordering for Low Power: Reordering of instructions in order to reduce the switching between consecutive instructions is a method for energy reduction that does not involve a corresponding reduction in the number of cycles. However, in the case of the 486DX2 and the '934, different reorderings of instruction sequences lead to very little change in the measured current. This is due to the limited variation in the circuit state overhead in these processors. For the DSP, however, this quantity is more significant and shows relatively greater variation. Thus, instruction reordering is more beneficial in this case. A scheduling algorithm that uses the measured overhead costs has been developed [2]. Energy reductions of up to 14% have been observed for some standard signal processing programs.

Processor Specific Optimizations: Instruction level power analysis of a given processor can lead to the identification of features specific to that processor that can then be exploited for energy efficient software. Some of the more noteworthy examples from the subject processors are briefly described below.

Instruction Packing and Dual Memory Loads: The DSP has a special architectural feature that allows certain pairs of instructions to be packed into a single instruction. The packed instruction executes in one cycle, as opposed to a total of two for the unpacked pair. Interestingly, even though the two cases represent the same functionality, the use of packing always leads to large energy reductions. Thus, instructions should be packed as much as possible, and it is worthwhile to perform instruction scheduling for a better utilization of packing. Doing so leads to large energy savings for real programs (e.g. 26% for IIR4 and 47% for LP_FIR60 [2]).

A related optimization exploits the dual data memory banks on the DSP. A special dual load instruction can transfer two operands, one from each memory, in one cycle. The same task can also be performed by two single load instructions over two cycles. However, the latter takes almost twice the energy. The large energy difference justifies the use of memory allocation techniques that can lead to better utilization of dual loads [1].

Swapping Multiplication Operands: The on-chip multiplier in the Fujitsu DSP was identified as a major source of energy consumption for signal processing applications. This multiplier is based on the Booth multiplication algorithm, which treats the two operands in very different ways. An appropriate swapping of the operands, in order to exploit this asymmetry, leads to up to 30% reduction in multiplication energy costs. This can translate into appreciable energy reduction for entire programs (e.g., 16% for LP_FIR60 [2]).

Software Controlled Power Management: The '934 provides a software mechanism for powering down unneeded parts of the CPU. The clock inputs to certain modules can be enabled or disabled through software. CPU energy reductions of up to 22% have been observed through the use of this mechanism. Providing software with the capability of controlling the power management of external devices can similarly lead to large energy reductions in the overall system.

5 Conclusions

The increasing role of software in today's systems demands that energy consumption be studied from the perspective of software. This paper describes a measurement based instruction level power analysis technique that makes it feasible to effectively analyze software power consumption. The ability to quantitatively analyze the power consumption of software makes it possible to deal with the overall system power consumption in an integrated way and can help in the development of effective power reduction techniques that are applicable for the entire system.

References

[1] T. C. Lee and V. Tiwari. A memory allocation technique for low-energy embedded DSP software. In *Proc. 1995 IEEE Symp. on Low Power Electronics*, San Jose, CA, Oct. 1995.

[2] T. C. Lee, V. Tiwari, S. Malik, and M. Fujita. Power analysis and low-power scheduling techniques for embedded DSP software. In *Proc. Intl. Symp. on System Synthesis*, Cannes, France, Sept. 1995.

[3] V. Tiwari and Mike T.-C. Lee. Power analysis of a 32-bit embedded microcontroller. In *Proc. Asia and South Pacific DAC*, Chiba, Japan, Aug. 1995.

[4] V. Tiwari, S. Malik, and A. Wolfe. Compilation techniques for low energy: An overview. In *Proc. 1994 IEEE Symp. on Low Power Electronics*, San Diego, CA, Oct. 1994.

[5] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, Dec. 1994.