Compiler Support for Reducing Leakage Energy Consumption*

W. Zhang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin and V. De*
Microsystems Design Lab, Penn State University and Intel Research Labs*

ABSTRACT

Current trends indicate that leakage energy consumption will be an important concern in upcoming process technologies. In this paper, we propose a compiler-based leakage energy optimization strategy. Our strategy is built upon a data-flow analysis that identifies basic blocks that do not use a given functional unit. Based on this information, the compiler then inserts activate/deactivate instructions in the code to set/reset a sleep signal which controls leakage current for functional units. Our experimental results show that the proposed compiler-based strategy is very effective in reducing leakage energy of functional units.

1. Introduction and Motivation

While dynamic energy is the dominant energy component in today's CMOS circuits, the trends show that leakage energy consumption will play a much larger role in upcoming circuit generations. This paper presents a compiler-based leakage energy optimization strategy. Our strategy first analyzes the control flow graph (CFG) representation of the program and, for each functional unit, determines the paths along which that functional unit is idle (unused). It then selects a suitable leakage control mechanism and inserts activate/deactivate instructions in the code to enable/disable the functional unit. Since reactivating a functional unit from the leakage control mode takes some extra execution cycles, we also consider a circuit pre-activation strategy which tries to bring the circuit to the normal operation mode before it is actually needed. In this work, two leakage control mechanisms are considered to exploit the idleness of functional units for reducing leakage energy. The first leakage control mechanism, called input vector control, exploits the state dependence of the leakage current and sets the inputs to values that have the minimum leakage current when the units are idle [3]. The second mechanism, called supply gating, eliminates the leakage energy consumption by cutting the power supply to the units [7].

The remainder of this paper is organized as follows. The details of our data-flow analysis to insert leakage control instructions in the code are explained in Section 2. The implementation and simulation environment are discussed in Section 3. Experimental results are given in Section 4. Section 5 shows the direction of our future work.

2. Data Flow Analysis to Reduce Leakage Energy

We formulate the idleness detection problem as a backward data-flow analysis problem and solve it using a worklist algorithm[6]. The formulation of our data-flow analysis can be described as the follows. Suppose that we have n functional units and that the functional unit usage is defined at the basic block level, instead of at the operation level. Obtaining the basic block level functional usage information is easy (in a VLIW architecture) as the scheduled code associates a functional unit with each operation in the basic block. Our compiler identifies each basic block using a number or id.

To build our data-flow equations, we use three different variables: $USE_{i,j}$, $OFF_{i,j}$, and $ON_{i,j}$. Informally, $USE_{i,j}$ tells us whether functional unit j is used by basic block i. $OFF_{i,j}$ and $ON_{i,j}$, on the other hand, are two sets and keep the numbers (ids) of basic blocks that will contain deactivate and activate instructions, respectively.

More formally:

$$USE_{i,j} = \begin{cases} 0, & \text{if no operation in basic block } i \text{ uses} \\ & \text{functional unit } j \\ 1, & \text{if at least one operation in basic block } i \text{ uses} \\ & \text{functional unit } j \end{cases} \quad (1)$$

Formal definitions for $OFF_{i,j}$ and $ON_{i,j}$ depend on the position of the basic block i in the CFG. If i is the last basic block (the terminal basic block), then we have:

$$OFF_{i,j} = \begin{cases} \{i\}, & \text{if } USE_{i,j} = 0 \\ \emptyset, & \text{if } USE_{i,j} = 1 \end{cases} \quad (2)$$

and

$$ON_{i,j} = \begin{cases} \{i\}, & \text{if } USE_{i,j} = 1 \\ \emptyset, & \text{if } USE_{i,j} = 0 \end{cases} \quad (3)$$

On the other hand, for the remaining blocks in the CFG, we have:

$$OFF_{i,j} = \begin{cases} \{i\}, & \text{if } USE_{i,j} = 0 \\ \cup_{k \in succ(i)} OFF_{k,j}, & \text{if } USE_{i,j} = 1 \end{cases} \quad (4)$$

and

$$ON_{i,j} = \begin{cases} \{i\}, & \text{if } USE_{i,j} = 1 \\ \cup_{k \in succ(i)} ON_{k,j}, & \text{if } USE_{i,j} = 0 \end{cases} \quad (5)$$

In these formulations, $\cup$ denotes set union and $\cup_{k \in succ(i)} OFF_{k,j}$ ($ON_{k,j}$) indicates the union of all $OFF_{k,j}$ ($ON_{k,j}$) sets, where $k \in succ(i)$.

Note that $OFF_{i,j}$ is a set of basic block number, in which we need to turn off functional unit j. Similarly, $ON_{i,j}$ is a set of basic block numbers, in which we need to turn the functional unit j on.

In the second step of our approach, the compiler inserts activate/ deactivate instructions in the code using the $OFF_{i,j}$ and $ON_{i,j}$ found in the first step.

3. Experimental Framework

We used Trimaran infrastructure [8] to implement our approach. To test the effectiveness of our energy-saving optimization, we used four benchmarks from MediaBench suite (`cordic`, `nbradar`, `rawcaudio`, and `rawdaudio` [5]) and two array-intensive applications (`tomcatv` and `vpenta`) from Spec benchmarks.

Our VLIW configuration has a total of nine functional units: 4 integer ALUs, 2 floating-point ALUs, 2 load/store units, and 1 branch unit. All results have been obtained using a register file of 32 entries and a perfect cache configuration (that is, all cache accesses are assumed to be hits). The energy numbers reported in this section are based on 0.1 micron technology, 1V supply voltage and 0.26V threshold voltage.

For each benchmark code, we experimented with six different versions as explained below. The versions differ from each other with respect to the leakage control mechanism used, whether or not profile information is utilized (in determining idle paths), and whether or not pre-activation is employed. What we mean by profile data here is the profile data indicating the duration of idleness.

- **Input Vector Control:** This version uses only input vector control.

*This work is supported in part by the NSF CAREER awards 0093082, 0093085, and NSF 0103583, and the MARCO GSRC grant 98-DT-660.

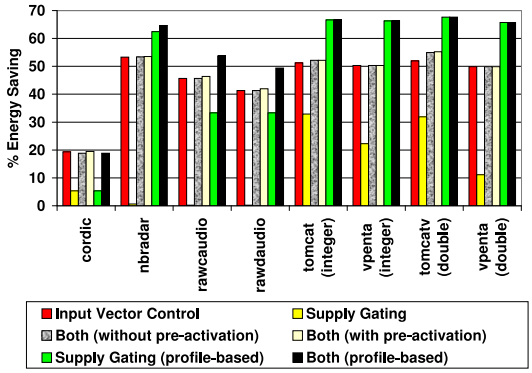

Figure 1: % energy improvements (with floating-point units).

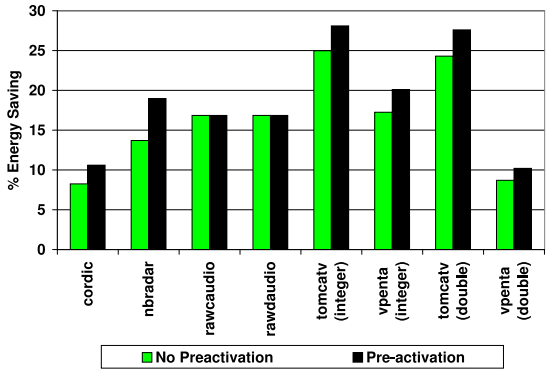

Figure 2: % energy savings with an reactivation latency of 45 cycles.

- **Supply Gating:** This version uses only power supply gating.
- **Both (without pre-activation):** This version employs both input vector control and supply gating. Depending on the duration of idleness, it chooses one of these mechanisms. It does not use pre-activation or profile data.
- **Both (with pre-activation):** This is similar to the previous version except that it employs pre-activation.
- **Supply Gating (profile-based):** This version uses only power supply gating. It uses profile data for obtaining duration of idleness. It does not use pre-activation.
- **Both (profile-based):** This version uses both input vector control and supply gating. To determine duration of idleness, it exploits profile data. It does not utilize pre-activation.

4. Results

Figure 1 gives the percentage leakage energy improvements for different optimized versions. It should be noted any increase in dynamic energy consumption and additional leakage consumed due to performance penalties resulting from our optimization strategies are also included in these results. We can make several observations from these results. First, the input vector control version performs very well. Specifically, it improves the functional unit leakage energy consumption by 45.4% on the average. The supply gating version, on the other hand, does not perform that well. In fact, in three applications (`nbradar`, `rawcaudio`, and `rawdaudio`), it hardly makes any difference in energy behavior. The main reason for this is the large re-activation time. When we combine these two leakage control mechanisms, we get an average improvement of 45.8%. This is only slightly better than the input vector control version. Again, the main reason for this is that in this combined version, input vector control dominates; that is, the compiler selects the input vector control in most cases due to the large re-activation time of the supply gating mechanism.

We also observe that including pre-activation did not bring too much benefit. This is again due to the fact that in most cases input vector control is used. Since the initiation and reactivation latencies of input vector control mechanism require a total of only 2 cycles, pre-activation can only avoid additional leakage energy consumed during the reactivation cycle. We also see that the profile-based supply gating generates much better results than the supply gating version. It generates an average energy reduction of 50.1%. This is because when we profile the code, we generally detect larger idle periods (e.g., we can detect the real execution time for loops instead of being conservative); as a result, the compiler uses power supply gating more aggressively.

To evaluate the impact of pre-activation on energy savings , we performed some experiments assuming per cycle energy savings similar to input vector control but with a hypothetical initiation (and re-activation) latency of 45 cycles (instead of 2 cycles). We see from the results given in Figure 2 that when the activation latency is larger, pre-activation starts to make a difference.

We also made experiments to measure the increase in execution time when different optimized versions are used. We observe that, except for `cordic`, the increase in execution cycles is bounded by 18%, and less than 10% in many cases.

5. Conclusions and Future Work

In this paper, we have proposed a compiler-based technique for optimizing leakage energy consumption in VLIW functional units. Our strategy is built upon a data-flow analysis that detects idle functional units along control-flow graph paths. After detecting idleness, the compiler takes into account the estimated basic block execution times and available leakage control strategies, and inserts (functional unit) activation/deactivation instructions in the code. Although the use of another runtime leakage control mechanism — body biasing [4] has not been considered in this work, it would be easy to investigate this technqiue using our strategy. This technique fits well in the phases abstracted for leakage control mechanisms except that it has no settling time penalty. Further, the leakage reduction will be around 50%-70% [4]. However, this technique will need a triple-well process for applying to both NMOS and PMOS. We are in the process of performing experiments with body biasing and of extending our data-flow analysis to an inter-procedural setting.

6. REFERENCES

[1] J. Casmira and D. Grunwald. Dynamic Instruction Scheduling Slack. In Proc. *2000 Kool Chips Workshop*, December 2000.

[2] D. Duarte, Y-T. Fai, N. Vijaykrishnan, and M. J. Irwin. Evaluating run-time techniques for leakage power reduction. In *Proc. ASP-DAC*, January, 2002, Banglore, India.

[3] J. P. Halter and F. Najm. A gate-level leakage power reduction method for ultra-low-power CMOS circuits. In Proc. *IEEE Custom Integrated Circuits Conference*, pp. 475–478, 1997.

[4] A. Keshavarzi et al. Effectiveness of Reverse Body Bias for Leakage Control in Scaled Dual Vt CMOS ICs In Proc. *ISLPED*, 2001.

[5] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communication systems. In Proc. *MICRO*, 1997.

[6] S. S. Muchnick. *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.

[7] S. Mutoh et. al. 1-V power supply high-speed digital circuit technology with multi-threshold-voltage CMOS. *IEEE Journal of Solid State Circuits*, vol. 30, no. 8, pp. 847–854, Aug. 1995.

[8] Trimaran homepage. http://www.trimaran.org.

[9] S.Rele, S.Pande, S.Onder, and R.Gupta, Optimization of Static Power Dissipation by Functional Units in Superscalar Processors, International Conference on Compiler Construction, LNCS 2304, Springer Verlag, pages 261-275, Grenoble, France, April 2002