Power-Aware Instruction Scheduling

Tzong-Yen Lin and Rong-Guey Chang *

Department of Computer Science
National Chung Cheng University
Chia-Yi, Taiwan

lty93@cs.ccu.edu.tw and rgchang@cs.ccu.edu.tw

Abstract. This paper presents an innovative DVS technique to reduce the energy dissipation. Our objective is to minimize the transitions between power modes by maximizing the idle periods of functional units with instruction scheduling. Our work first analyzes the control flow graph of the application, which contains many regions. Second, we collect the power information and build its power model for each region. Then two regions with the same functional units will be merged if no dependencies exist between them. The process is repeated until no further mergings can be performed. Next, the idle functional units will be turned off and each region will be assigned a power mode based on the power model. Finally, the application is rescheduled to merge the regions to reduce the transitions between power modes. The experimental results show that our work can save the energy by 26%.

1 Introduction

As most embedded systems are portable, how to reduce the energy dissipation when running applications to extend the lifetimes of batteries has become a crucial issue. The energy dissipation is closely related to voltage and clock frequency [11]. The architectures of modern processors provide several power modes to reduce energy dissipation by adjusting the voltage and the clock frequency at run time.

Previous work showed the switching overheads of power modes make a significant impact on energy dissipation [1,14]. In this paper, we reduce the energy dissipation with instruction scheduling to minimize the transitions between power modes by maximizing the idle periods of functional units. To exploit the maximum potential for instruction scheduling, for an application, we first use the SUIF compiler infrastructure to build its control flow graph (CFG) and data flow graph (DFG) [9,14]. Then these graphs are divided into many basic blocks. Each basic block is composed of several regions, each of which will contain at least one functional unit. The power model of the application is also built in this step by profiling the CFG. Second, to minimize the transitions between power modes, two regions that contain the same functional units will be merged if no dependencies exist between them by referring to DFG. Then for each region, the idle functional units in it will be turned off and it is

* This work was supported in part by National Science Council, Taiwan, under Grant NSC 94-2220-E-194-011-

assigned one power mode by referring to the power model. Afterward, instruction scheduling is applied again so that the regions with the same power modes will be merged if no dependencies exist in them. Finally, we assign the power modes to basic blocks to further optimize energy and performance. The experiments will show how much this potential can be exploited for our benchmarks.

The remainder of this paper is organized as follows. Section 2 presents our main idea with a motivating example. Section 3 describes our compilation flow, system architecture, and the algorithm in detail. The experimental results are shown in Section 4. Section 5 is the related work. Finally, section 6 concludes our work briefly.

2 Motivating Example

Figure 1 shows a motivating example to explain the basic idea of our work. Figure 1b is the optimized code segment for Figure 1a. In Figure 1a, There are four regions: two loops and two multiplication expressions. We assume that the adder is turned on, the multiplier is turned off, the default power mode is the normal mode, and two loops are assigned power down mode. To reduce energy dissipation, the functional units will be turned off. In Figure 1a, the adder will be turned on and the multiplier will be turned off in two loops, and the adder will be turned off and the multiplier will be turned on when doing multiplications. In Figure 1b, two loops and two expressions can be merged into two regions respectively since they contain the same functional unit and no dependences exist in them. Then the multiplier is turned off in the newly loop then the adder is turned off in the expressions. Finally, we assign the loop region power down mode and the expression region normal mode. In contrast with Figure 1a, Figure 1b can reduce more power consumption since it maximize the idle periods of functional units and save two transitions of power modes.

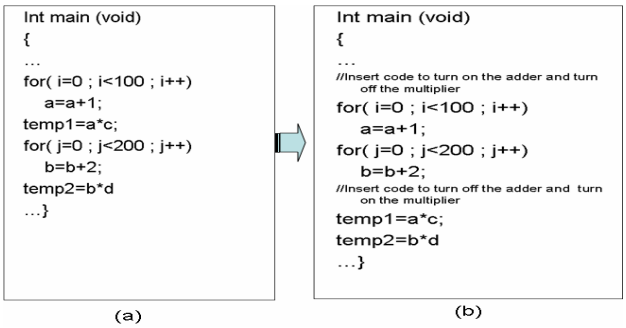


Fig. 1. Motivating example

3 Power-Aware Compilation

In this section, we first describe compilation flow and system architecture of our work and then present our algorithm.

3.1 Compilation Flow and System Architecture

Figure 2 shows our compilation flow and system architecture. Our approach is implemented on the basis of the SUIF2 compiler infrastructure and the Wattch simulator. SUIF2 helps us to build the control flow graph (CFG) and the data flow graph (DFG) and then we perform our low-power optimizations. First, we build the power model by profiling the graphs of the application. Meanwhile, the control flow graph will be divided into many regions according to the usage of functional units. Then each region will be assigned a power mode by referring to its power model. Next, the idle functional units in a region will be turned off. Then two regions will be merged to reduce the transitions of power modes with instruction scheduling, if the functional units in them have the same statuses, they have the same power modes, and no dependencies exist in them. The process will repeat until no further mergings can be performed. Finally, the resulted DVS'ed program is compiled and linked with run-time library to produce an executable code running on the Wattch simulator [3].

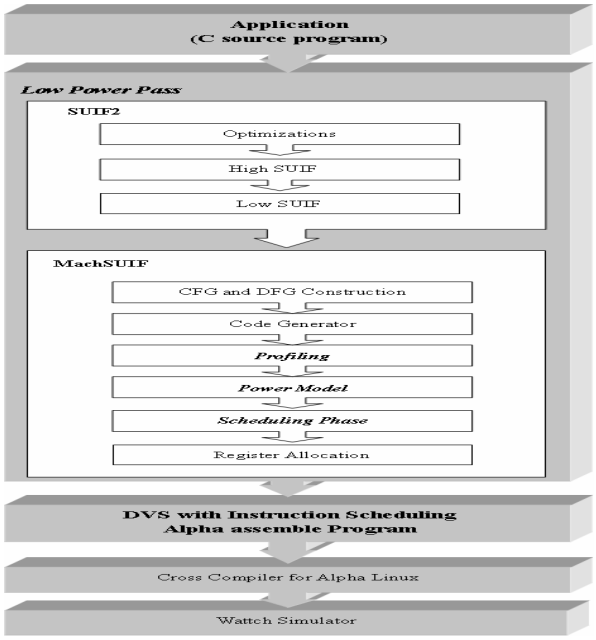


Fig. 2. Compilation flow and system architecture

3.2 Architectural Support

To turn on and off the functional units, we add new instructions into the Alpha instruction set in the Wattch simulator, which is listed below. In this paper, we only consider the adders and the multipliers for integer and floating point.

alu.on	switch ON one integer ALU functional unit
alu.off	switch OFF one integer ALU functional unit
mdu.on	switch ON one integer multiplier/divide functional unit

mdu.off	switch ON off integer multiplier/divide functional unit
alu.s.on	switch ON one float ALU functional unit
alu.s.off	switch OFF one float ALU functional unit
mdu.s.on	switch ON one float multiplier/divide functional unit
mdu.s.off	switch OFF one float multiplier/divide functional unit

In our algorithm, to perform the optimizations on an application, we define the following symbols that are used in this paper. For a basic block B and a region R in the basic block B,

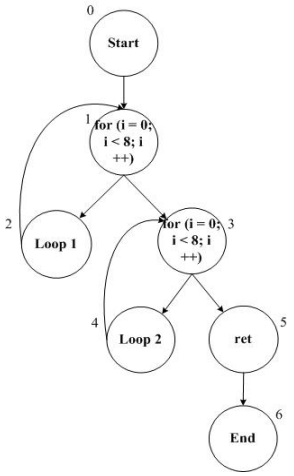
N(B) is the number of times when B is executed,
FU(B) is the set of functional unit used in B,
FU(R) is the set of functional unit used in R,
f_{mem} is the ratio of memory accesses to total instructions in B,
Tper(R) is the ratio of the execution time to that of B,
f (B) indicates the power mode of B, and
f (R) indicates the power mode of R.

After the program is partitioned into regions, two regions, say R_i and R_j, will be merged if the following conditions are satisfied. (1) $FU(R_i) = FU(R_j)$. It implies these two regions use the same functional units. (2) $D(R_i, R_j) = \varnothing$. It means that no dependencies exist in them. The merging process will be repeated until no regions satisfy the above two conditions. Here we first give the definitions of the following parameters used in this paper.

γ is the threshold that used to turn on or off a functional unit and
α and β are the thresholds that used to assign a region a power mode.

Next, for a region R, we decide to turn on or off the functional units in it if $T_{per}(R)$ is larger than the threshold γ, which is the default number of clock cycles. In our work, two power modes are implemented in the Wattch simulator by referring to the Crusoe TM5800 processor of Transmeta [15]. One is normal mode with voltage 1.5V and clock frequency 600MHz, and the other represented by f_{down} is the power down mode with voltage 0.3V and clock frequency 300MHz. Note that in this paper, the default power mode of a region is the normal mode. Moreover, we determine the assignment of power modes in the following two steps. In the first step, we first assign each region a power mode depending on the threshold α and then perform the merging repeatedly until no further mergings can be performed. Formally, the regions R_i and R_j satisfying $f(R_i) = f(R_j)$ and $D(R_i, R_j) = \varnothing$ will be merged across basic blocks. In the second step, the basic block will be assigned a power mode depending on the threshold β.

Now we use a code segment selected from 8x8 IDCT, which is used frequently in digital signal processing, as an example to demonstrate our idea. Figure 3 shows its control flow graph and power model. For simplicity, we use another code segment selected from 8x8 IDCT shown in Figure 4 to explain our algorithm. The original code segment shown in Figure 4a consists of five regions. Notice that regions R_1, R_3 and R_5 use adder only, and regions R_2 and R_4 use both adder and multiplier. Initially, we assume that the adder is turned on since some instructions such as load and store will use it to calculate the target address and the multiplier is turned off.



B_i	R_{ij}	$N(B_i)$	$T_{per}(R_{ij})$
B_0	N/A	1	N/A
B_1	R_0: Integer ALU	8	R_0 : 0.02
B_2	R_1: Integer ALU R_2: Integer ALU Integer multiplier/divide R_3: Adder R_4: Integer ALU Integer multiplier/divide R_5: Adder	8	R_1: 0.07 R_2: 0.07 R_3: 0.04 R_4: 0.18 R_5: 0.05
B_3	R_6: Integer ALU	8	R_6 :0.01
B_4	…	8	…
B_5	…	1	…
B_6	N/A	1	N/A

Fig. 3. Control flow graph and power model for a code segment selected from 8x8 IDCT

```
//R1
x8 = x7 + x1;
x1 -= x7;
x7 = x0 + x4;
x0 -= x4;
x4 = x1 + x5;
x1 -= x5;
x5 = x3 + x8;
x8 -= x3;
//R2
x3 = ir2c6 * (x2 + x6);
x6 = x3 + (- ir2c6 - ir2s6) * x6;
x2 = x3 + (- ir2c6 + ir2s6) * x2;
//R3
x3 = x7 + x2;
x7 -= x2;
x2 = x0 + x6;
x0 -= x6;
//R4
x6 = ic3 * (x4 + x5);
x5 = (x6 + (- ic3 - is3) * x5) >> 6;
x4 = (x6 + (- ic3 + is3) * x4) >> 6;
x6 = ic1 * (x1 + x8);
x1 = (x6 + (- ic1 - is1) * x1) >> 6;
x8 = (x6 + (- ic1 + is1) * x8) >> 6;
//R5
x7 += 512;
x2 += 512;
x0 += 512;
x3 += 512;
```

```
//R1
// turn off multiplier
x8 = x7 + x1;
x1 = x7;
x7 = x0 + x4;
x0 = x4;
x4 = x1 + x5;
x1 = x5;
x5 = x3 + x8;
x8 = x3;
//turn on multiplier
//R2
x3 = ir2c6 * (x2 + x6);
x6 = x3 + (- ir2c6 - ir2s6) * x6;
x2 = x3 + (- ir2c6 + ir2s6) * x2;
//R3-5
// turn off multiplier
x3 = x7 + x2;
x7 = x2;
x2 = x0 + x6;
x0 = x6;
x7 += 512;
x2 += 512;
x0 += 512;
x3 += 512;
//turn on multiplier
//R4
x6 = ic3 * (x4 + x5);
x5 = (x6 + (- ic3 - is3) * x5) >> 6;
x4 = (x6 + (- ic3 + is3) * x4) >> 6;
x6 = ic1 * (x1 + x8);
x1 = (x6 + (- ic1 - is1) * x1) >> 6;
x8 = (x6 + (- ic1 + is1) * x8) >> 6;
```

(a) **(b)**

Fig. 4. The sample and optimized code segment selected from 8x8 IDCT

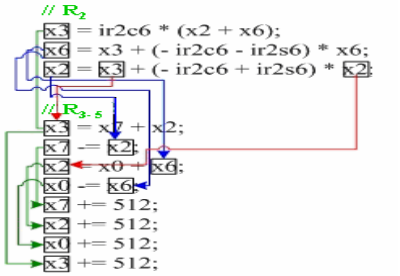


Fig. 5. Dependence graph between R_2 and $R_{3\text{-}5}$

According to the DFG of Figure 4a, we can merge R_3 and R_5 into a new region $R_{3\text{-}5}$ shown in Figure 4b after applying our approach to it. R1 cannot be merged with R_3 and R_5 due to the dependences, although they use the same functional unit. In addition, region R_2 and $R_{3\text{-}5}$ cannot be merged further into a region. Figure 5 shows the dependence graph between R_2 and $R_{3\text{-}5}$ and it explains why they cannot be merged. Thus, we can apply our algorithm to reschedule the code so that the regions with the same statuses can be merged. However, R_3 and R_5 can be merged into a new one, because multiplier can be shut down in them and no dependences exist between them. Figure 4b shows the optimized code segment.

4 Experimental Results

In this section, we present the experimental results of our work. Our work is implemented on the basis of the SUIF2 compiler infrastructure and the Wattch simulator. The applications are complied into the Alpha 21264 instructions by a cross compiler. The Alpha 21264 is one of the most advanced processors. It has four ALUs, aggressive branch predictor, two-level cache, TLB (translation look-aside buffer), and other modern designs [2]. The experiments are measured using SPEC2000, DSPstone, and Mediabench as benchmarks by setting three thresholds α = 0.3, β = 10, and γ = 3. With our experiences, these settings are better choices after applying our work to the above benchmarks. The impact of the settings on our work will be discussed further in the near future. In this section, we use “with scheduling” to represent all optimizations of our work and “without scheduling” to represent the optimizations without applying instruction scheduling.

4.1 Energy Evaluation

To demonstrate the effect of our work, we first show the power reductions after applying our approach to Dspstone, SPEC 2000, and Mediabench benchmarks, which are plotted with columns on the left axis of Figure 6 to Figure 8.

In Figure 6, our work can achieve an average energy reduction of 24% with scheduling and an average energy reduction of 19% without scheduling. For DSPstone, the effect of scheduling is small since the code sizes of applications in it are smaller. Basically, for a benchmark, our method will save more energy if its complexity is higher or its length is longer. For the matrix, the energy reduction caused by using scheduling can achieve up to 23% since it has a higher instruction level parallelism. It contains a 10 x 10 x 10 nested loop that calculates the multiplication of matrices. In this loop, several multiplications are calculated in different places. Our algorithm can work very well for "matrix" by grouping these multiplications together to maximize the idle period of the multiplier. So we can reduce the energy dissipation of a program. Figure 7 shows the experimental results after applying our approach with and without scheduling for the SPEC2000 benchmark. On average, the energy reduction can achieve up to 27%. The average energy reduction caused by the scheduling is around 11% that is larger than that of DSPstone. The reason is that the code sizes of applications in SPEC2000 are larger in

comparison with those of DSPstone and consequently we can exploit more instruction level parallelism to optimize. Figure 8 shows the experimental results after applying our approach with and without scheduling for the Mediabench benchmark. The average energy reductions with and without can achieve 25% and 18%. In Mediabench, applications like Jpeg, epic, and pgpwit have less dependences in them, thus the effects are better. In fact, with our experiences, our work can acquire better energy saving if the number of multipliers in the CPU is large.

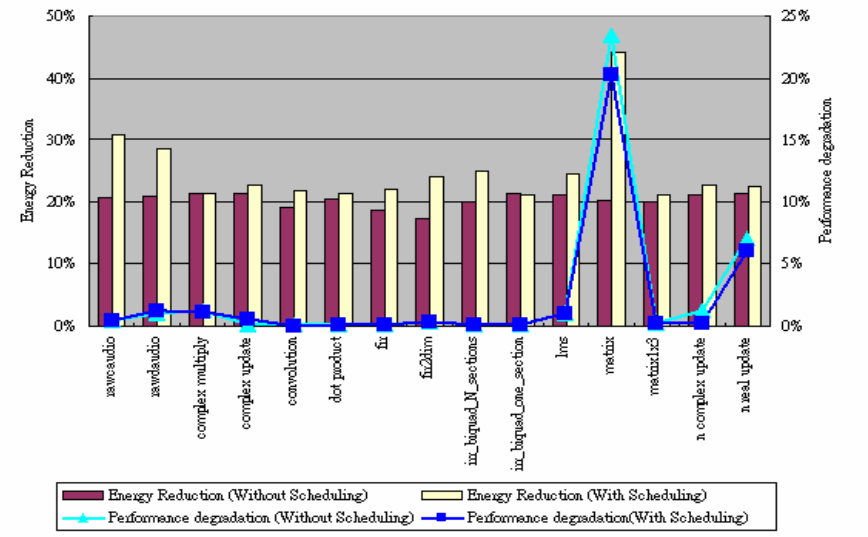


Fig. 6. Energy reduction and performance evaluation for DSPstone

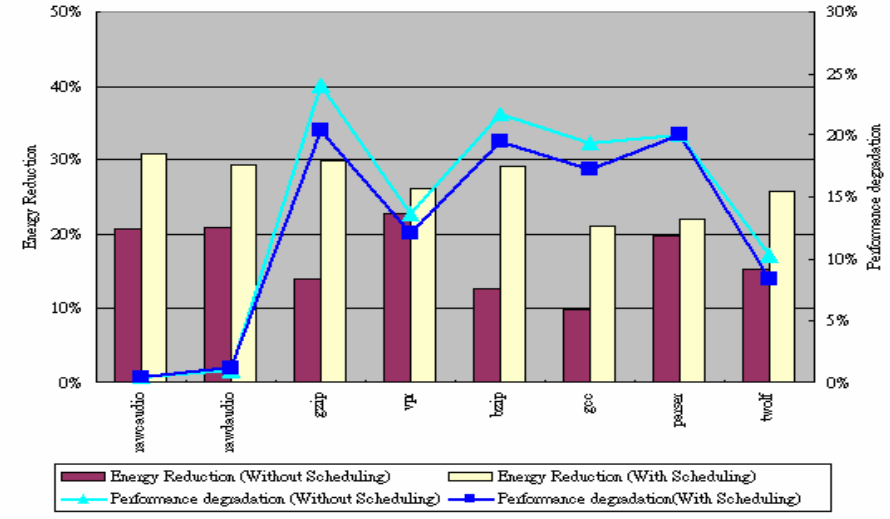


Fig. 7. Energy reduction and performance evaluation for SPEC2000

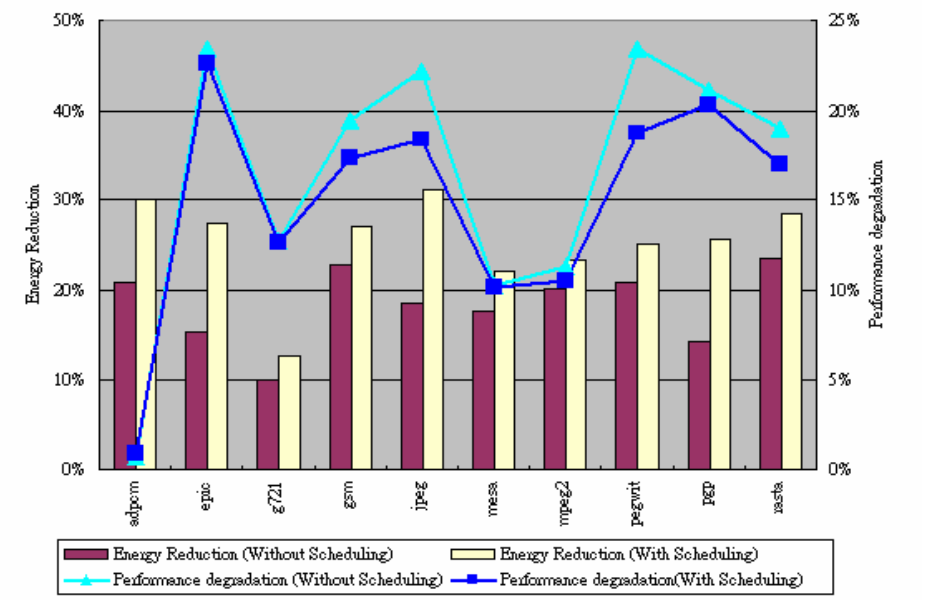


Fig. 8. Energy reduction and performance evaluation for Mediabench

4.2 Performance Evaluation

The performance impact on Dspstone, SPEC2000, and Mediabench with our approach is plotted with lines on the right axis of Figure 6 to Figure 8 It shows that our approach leads to performance degradation since our optimizations take some time to save energy. In DSPstone, the code sizes of most applications in DSPstoneit are smaller. Thus, the performance degradation caused by slowing down the CPU is smaller except matrix since its code size is large and it uses too many macros. By contrast, due to the larger code sizes of applications, the performance degradation of SPEC2000 is around 12.6% and 14.1% on average with and without scheduling. For Mediabench, the code sizes of most applications in it are larger, which results in worse performance. On average, the performance degradations are 14.9% and 16.4% with and without scheduling, respectively. The performance degradation of adpcm is better since its size is very small. In our experiments, no performance degradations of benchmarks will exceed 20% if "with scheduling" is applied. For the remaining cases, the performance degradations are under 10% except 'gzip'. In addition, with our knowledge, no previous DVS algorithms that use compilation techniques have shown the impact on the performance.

5 Related Work

Previous work reduced the energy dissipation by proposing various DVS techniques [1,4,6,7,8,10,12,13,14]. Some work [8,10] focused on scheduling the tasks using DVS to meet the real-time constraint to lower energy consumption. Shin et al. also aimed at the intra-task scheduling under the real-time constraint based on the power information generated by compiler [12]. Previous work addressed the DVS issue by slowing down the frequency with a low voltage in the regions containing many memory accesses [6,7]. In their work, the issue was modeled as the minimization problem with the performance and the transition constraints. Although they took the transition overheads into account, they did not reschedule the program to exploit the potential of reducing the transitions between power modes. Rele et al. devised a region-based approach to reduce the energy dissipation by turn off the idle functional units for superscalar processors [11]. Their work only showed the impacts on the utilization of functional units and performance after applying their work to programs, but it did not demonstrate the experimental results about power dissipation. By contrast, on the one hand our work extends the period of idle functional units and on the other hand we performs instruction scheduling on the programs to reduce the number of transitions between power modes. In comparison with previous work [5,6], they also divided an application into regions and adjust voltages and frequencies of power modes, but their approaches did not consider turning off the idle functional units and the transitions between power modes to save energy. You et al. presented three low-power optimizations with respect to a basic block [16]. One is to turn off the idle functional units in a basic block and the other two are to adjust the voltage of an execution path according to their two different power constraints. Their approach is partially similar to ours. But ours differs with theirs in the following two ways. (1) Our work targets at Alaph 21264 with four ALUs, while theirs was performed on a virtual architecture proposed by themselves. (2) Our work can maximize the idle periods of functional units with instruction scheduling and minimize the transitions between power modes, while theirs just turned off the idle functional unit without doing further optimizations.

6 Conclusions and Future Work

This paper presents an effective DVS approach at compiler time to reduce energy dissipation by attempting to minimize the transitions between power modes by maximizing the idle periods of functional units with instruction scheduling. To reduce energy dissipation when executing an application, we first implement new instructions to turn off the idle functional units and two power modes to adjust voltage and clock frequency of CPU. Then we apply instruction scheduling to maximize the idle periods of functional units and minimize transitions between power modes .Our work is performed with DSPstone, SPEC2000, and Mediabench benchmarks on the basis of the SUIF2 compiler infrastructure and the Wattch simulator. On average, the experimental results show that our work can save the energy by around 26% and lead to the performance degradation less than 18% for

most benchmarks. Our future research will focus on the settings of three thresholds to see how they influence the optimizations of our work.

References

1. N. AbouGhazaleh, D. Moss'e, B. Childers, and R. Melhem. Toward the placement of power management points in real time applications. In Proceedings of the Workshop on Compilers and Operating Systems for Low Power, September 2001
2. Alpha , Alpha 21264 Processor Technical Reference Manual, http://www.alpha.com.
3. D. Brooks , V. Tiwari , and M. Martonosi. Wattch: A Framework for Architectural Level Power Analysis and Optimizations. In International Symposium on Computer Architecture (ISCA) , Vanconver , British Columbia , 2000
4. T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In Proceedings of 2000 International Symposium on Low Power Electronics and Design, July 2000
5. C.H. Hsu and U. Kremer. Compiler-directed dynamic voltage scaling based on program regions. Technical Report DCS-TR-461, Department of Computer Science,Rutgers University, November 2001
6. C.H. Hsu and U. Kremer. Single region vs. multiple regions: A comparison of different compiler-directed dynamic voltage scheduling approaches. In Workshop on Power-Aware Computer Systems, 2002
7. C.H. Hsu and U. Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation, June 2003
8. C.M. Krishna and Y.-H. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. In Proceedings of the 6th Real Time Technology and Applications Symposium (RTAS'00), May 2000
9. MachSuif: A Framework built on top of SUIF for building back-ends http://www.eecs.harvard.edu/~hube
10. A. Manzak and C. Chakrabarti. Variable voltage task scheduling for minimizing energy or minimizing power. In Proceeding of the International Conference on Acoustics, Speech and Signal Processing, June 2000
11. K. Roy. Leakage Power Reduction in Low-Voltage CMOS Design. In IEEE International Conference on Circuits and Systems , Pages 167-173, 1998
12. Sannella, M. J. Constraint Satisfaction and Debugging for Interactive User Interfaces. Ph.D. Thesis, University of Washington, Seattle, WA, 1994
13. D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. IEEE Design and Test of Computers, 18(2), March/April 2001
14. SUIF. Stanford University Intermediate Format. http://suif.stanford.edu
15. Transmeta, Crusoe TM5800 Processor Technical Reference Manual http://transmeta.com/
16. Yi-Ping You, Chingren Lee, and Jenq Kuen Lee, Compilers for Leakage Power Reduction," accepted, ACM Transactions on Design Automation of Electronic Systems.