Design and Evaluation of Fine-Grained Power-Gating for Embedded Microprocessors

Masaaki Kondo[1], Hiroaki Kobyashi[2], Ryuichi Sakamoto[2], Motoki Wada[2], Jun Tsukamoto[2], Mitaro Namiki[2], Weihan Wang[3], Hideharu Amano[3], Kensaku Matsunaga[4], Masaru Kudo[4], Kimiyoshi Usami[4], Toshiya Komoda[1], and Hiroshi Nakamura[1]

[1]The University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
[2]Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei-shi, Tokyo 184-8588, Japan
[3]Keio University, 3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522, Japan
[4]Shibaura Institute of Technology, 3-7-5 Toyosu, Koto-ku, Tokyo 135-8548, Japan

***Abstract*—Power-performance efficiency is still remaining a primary concern for microprocessor designers. One of the sources of power inefficiency for recent LSI chips is increasing leakage power consumption. Power-gating is a well known technique to reduce leakage power consumption by switching off the power supply to idle logic blocks. Recently, fine-grained power-gating is emerged as a technique to minimize leakage current during the active processor cycles by switching on and off a logic blocks in much finer temporal/spatial granularity. Though fine-grained power-gating is useful, a comprehensive evaluation and analysis has not been conducted on a real LSI chips.**

In this paper, we evaluate fine-grained run-time power-gating for microprocessors' functional units using a real embedded microprocessor. We also introduce an architecture and compiler co-operative power-gating scheme which mitigates negative power reduction caused by the energy overhead associated with fine-grained power-gating. The experimental results with a fabricated core shows that a hardware-based scheme saves power consumption of functional units by 44% and hardware compiler co-operative scheme further improves power efficiency by 5.9% when core temperature is 25 ℃.

I. Introduction

With the end of Dennard scaling, the performance improvement of microprocessors comes at the cost of increasing power consumption. Power efficiency still remains a primary concern for microprocessor designers. One of the reasons for power inefficiency in recent LSIs is increasing leakage power. Despite many research efforts, it occupies a large portion of total power consumption, for example, 20% to 30% of total chip power in some processors fabricated with advanced semiconductor technologies [6], [7]. In recently attracted near-threshold voltage computing [2], in which the supply voltage approximately equals to the threshold voltage of transistors, the leakage power will be more dominant [10].

Power-gating (PG) is a well known technique to reduce leakage power by switching off the power supply to idle logic blocks. A number of manufactured LSI chips have already integrated the PG capability. Due to the lack of formulated design methodologies and energy overhead caused by switching on and off a logic block, PG is usually applied in coarse granularity. Most of today's microprocessors or system LSIs apply the core-level or the IP block-level PG with the temporal granularity as coarse as the task-level which is usually tens of microsecond to one millisecond [11], [15], [22]. Thus, PG is typically applied for standby period of the cores.

As leakage power becomes a major contributor to the total chip power, it is necessary to reduce the leakage current not only for a standby period of the core but also an application running period. Run-time power gating is recently emerged as a technique to minimize leakage current during the active processor cycles by switching on and off a logic block in much finer temporal/spatial granularity. Run-time PG is usually applied for caches [12] or microprocessor functional units [3], [8], [26], [23].

Though a number of research efforts have been conducted for effective PG so far, only few of them evaluate fine-grained PG on a fabricated LSI chip. In this paper, we evaluate fine-grained run-time PG (FRPG) in a fabricated microprocessor chip called Geyser-3 or FRPG core and analyze the effectiveness of the PG. Geyser-3 is upgraded and fully functional version of our prototype chips Geyser-1/2 [9], [27]. We also evaluate an architecture and compiler co-operative PG scheme which mitigates negative power reduction caused by the energy overhead associated with the PG. We implement FRPG in microprocessor functional units on a MIPS compatible embedded processor core with a single-issue in-order 5-stage pipeline.

The simple in-order pipeline is still widely utilized in many fields, specially in the embedded system area. Recent high-performance many-core processors such as GPUs also adopt a kind of in-order core as its base component due to its power efficiency. In such processors, functional units occupy a large portion of chip area. Therefore, microprocessors' functional units consume substantial leakage power. In this paper, we make the following contributions:

- We evaluate FRPG in a real LSI chip which implements an in-order single-issue embedded processor. The leakage power depends on some operational conditions, specially on chip temperature. We evaluate power-gating efficiency in several chip temperatures.
- We present a hardware and compiler co-operative PG management technique. The baseline hardware switches off the functional units eagerly, but the compiler restrain the hardware from switching off the units if the PG makes negative effect on power.
- We present the experimental results of the proposed hardware and compiler co-operative PG management technique in our processor chip. We show how the tight co-operation is effective for FRPG.

This work was partially supported by JSPS KAKENHI S Grant Number 25220002.

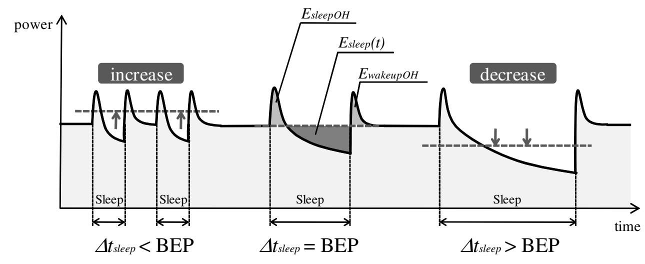
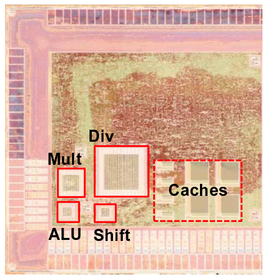


Fig. 1. Power profile with power-gating.



Process Technology	Fujitsu e-shuttle CMOS 65nm,12 metal layers
Chip Area [um]	ALU: 121.4 x 113.4 Shift: 116.4 x 114.8 Mult: 199.4 x 199.4 Div: 369.0 x 368.6 Total: 1610 x 1443
Vdd	1.2 [V]
L1 cache	L1-I: 8KB, 64B-line, 2way L1-D: 8KB, 64B-line, 2way
Synthesis	Sysnopsys Design Compiler
Layout	Synopsys ICC UPF

Fig. 2. Die photo of FRPG core and its specifications.

This paper is organized as follows. The next section describes the fundamentals of PG and related work. In Section 3, we describe an implemented PG processor core and a compiler support. The experimental environment and results are presented in Section 4. Finally, we conclude in Section 5.

II. Fine-Grained Power Gating

A. Basics of Power-Gating.

The MTCMOS or power-gating (PG) [18] is a well-known technique to reduce leakage current by switching off the power supply to circuit blocks with low threshold voltage (low-Vth) transistors. A series of sleep transistors with high threshold voltage (high-Vth) is inserted between a circuit block and the Vdd (or the ground). When the target circuit is in idle, leakage current can be suppressed by switching the sleep transistors off, putting the circuit block into the *sleep state*.

The mode transition between normal and sleep state incurs the delay and energy overhead due to sleep signal propagation, power-switch driving, and retrieval of electrical charge in the target circuit. Due to the energy overhead, switching off a circuit block does not necessarily lead to total power reduction. An example of power profile when several mode transitions occur is illustrated in Fig. 1. In this figure, $E_{sleepOH}$ and $E_{wakeupOH}$ represent energy overhead required to switch off and wake up the circuit block, respectively. $E_{sleep}(t)$ indicates the amount of energy reduced by PG after the circuit block is switched off, which of course depends on the duration of a sleep period of t. There exists a certain time period where the amount of reduced leakage energy is equal to the energy overhead as follows.

$$E_{sleep}(t') = E_{sleepOH} + E_{wakeupOH} \tag{1}$$

This time duration of t' is called *Break-Even Point* or *BEP*. If a sleep period is longer than BEP, we will obtain power saving. On the other hand, if a sleep period ends before BEP, we will encounter a power loss by PG. Therefore, several researchers focus on how to avoid this negative power saving [26], [16], [17].

B. Related Work

Due to exponential increase in leakage power consumption as process technology advances, there have been a lot of research efforts to reduce leakage current. Because a number of transistors are devoted to caches in today's microprocessors and leakage power is likely to increase as the number of transistors increases, there have been many proposals to reduce cache leakage power [12], [4]. Besides them, researchers have also studied to save leakage power for logic blocks in an architecture level [3], [8], [26], [23]. Saving leakage power for logic blocks used in a processor core still has a large impact on the total power consumption because transistors for logic blocks are usually more leaky than those used in caches [1].

In [3], analytical models to control the sleep mode for domino logic circuits are proposed. The potential of leakage-energy saving by simple PG control strategies is evaluated in [8]. They have investigated a simple time-based technique which switches off a functional unit after observing a pre-determined number of idle cycles. In [26], a more sophisticated sleep mode control technique is proposed.

Many compiler techniques for functional units' leakage power reduction are proposed so far. Most of them employ static code analysis or dynamic profiling to identify the idle period of a functional unit. In [20], long idle periods for each execution unit are detected and the mode control is directed by the compiler. A compiler-based leakage control technique for VLIW architectures are explained in [13]. You et al. proposed the compiler analysis framework for estimating the component activities and sleep instruction scheduling policies [25]. An instruction scheduling algorithm that assists the hardware based scheme in the context of VLIW and clustered VLIW is proposed in [19]. Roy et al. tried to predict the idle time of functional units in embedded microprocessors by analyzing loop structures of embedded applications [21]. Talli et al. used dynamic profiling information to identify the functional unit idle periods [24]. Unlike many compiler based techniques, our compiler support does not insert any additional instruction into a binary code. We try to mitigate negative power saving by the compiler-based PG control cooperated with the hardware-based PG strategy.

So far, fine-grained PG is mainly evaluated with a simulation. Though a prototype chip with fine-grained PG was partially evaluated [9], [27], a limited number of experiments are presented and compiler-based PG control mechanism is not evaluated. We evaluate fine-grained PG and the cooperative control scheme in a real LSI chip and confirmed the scheme is very useful. This emphases the strength of this paper.

III. Implementation of Fine-Grained Power-Gating

A. Fine-Grained Power-Gating Core

As a platform for a feasibility and proof-of-concept study, we have implemented fine-grained PG in microprocessor functional units (FUs). The architecture of the microprocessor core

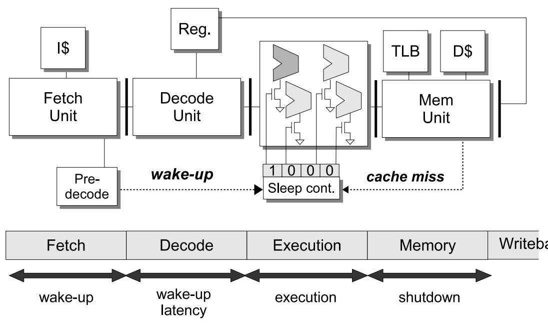


Fig. 3. Power gating architecture.

is MIPS R3000, a 32 bit RISC processor widely used in embedded systems. Fig. 2 shows a die photo of our FRPG core and its specifications [1]. The core has a traditional 5-stage pipeline, 8KB instruction and data caches, and a 64-entry TLB.

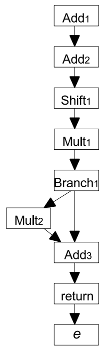
1) Basic Power-Gating Functionality: The FRPG core has four independent PG domains, ALU, shifter, multiplier, and divider. Switching on and off these FUs is individually controlled by a dedicated sleep controller. Because the activity of the FUs changes cycle by cycle and their average usage ratio is not very high, they are good target for fine-grained PG. We implement a hardware-based instruction-by-instruction PG methodology in the FRPG core.

The core puts each FU into sleep state automatically after the operation on the corresponding FU completes. For waking up, the sleep controller detects an FU to be used every cycle by checking a fetched instruction. It is easy to detect an FU used for each instruction by decoding the *opcode* and *funct* fields. That process is usually done in the decode stage of the pipeline located one stage before the execution stage. If wake-up decision is made in the decode stage, the operation must be delayed since it takes a certain time to wake up an FU. To hide this latency, we use pre-decoding technique. When an instruction is fetched in the fetch stage, the sleep controller immediately checks it and detects an FU to be used as shown in Fig. 3. This allows us one cycle time margin for the wake-up. Note that for a cache miss, the core is stalled and all the FUs are in idle for a long period of time. Therefore, all the FUs are forced to be switched off whenever a cache miss happens.

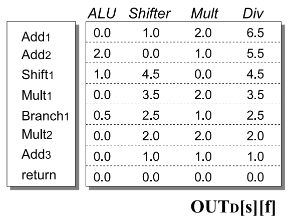
2) Software Control Interface: Because of the energy overhead, the instruction-by-instruction PG does not always bring power reduction. It is preferable that the core switches off an FU only when the following idle period is longer than BEP of the FU. To appropriately control the PG functionality, the FRPG core has software interface: a mode control register and a PG-cancel flag associated with assembly instructions.

The role of the mode control register is simple, enabling or disabling PG functionality for each FU. The register can be controlled by an OS. The PG-cancel flag is used to cancel instruction-by-instruction PG by assembly instructions. It is encoded into opcode field of arithmetic and logic instructions. We assign an unused bit of opcode in these instructions for the flag. Once an operation is performed in the execution stage, the corresponding FU is kept powered-on or switched

[1]The right part of the die implements test circuits for another purpose which does not affect any of the discussion and evaluation presented in this paper.



(a) Example CFG



	ALU	Shifter	Mult	Div
Add1	0.0	1.0	2.0	6.5
Add2	2.0	0.0	1.0	5.5
Shift1	1.0	4.5	0.0	4.5
Mult1	0.0	3.5	2.0	3.5
Branch1	0.5	2.5	1.0	2.5
Mult2	0.0	2.0	2.0	2.0
Add3	0.0	1.0	1.0	1.0
return	0.0	0.0	0.0	0.0

OUTD[s][f]

(b) Predicted idle cycle

Fig. 4. Example of a CFG.

off according to the flag. The compiler sets the flag based on predicted idle time of each functional unit. The algorithm of idle cycle prediction is described in the next section.

B. Compiler Based Power-Gating Control

The role of our compiler is to predict the idle period of each FU statically and to set the appropriate PG-cancel flag for each assembly instruction based on the predicted idle period so that PG decision is made only when the idle period of each FU is longer than the BEP. To estimate the idle period, we analyze a control-flow graph (CFG) constructed from the assembly code. Fig. 4 shows an example CFG for a function. Each node except e represents an assembly instruction, and the node e represents the exit point of the function which does not actually exist in the assembly code.

The expected idle period of each FU is estimated based on data flow analysis in order to deal with branches which make the control-flow undeterministic. We define the following real number variables which express the expected usage interval of FUs for each node and for every functional unit.

$$IN_D[s][f], IN_P[s][f], OUT_D[s][f], OUT_P[s][f]. \qquad (2)$$

Here, $OUT_D[s][f]$ indicates the expected number of cycles between node s and the next instruction which uses FU of f. $IN_D[s][f]$ represents a value that has the same meaning with $OUT_D[s][f]$ but for the instruction right before node s. $OUT_P[s][f]$ expresses the probability of reaching to the exit point of the function at node s without executing any instruction which uses FU of f. $IN_P[s][f]$ is the same as $OUT_P[s][f]$ but defined for right before reaching to node s.

We also define constant values of $T_D[s][f]$ and $T_P[s][f]$ for each node. They are determined based on whether the node uses FU f or not.

$$T_D[s][f] = \begin{cases} 0 & (\text{if } s \text{ uses unit } f) \\ c & (\text{otherwise}) \end{cases} \qquad (3)$$

$$T_P[s][f] = \begin{cases} 0 & (\text{if } s \text{ uses unit } f\ ) \\ 1 & (\text{otherwise}) \end{cases} \qquad (4)$$

$T_D[s][f]$ represents the expected increase in idle time of FU f when the core is executing the instruction at node s. The value is 0 if FU f is used by the instruction since idle time does not increase in this node. Otherwise, the value is the

$$\boxed{\begin{array}{l} IN_D^{(0)}[s][f] = T_P[s][f] * T_D[s][f] \\ IN_P^{(0)}[s][f] = T_P[s][f] \\ OUT_D^{(0)}[s][f] = 0,\ OUT_P^{(0)}[s][f] = 0 \\ IN_D^{(b)}[s_{exit}][f] = 0,\ IN_P^{(b)}[s_{exit}][f] = 1 \text{ (for exit node)} \end{array}}$$

Fig. 5. Initial values for the algorithm.

pipeline occupation time for the instruction (denoted as "c" in Equation (3)). If node s is a load or store instruction which may cause a cache miss, "c" could be average memory access time. A profiling technique can be used to estimate the average memory access time by obtaining the cache miss probability of the load/store instructions. $T_P[s][f]$ means the probability that the instruction uses FU f.

With those variables and constants, we make use of the following data-flow equations which provide delivery of the variables between adjacent nodes.

$$\begin{array}{l} IN_D[s][f] = T_P[s][f] * OUT_D[s][f] + T_D[s][f] \\ IN_P[s][f] = T_P[s][f] * OUT_P[s][f] \end{array} \tag{5}$$

$$OUT_D[s][f] = \begin{cases} q[s]*IN_D[s_{next1}][f]+(1-q[s])*IN_D[s_{next2}][f] \\ \quad (\text{if } s \text{ is branch.}) \\ IN_D[s_{next}][f] \\ \quad (\text{otherwise}) \end{cases} \tag{6}$$

$$OUT_P[s] = \begin{cases} q[s]*IN_P[s_{next1}][f]+(1-q[s])*IN_P[s_{next2}][f] \\ (\text{if } s \text{ is branch.}) \\ IN_P[s_{next}][f] \\ (\text{otherwise}) \end{cases} \tag{7}$$

Here, s_{next} indicates the node next to node s for the case that s is not a branch instruction. If node s is a branch instruction, there are two following nodes which are denoted as s_{next1} and s_{next2} in Equation (6) and (7). Moreover, $q[s]$ is the probability that the branch jumps to the node s_{next1}. This branch probability can be obtained by several ways such as static code analysis or profiling. If these techniques are unavailable, the probability of 0.5 will be used.

We calculate data-flow equations (5)–(7) iteratively so that the information of the expected idle time (i.e. $OUT_D[s][f]$) is propagated backward along the CFG. The initial values for the iterative calculation for each node s and the boundary values for the exit node are given in Fig. 5. Note that the above analysis is performed within a function. To predict the idle time more accurately, an interprocedural analysis is required. The detail about the interprocedural analysis is found in some litterateurs such as [14].

While the target FRPG core in this paper is a single-issue processor, this compiler analysis is extensible to superscalar processors. For superscalar processors, we need to merge multiple nodes which are likely to be executed simultaneously. The maximum values of $T_D[s][f]$ and $T_P[s][f]$ among merged nodes are selected as their new values.

1) Generation of Binary Code with PG-cancel Flag: Once the expected idle period of each FU for all the nodes is estimated, we next generate a binary code with PG-cancel flags. The compiler just sets PG-cancel flag for each instruction if the predicted idle period of that instruction is shorter than the given BEP. Note that if we do not set the PG-cancel flag for all the instructions, the binary code becomes the same as that of original one. Since the wake-up from the sleep state is automatically performed by the hardware, the FRPG core is binary-compatible to an ordinary MIPS R3000 processor.

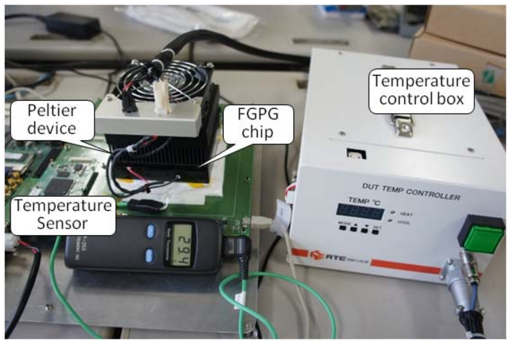


Fig. 6. Evaluation environment.

Since a binary code is statically generated for a pre-determined target BEP, the compiler scheme is useful if the actual BEP is known in advance. However, BEP can vary according to core temperature and process variation. Therefore, multiple object codes for various BEPs are first generated by the compiler. Based on operating environment, one of them are selected dynamically by a loader which loads a binary code into main memory at runtime. Even if BEP varies within a task execution due to temperature fluctuation, the code can be dynamically altered by the help of the OS so that the core executes appropriate code corresponding to the current BEP. This dynamic code management is left for our future work.

IV. Evaluation

A. Experimental Environment

Fig. 6 is a photograph of our experimental environment. The FRPG chip is mounted on a daughter card which is stacked on a main system board. A commercially available FPGA board with several I/O devices including a DIMM is connected to the system board. To evaluate FRPG chip with various core temperatures, a temperature control system with a Peltier device is attached on top of the chip. A temperature sensor is put between the chip and the Peltier device to monitor the actual chip temperature.

We equip the system board with current sensors. There are two independent power supply domains, one is for four FUs and the other is for the entire chip except these FUs. The sensors measure the current every 33ms. The program on FRPG core can read the information from the sensors through device driver on the Linux kernel. In this section, we evaluate only the power consumption of the FU domain.

We evaluate several programs from MiBench[5]. The gcc compiler is used to generate binary codes of applications. Since the FRPG core does not have floating-point functional units, we use the soft-float emulation provided by gcc.

Though the FRPG core itself can operate at the clock frequency up to 200MHz with instruction-by-instruction PG, we use 40MHz core clock frequency in this evaluation because the I/O bus connecting to the FPGA board cannot be operated with higher clock speed.

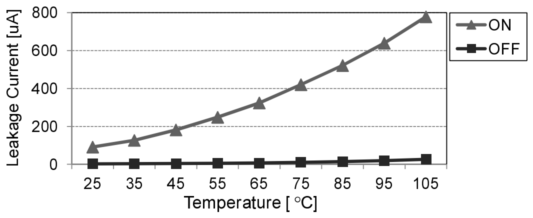


Fig. 7. Leakage reduction with power-gating.

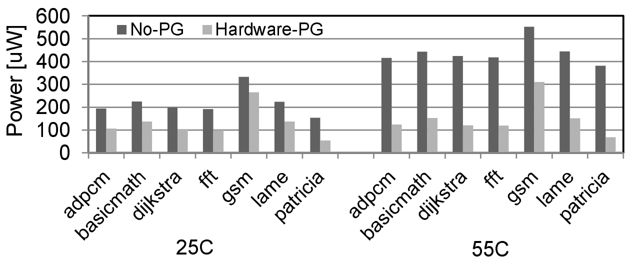


Fig. 8. Power reduction of hardware based power-gating.

TABLE I. TARGET BEP FOR COMPILER CODE GENERATION.

	ALU	Shift	Mult	Div
BEPset(25C)	56	47	28	11
BEPset(55C)	21	21	11	4

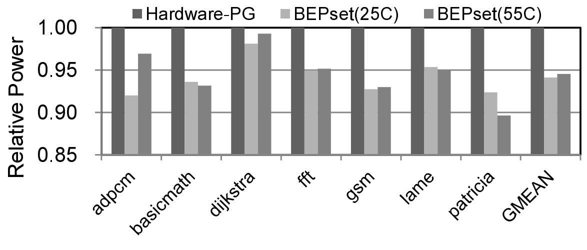
B. Result

1) Basic Power-Gating Performance of the Platform: We briefly show basic PG performance of the FRPG core. Fig. 7 shows leakage current for various chip temperatures. We compare the leakage current for two cases, all the functional units are powered-on (ON) and powered-off (OFF). In order to exclude the dynamic power consumption, the clock signal is stopped in this experiment. As seen from the figure, the FRPG core successfully reduces the leakage current by PG, for example, by 97.3% at 85 ℃.

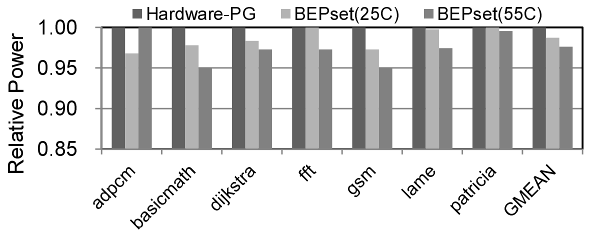
2) Hardware-Based Power-Gating: Fig. 8 presents power consumption of the FUs for two cases of chip temperatures, 25 ℃ and 55 ℃. In the figure, "No-PG" indicates the case where all the FUs are always switched on (PG is disabled) and "Hardware-PG" indicates the simple hardware based scheme.

As seen from the figure, Hardware-PG greatly reduces power consumption compared with no-PG. On average among all the programs, 44% and 67% of power reduction is achieved for the core temperature of 25 ℃ and 55 ℃, respectively. For higher chip temperatures, FRPG core saves more power consumption because the leakage power increases in higher chip temperatures. Even in the case of 25 ℃, the effect of power saving is significant. Since all the FUs are not utilized at the same time and the Mult and the Div units are idle in most of execution time, leakage power consumption is dominant in microprocessors' functional units. This indicates that the fine-grained runtime PG with instruction-by-instruction sleep transistor control is very useful.

3) Compiler-Supported Power-Gating: Next we discuss how the compiler-based PG control affects the overall power consumption. We used target BEP values shown in Table I for compiler code generation. For example, "BEPset(25C)" means a case where the core executes the code generated by our compiler targeted for BEP values for 25 ℃.



(a) Core temperature of 25 ℃.



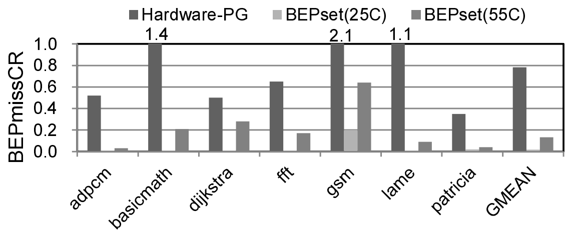
(b) Core temperature of 55 ℃.

Fig. 9. Additional power saving by compiler support.

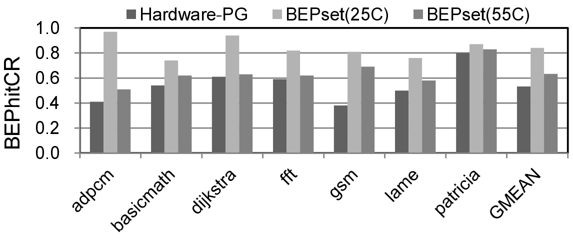
Fig. 9 (a) and (b) show relative power consumption of the FUs for the compiler-based scheme normalized to the hardware-based PG for chip temperature of 25 ℃ and 55 ℃, respectively. As seen from the figures, the compiler-based scheme reduces power consumption compared with simple hardware-based PG in almost all the cases. It is observed that the compiler-based scheme reduces more power for lower chip temperature. For chip temperature of 25 ℃ (Fig. 9 (a)), the compiler support with BEPset(25C) and BEPset(55C) reduces power consumption compared with hardware-based by 5.9% and 5.4%, respectively. When the chip temperature is higher (Fig. 9 (b)), the effect of the compiler support is not very significant. This is because BEP is relatively large in lower temperatures, switching off the power-supply whenever each FU is in idle incurs much dynamic energy overhead. These results indicate that the compiler-based scheme is very useful when the chip temperature is low which is usually the case for embedded systems.

On the whole, when the presuming BEP set is close to the actual BEP of the core, we obtain better power saving in most of the cases. This means using a correct BEP set in compilation time is very important. In some programs, the best power saving is achieved with a BEP set being different from the assumed core temperature. For example, for `basicmath`, `lame`, and `patricia` in Fig. 9 (a), BEPset(55C) whose target BEP values are assumed for the chip temperature of 55 ℃ achieves better results even though the actual chip temperature is different. This is because, for those programs, the compiler cannot predict the idle period correctly which leads to wrong PG decisions. The idle period differs from the predicted one if branch direction is inclined and cache miss happens frequently. If we use some profiling technique to know branch taken/not-taken rate and cache miss rate for each branch or load/store instruction, the estimated idle periods will become more accurate.

4) Break-Even Point Miss and Hit Cycle Rate: The efficiency of fine-grained PG is strongly affected by appro-



(a) Break-even point miss cycle rate.



(b) Break-even point hit cycle rate.

Fig. 10. Break-even point miss/hit cycle rate for 25 ℃.

priateness of the PG decisions. We discuss *BEP miss cycle rate (BEPmissCR)* and *BEP hit cycle rate (BEPhitCR)* whose definitions are as follows.

$$BEPmissCR = \sum_{i=0}^{BEP} (BEP - i)NS_i/TotalSleepCycle \quad (8)$$

$$BEPhitCR = \sum_{i=BEP+1}^{\infty} (i - BEP)NS_i/TotalSleepCycle \quad (9)$$

Here, NS_i is the number of occurrences of i-cycle sleep and $TotalSleepCycle$ is the total sleep cycles. These numbers were measured by hardware counters equipped in the evaluation environment. The smaller the BEPmissCR, the better the PG efficiency. Also the higher the BEPhitCR, the better the PG efficiency.

Fig. 10 presents BEPmissCR and BEPhitCR of the Shift unit for the chip temperature of 25 ℃. As shown in Fig. 10 (a), the hardware based scheme incurs relatively large BEPmissCR which corresponds to large energy overhead. The compiler scheme greatly reduces BEPmissCR when using an appropriate BEP set. BEPhitCR is also important for evaluating efficiency of FRPG. Using the compiler support with an appropriate BEP set, BEPhitCR increases in all the programs in the Shift unit as presented in Fig. 10 (b). These results indicate our compiler scheme is effective for PG mode control.

V. Conclusion

In this paper we evaluated fine-grained power-gating (PG) in a real embedded microprocessor. We introduced an architecture and compiler cooperative PG scheme which mitigates negative power reduction caused by the energy overhead of PG. The experimental results shows the simple hardware-based scheme is very effective for leakage reduction. We can save 44% to 67% of power consumption by hardware-based fine-grained PG. When we cooperatively control PG with the compiler, the power efficiency improves by 5.9%. These results indicate tight cooperation of hardware and compiler for PG management is very promising for better power reduction

Our future work is to improve compiler's idle period estimation accuracy by profiling. The evaluation with wide verity of programs is also needed to analyze PG efficiency. Evaluating the dynamic code management mechanism varying the chip temperature is left for our future work.

References

[1] J. Butts and G. Sohi, "A static power model for architects," in *Proc. 33rd MICRO*, pp. 191–201, 2000.

[2] R. Dreslinski *et al.*, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[3] S. Dropsho *et al.*, "Managing static leakage energy in microprocessor functional units," in *Proc. 35th MICRO*, pp. 321–332, 2002.

[4] K. Flautner *et al.*, "Drowsy caches: Simple techniques for reducing leakage power," in *Proc. 29th ISCA*, pp. 148–157, 2002.

[5] M. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. 2001 International Workshop on Workload Characterization*, pp. 3–14, Dec. 2001.

[6] J. Hart *et al.*, "3.6ghz 16-core SPARC SoC processor in 28nm." in *Proc. 2013 ISSCC (in presentation slide)*, pp. 48–49, 2013.

[7] W. Hu *et al.*, "Godson-3b1500: A 32nm 1.35ghz 40w 172.8gflops 8-core processor." in *Proc. 2013 ISSCC*, pp. 54–55, 2013.

[8] Z. Hu *et al.*, "Microarchitectural techniques for power gating of execution units." in *Proc. the 2004 ISLPED*, pp. 32–37, 2004.

[9] D. Ikebuchi *et al.*, "Geyser-1: A MIPS R3000 cpu core with fine grain runtime power gating," in *Proc. the 2009 ASSCC*, pp. 281–284, 2009.

[10] S. Jain *et al.*, "A 280mv-to-1.2v wide-operating-range IA-32 processor in 32nm CMOS," in *Proc. 2012 ISSCC*, pp. 66–68, 2012.

[11] Y. Kanno *et al.*, "Hierarchical power distribution with 20 power domains in 90-nm low-power multi-cpu processor," in *Proc. 2006 ISSCC*, pp. 540–541, 2006.

[12] S. Kaxiras, H. Zhigang, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. 28th ISCA*, pp. 240–251, 2001.

[13] H. Kim *et al.*, "Adapting instruction level parallelism for optimizing leakage in VLIW architectures," in *Proc. the 2003 LCTES*, pp. 275–283, 2003.

[14] T. Komoda *et al.*, "Compiler directed fine grain power gating for leakage power reduction in microprocessor functional units." in *Proc. 7th ODES*, 2009.

[15] J. Koppanalil *et al.*, "A 1.6 ghz dual-core ARM Cortex A9 implementation on a low power high-k metal gate 32nm process," in *Proc. 2011 VLSI-DAT*, pp. 1–4, 2011.

[16] A. Lungu *et al.*, "Dynamic power gating with quality guarantees," in *Proc. the 14th ISLPED*, pp. 377–382, 2009.

[17] N. Madan *et al.*, "A case for guarded power gating for multi-core processors," in *Proc. the 17th HPCA*, pp. 291–300, 2011.

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

[19] R. Nagpal and Y. Srikant, "Compiler-assisted leakage energy optimization for clustered VLIW architectures," in *Proc. the 6th EFSOFT*, pp. 233–241, 2006.

[20] S. Rele *et al.*, "Optimizing static power dissipation by functional units in superscalar processors." in *Proc. the 11th CC*, pp. 261–275, 2002.

[21] S. Roy, S. Katkoori, and N. Ranganathan, "A compiler based leakage reduction technique by power-gating functional units in embedded microprocessors," in *Proc. the 20th VLSID*, pp. 215–220, 2007.

[22] S. Sawant *et al.*, "A 32nm Westmere-EX Xeon enterprise processor." in *Proc. 2011 ISSCC*, pp. 74–75, 2011.

[23] N. Seki *et al.*, "A fine grain dynamic sleep control scheme in MIPS R3000," in *Proc. 26th ICCD*, pp. 612–617, 2008.

[24] S. Talli, R. Srinivasan, and J. Cook, "Compiler-directed functional unit shutdown for microarchitecture power optimization," in *Proc. the 26th IPCCC*, pp. 372–379, 2007.

[25] Y.-P. You, C. Lee, and J. Lee, "Compilers for leakage power reduction," *ACM Tr. on Design Automation and Electronic Systems*, vol. 11, no. 1, pp. 147–164, 2006.

[26] A. Youssef, M. Anis, and M. I. Elmasry, "Dynamic standby prediction for leakage tolerant microprocessor functional units." in *Proc. 39th MICRO*, pp. 371–384, 2006.

[27] L. Zhao *et al.*, "Geyser-2: The second prototype cpu with fine-grained run-time power gating," in *Proc. the 2011 ASP-DAC*, pp. 25–27, 2011.