[8] J. Donald and M. Martonosi, "Leveraging simultaneous multithreading for adaptive thermal control," in *Proc. 2nd Workshop Temperature-Aware Computer Systems (TACS) in Conjunction With ISCA-32*, 2005.
[9] Wei *et al.*, "Low voltage low power CMOS design techniques for deep submicron ICs," *Proc. VLSI Design*, pp. 24–29, Jan. 2000.
[10] *Intel Nehalem Processor (i7)*, [Online]. Available: http://www.intel.com/products/processor/corei7/index.htm
[11] Hu *et al.*, "Microarchitectural techniques for power gating of execution units," *Proc. ISLPED*, pp. 32–37, 2004.
[12] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," in *Proc. IEEE*, Feb. 2003, vol. 91, no. 2, pp. 305–327.
[13] Tullsen *et al.*, "Simulation and modeling of a simultaneous multithreading processor," in *Proc. Computer Measurement Group Conf.*, 1996, pp. 819–828.
[14] Choi *et al.*, "Learning-based SMT processor resource distribution via hill-climbing," in *Proc. ISCA*, 2006, pp. 239–251.
[15] Hrishikesh *et al.*, "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays," *Proc. ISCA*, pp. 14–24, May 2002.
[16] Heo *et al.*, "Power-optimal pipelining in deep submicron technology," *Proc. ISLPED*, pp. 218–223, 2004.
[17] Anis *et al.*, "Design and optimization of multi-threshold CMOS (MTCMOS) circuits," *Trans. CAD*, vol. 22, no. 10, pp. 1324–1242, Oct. 2003.
[18] Parekh *et al.*, "Thread-sensitive scheduling for SMT processors," Comput. Sci. Eng. Dept., Univ. of Washington, Seattle, WA, Tech. Rep., 2000.

Register File Partitioning and Compiler Support for Reducing Embedded Processor Power Consumption

Xuan Guan and Yunsi Fei

***Abstract*—Register file (RF) in modern embedded processors contributes a substantial budget in the energy consumption due to its large switching capacitance and long working time. For embedded processors, on average 25% of registers count for 83% of RF accessing time. This motivates us to partition the RF into hot and cold regions, with the most frequently used registers placed in the hot region, and the rarely accessed ones in the cold region. We employ the techniques of bit-line splitting and drowsy register cell to reduce the overall accessing power of RF. We propose a novel approach to partition the RF in a way that can achieve the largest power saving. We formulate the RF partitioning process into a graph partitioning problem, and apply an effective algorithm to obtain the optimal result. We evaluate our algorithm on MiBench and SPEC2000 applications, and an average saving of 58.3% and 54.4% over the non-partitioned RF accessing power is achieved for the SimpleScalar PISA system, respectively. The area overhead is negligible, and the execution time overhead is acceptable.**

***Index Terms*—Compilers, low-power design, partitioning, processor architectures.**

I. Introduction

As embedded systems are being widely used in communications, multimedia, and networking areas, low power consumption has remained one of the most critical concerns in embedded processor design [1]. With the increase of register file (RF) size and accessing time, RF has been a major source of power dissipation in modern embedded processors. For example, the RF power reaches 25% of the total processor power for a pre-synchronization embedded processor for multimode software-defined radio (SDR) terminals in [2]. In a recent network-on-chip (NOC) platform, the RF consumes up to 37.1% of the total system power consumption, which becomes the power bottleneck rather than the heavily used communication architecture [3].

In this paper, we focus on reducing the RF power consumption by employing compiler support for a partitioned RF. Our work is motivated by the observation in [4] that in a single-issue pipelined RISC processor, 25% of registers count for 75%–92% (on average 83%) of total RF accesses, a phenomenon known as 80/20 rule [5]. We put the most frequently used registers in a small section of the RF, which consumes much less power than the whole RF, and thus the overall RF accessing power consumption can be greatly reduced.

A. Related Work

System-level optimizations for reducing the RF power consumption can be classified into two major categories: reducing the RF access frequency and reducing the power consumption of each access. For the second category, one popular approach is RF window. In Rajiv's work [6], the RF is organized as a set of identical register windows with their own address space. Power consumption is saved by activating only a single window during execution. However, the window activating and data transfers between windows incur maintenance overhead. Lin *et al.* presented a distributed ping-pong RF organization optimized for stream processing [7]. Each functional unit has direct access to eight private registers, and the other eight public registers are dynamically shared by a pair of functional units. By this way the average number of registers in use is reduced. Ayala *et al.* employed the similar idea as ours of only activating the registers used by a code section, putting other unused registers into a low-power ("drowsy") state [8], [9]. However, their RF partition is based on analysis of one or two most time-consuming functions, instead of majority of the program code. In addition, they have fixed the RF partition configuration (1/4 for the small region), and our approach explores all possible configurations and finds the best one. They inserted instructions at the beginning and end of code sections to turn a set of registers on and off, which increases the code size and causes cycle overheads.

B. Paper Overview

Our work falls into the second category, and targets reducing the RF access power throughout the whole program. We divide the register file into two regions that share the same address space, where one has the smaller number of registers but is used more frequently and the other larger region but used less. Instead of simply picking the most frequently used registers for the hot region by profiling, as used in [8], [9], we have devised a sophisticated algorithm to find the best candidates based on the original register allocation result. We also explore different partition configurations to find the best one. We extend the code-generation stage with an additional register renaming process to avoid complex compiler modification.

The rest of the paper is organized as follows. Section II introduces the overall framework of the compiler system we enhanced for RF partitioning. In Section III, we discuss the sources of power dissipation and derive the RF power model. In Section IV, we formulate the RF partitioning process into a graph partitioning problem. Section V presents the algorithm we proposed to solve this graph partitioning problem. Section VI outlines the experimental environment and provides the experimental results with analysis. Finally, Section VII draws conclusions.

Manuscript received October 02, 2008; revised January 23, 2009; accepted March 16, 2009. First published September 15, 2009; current version published July 23, 2010. This work was supported by National Science Foundation under the grant CCF-0541102.

The authors are with the University of Connecticut, Electrical and Computer Engineering, U-2157, Storrs, CT 06269 USA (e-mail: xug06002@engr.uconn.edu; yfei@engr.uconn.ed).

Digital Object Identifier 10.1109/TVLSI.2009.2020860

```
void sum(void){                    1. copy R0, #0
  int i, j, s;                     2. copy R1, #0
  s = 0;                       B0: 3. copy R2, #0
  for(i = 0; i < 10; i++){     B1: 4. add R3, R1, R2
    for(j = 0; j < 10; j++){       5. add R0, R0, R3
      s+ =i + j;                   6. inc R2
    }                              7. cmp R2, #10
}                                     br   1, B1
                                   8. inc R1
                                   9. cmp R1, #10
                                      br   1, B0
                                      ret
```

(a) (b)

Fig. 1. Example program code. (a) source C code, (b) assembly code.

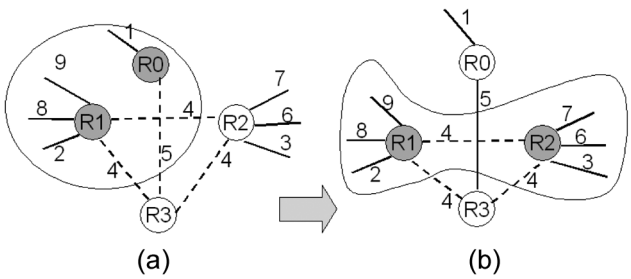

Fig. 2. Register allocation graph with RF partitioning. (a) Before reallocation; (b) after reallocation.

II. General Framework of Our Register File Partitioning and Compiler Support

The profiling result we get for MiBench applications[1] shows that the top 25% registers (in terms of usage frequency) are involved in 66% of total RF access, and 50% registers in 87% of usage [10]. However, it does not mean that placing the most frequently used 25% registers in the hot region will yield the best energy saving. Here we have to consider other factors, such as instructions using registers from different regions and the switching time overhead.

We first describe the register allocation problem with a simple example. Fig. 1(a) gives the original C code and (b) shows the assembly code of a sample program on a platform of SAMSUNG CalmRISC8 [11]. We only label the nine instructions that access the RF.

We then translate the program code register allocation into a graph in Fig. 2(a), where each node represents a register, and each edge represents an instruction, which connects the registers it uses with the instruction number labeled on the edge. Assume the RF is partitioned into two regions at the same size, (R0, R1) and (R2, R3), as shown in Fig. 2(a). With the original register allocation, 2 instruction (4 and 5) are crossing the regions, 4 instructions (1, 2, 8, and 9) are only using region 1, and 3 instructions (3, 6, and 7) region 2 only. However, since most of the instructions are accessing register R1 and R2, if we put R1 and R2 into region 1, as shown in Fig. 2(b), there will be only one instruction (in dashed line: 4) using both of the two regions, and it may result in less power consumption than the original allocation. We implement the RF partitioning result of Fig. 2(b) by renaming the registers. For example, all the R0 and R2 in the original code are swapped. This example shows that the original register allocation is not the most energy-efficient one without considering RF partitioning.

III. Register File Power Model

In this section we will examine the hardware structure and the low power techniques for the partitioned register file.

[1]Available [online] at http://www.eecs.umich.edu/mibench/

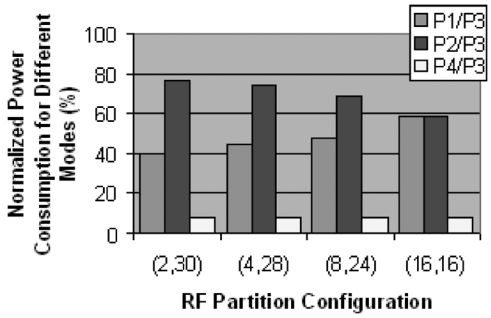

Fig. 3. Access power consumptions for different working modes of the partitioned register file.

A. Register File Access Energy Consumption and Power Reduction Techniques

The overall RF accessing energy includes both dynamic and leakage energy dissipation. There are four constituents of the dynamic energy consumption: the word-line energy, bit-line energy, sense amplifier energy, and energy for driving control signals. According to previous research [12], the bit-line is the dominant component, taking up to 70% of the dynamic energy consumption of RF, while each of the other parts consumes no more than 10% of total energy. The leakage energy exists for all the components even when the registers are not accessed. It could be a significant source of power consumption as the technology keeps shrinking. For example, it takes less than 1% of the total power consumption under 0.18-μ technology, and with the 90–nm technology, it is approaching 50% [13], [14].

We have employed two techniques to reduce the average power consumption of RF, bit-line splitting and drowsy technique.

Since the power dissipation of bit-lines is linearly dependent on the number of registers on the bit-line [12], the dynamic power consumption can be reduced by splitting the bit-line to several segments and adding a multiplexer to choose the output from them [4]. Here we only consider two segments, and keep the size of the small region a power of 2, so that some RF address lines can be used directly to control the multiplexers for selecting the bit-line, without additional control instructions or complex control circuits. Hence, the time for selecting a bit-line can be reduced.

The static power consumption of the unused RF region can be reduced by applying the "drowsy" technique [9], [15], which can preserve the register contents by adaptively scaling the supply voltages (0.3 V or 1 V) of each bit-line. The hardware overhead and switching time between register modes have been evaluated in [15]. With HSPICE simulations and CACTI model for a 70 nm process, it is found that the wake-up time for the register file from the drowsy mode to normal active mode and the reverse switching time is one-cycle, and the area overhead of the voltage controller and bit-line split logic is less than 3% of the whole RF. We will consider this mode switching overhead in our later experiments.

B. Register File Power Model

Based on the aforementioned power saving techniques, an average power model for accessing the RF during a program execution is derived as follows. $P = (\alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 + \delta \cdot P_4)/(\alpha + \beta + \gamma + \delta)$, where P_1, P_2, P_3, and P_4 represent the RF accessing power for four possible working modes, i.e., only region 1 is accessed, only region 2 is accessed, both regions are used, and none of the two regions is accessed. $\alpha, \beta, \gamma, \delta$ are the total instruction counts in each mode, respectively. The value of P_1, P_2, P_3, and P_4 includes both the dynamic and static power, and it depends on specific RF partition configuration. Under the 70 nm process, with the bit-line split and drowsy register techniques applied, Fig. 3 shows the ratio of P_1, P_2, and P_4 to the original 32-entry RF power

TABLE I
MATRIX REPRESENTATION OF PROGRAM CODE

$M_{4\times9}$	$\mathbf{I}_1$	$\mathbf{I}_2$	$\mathbf{I}_3$	$\mathbf{I}_4$	$\mathbf{I}_5$	$\mathbf{I}_6$	$\mathbf{I}_7$	$\mathbf{I}_8$	$\mathbf{I}_9$
R0	1	0	0	0	1	0	0	0	0
R1	0	1	0	1	0	0	0	1	1
R2	0	0	1	1	0	1	1	0	0
R3	0	0	0	1	1	0	0	0	0
Sum_I	1	1	1	3	2	1	1	1	1

consumption, P_3. The ratios for P_1 and P_2 range from 40% to 75%, on average 58%. They vary for different partition configurations. The RF idle power P_4 is independent of the partition configuration. At any time, the whole RF stays in only one of these four working modes.

IV. REGISTER FILE PARTITIONING PROBLEM FORMULATION

With this power model, our partitioning goal is to put the most frequently used registers in a small region, so that α, β, and γ are distributed in a way that the average power consumption is minimized. This is an NP-complete problem. Since the value of δ is determined by the program itself, its contribution to the average RF accessing power will not be affected by our partition algorithm. The problem is simplified to minimizing the average power when RF is accessed, $P' = (\alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3)/(\alpha + \beta + \gamma)$. We formulate this problem into a graph partitioning problem. The graph includes all the information in the program code, like the existing register allocation results, the relationship between registers (whether independent or used in the same instruction), etc. For a realistic program, the register allocation graph could be very large and difficult to analyze. We generalize the program code representation to a more formal mathematic form and seek the best RF partitioning scheme.

Table I demonstrates a matrix representation of the sample program. In matrix M, each row represents a register, and each column denotes an instruction. The sample code in Fig. 1 uses 4 registers by 9 instructions, and it is transformed into a 4×9 matrix. The matrix element $M[i,j]$ ($i = 1 \sim 4, j = 1 \sim 9$) is set to 1 if instruction j uses register $i-1$, otherwise it is set to 0. In the last row, each element in Sum_I is the sum of values in the specific column of matrix M, representing the number of registers that the corresponding instruction has accessed.

Suppose the RF size is N and the total number of instructions in the program code that use registers is N_{IR}, we define a $1 \times N$ vector, X, to represent the partitioning scheme. For register $i-1$, $X[i] = 1$ means it is in region 1 (the smaller region), and $X[i] = 0$ means it is in region 2. For example, for the partitioning scheme of R1 and R2 in region 1 and R0 and R3 in region 2, there is $X = [0,1,1,0]$. We define a $1 \times N_{IR}$ vector $B = X * M$, where each element represents the number of registers in region 1 that the corresponding instruction has used. For example, $B[3] = X * M[3] = \sum_{i=1}^{4}(X_i * M[i,3])$, where each element of X (X_i) indicates if the corresponding register $(i-1)$ is in region 1, and each element in the third column of M ($M[i,3]$) represents whether instruction I_3 uses the register $(i-1)$ or not. For the sample code and the example RF partitioning scheme in Fig. 2(b), $B = [0,1,1,2,0,1,1,1,1]$. If $B[j] = Sum\_I[j]$, it means that all the registers that instruction j has used are in region 1. Thus, we get the value of α, the number of instructions that only access RF region 1, as follows.

- α number of zeros in $(Sum\_I - B)$. For the example Sum_I and B, we have α=6.
- β number of zeros in B. $B[j] = 0$ means that none of the registers used by instruction j is in region 1. For example, instructions I_1 and I_5 do not use registers in region 1, as shown in Fig. 2(b), thus, $\beta = 2$.
- γ $N_{IR} - \alpha - \beta$, is the number of instructions that use both region 1 and region 2. In the example, $\gamma = 1$.

Based on the analysis above, the graph partitioning problem is formulated into a mathematic form as below. Given a program code and certain RF, we have the following:

- N total number of registers in the RF;
- N_{IR} total number of instructions in the code that use registers;
- M the register allocation matrix with size of $N \times N_{IR}$, where

$$M[i,j] = \begin{cases} 1 & : \text{ instruction } j \text{ uses register } i-1 \\ 0 & : \text{ instruction } j \text{ does not use register } i-1. \end{cases}$$

We define the RF partition vector X with size $1 \times N$:

$$X[i] = \begin{cases} 1 & : \text{ register } i-1 \text{ is in region 1(smaller region)} \\ 0 & : \text{ register } i-1 \text{ is out of region 1.} \end{cases}$$

The value of α, β, and γ is expressed as a function of $X[i]$ and $M[i,j]$, as shown in (1).

$$\begin{aligned} \alpha &= \sum_{j=1}^{N_{IR}} \prod_{i=1}^{N} (1 - M[i,j] + X[i] \cdot M[i,j]) \\ \beta &= \sum_{j=1}^{N_{IR}} \prod_{i=1}^{N} (1 - X[i] \cdot M[i,j]) \\ \gamma &= N_{IR} - \alpha - \beta. \end{aligned} \tag{1}$$

Our algorithm is to look for the best value for vector X, so that the average RF accessing power, $P' = (\alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3)/(\alpha + \beta + \gamma)$, is minimized.

V. ALGORITHM DESCRIPTION

We next elaborate on the algorithm that we propose to address the RF partitioning problem. Other than the expensive exhaustive search, we adopt the widely-used Kernighan–Lin algorithm for this partitioning problem [16], which has much lower computation complexity and reasonable optimization result. Note that other partitioning heuristics may also work. We put the K-L algorithm for a fixed partition configuration in the inner loop, and explore different configurations in the outer loop. We will compare all the largest power saving value for each configuration, and find the best partition configuration and the register allocation result.

The partition process we have discussed above is within one basic block. In real applications, a program may have many functions, and each function may contain multiple basic blocks. Considering the large number of basic blocks in a program, applying our partitioning algorithm to the whole program may be very time-consuming. We need to find the best range of basic blocks under consideration that can deliver the largest power consumption for the whole program. Since each basic block has its own execution frequency, i.e., importance, we choose the most important basic blocks to optimize. The optimization flow is as follows.

1) Rank all the basic blocks in a decreasing order of execution frequency $\times$ block size.
2) Do RF partitioning within the first ranked basic block, and get the largest power saving result for the whole program, Saving_1, and the RF partition result X_1.
3) Consider the second ranked basic block together with the first one, start at the last partition result X_1 and repeat the K-L searching algorithm to find the best RF partitioning for them, X_2, and the largest power saving result, Saving_2.
4) Each time start from the last partitioning result X_{i-1}, and continue considering more basic blocks until the power saving value has not increased for five steps.

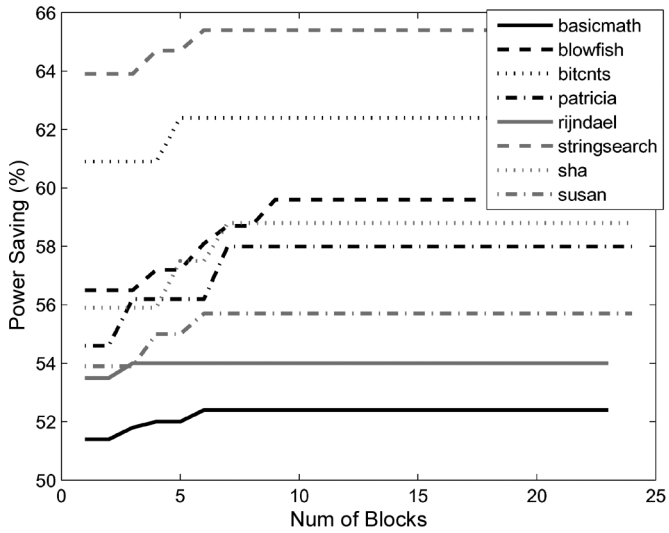

Fig. 4. Power saving results for considering multi-blocks.

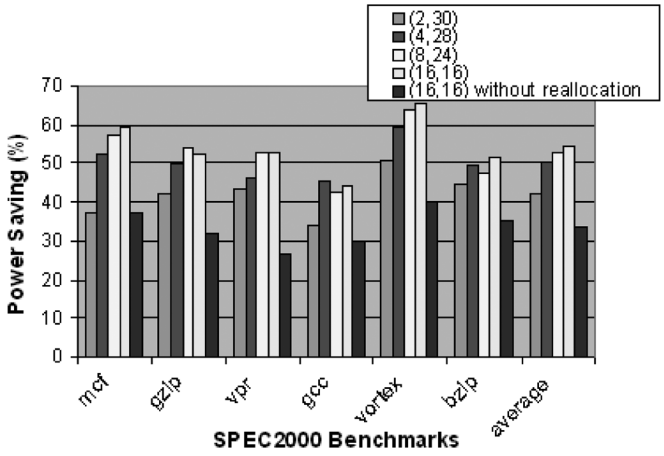

Fig. 5. Power saving results for SPEC2000 benchmarks on the PISA system.

Since the more frequently executed and the larger basic blocks have better potential in saving RF power consumption, the first several basic blocks we optimize would have the largest influence over the whole program power consumption. Each time with a new basic block added for consideration, we start from the last best partition scheme which achieves the highest power saving for the whole program, and perform the K-L algorithm on the group of basic blocks together. We have run eight benchmark applications in MiBench without a stop criterion, i.e., exploring all the basic blocks in the program, and output the power saving result for each exploration step in Fig. 4. For most of the time, the first value that keeps to be the peak for five steps is very close to the final global maximum, and for all the benchmarks the saving starts to saturate when more than ten dblocks have been considered. Compared to the full exploration of all the basic blocks, our reasonable stop criterion can save the execution time greatly without sacrificing the quality of solutions.

VI. Experimental Results

In our experiments, we evaluate the effects of both different benchmarks and different architectures on RF power saving and execution time overhead. We first experiment with a set of integer benchmarks randomly picked from MiBench and SPEC2000[2] , which represent embedded applications and general purpose integer applications, respectively. All the benchmarks are compiled and profiled by the modified SimpleScalar, with all the information such as the original register allocation, basic block execution frequency, and control flow graph (CFG), obtained. We test our algorithm on top of the machine targeted by SimpleScalar, PISA, which is a RISC architecture with 32 registers in total. The RF is partitioned into two regions with four different partition configurations. We then test the embedded application benchmarks MiBench on two different architectures, X86 and Alpha.

A. Experimental Results for MiBench and SPEC2000 on PISA Architecture

1) Reduction in Register File Power Consumption: Fig. 5 shows the power saving result for 6 SPEC2000 benchmarks on the PISA system over the non-partitioned RF. Each benchmark has run for four RF partition configurations (2, 30), (4, 28), (8, 24), and (16, 16). In most cases, the configuration of (16, 16) gets the best saving, and a few times (8, 24) gives a better result. The best power saving ranges from 44.4% to 65.9%. We also compare our power saving result with the one that has the same partitioned RF but for the original register allocation. For configuration of (16, 16), our software RF optimization through register renaming achieves an additional 20.9% power reduction on average.

TABLE II
Best Partitioning Result for the MiBench on PISA System

Bench-marks	Power Saving	C_{best}	# of bb	Best Partition
basicmath	52.4%	(8, 24)	9	0,2,3,4,6,16,28,29
blowfish	59.7%	(8, 24)	12	2∼9
bitcnts	62.4%	(8, 24)	10	0, 2∼5,16,29,31
partricia	56.6%	(8, 24)	11	0,2∼5,7,8,15
rijndael	54.1%	(8, 24)	6	2∼6,8,13,14
stringsearch	65.4%	(8, 24)	11	0,2∼6,16,21
sha	60.4%	(16, 16)	10	0,2∼13,16,17,19
susan	55.7%	(8, 24)	13	0,2,3,5,6,8,10,15

The same experiments have been run for eight MiBench applications. For most applications, the best configuration is (8, 24), the average power saving is 58.3% for configuration (8, 24), which is 19.6% more than the power saving from the partitioned RF without reallocation. The best partition configuration for MiBench differs from that for SPEC2000. This result demonstrates the difference between general-purpose and embedded applications, e.g., general-purpose applications may use registers more uniformly than embedded applications.

Table II shows the detailed partitioning result for the eight MiBench applications on the PISA system, including the largest power saving, the best RF partition configuration, $C_{\rm best}$, the number of basic blocks considered for optimization, and the RF partition result (the registers that have their X element values as 1). The results obtained in Table II direct us to selected the best configuration (8, 24) to divide the RF through bit-line splitting and additional control logic. At the software level, we then rename the registers in the last column of Table II, i.e., rewrite the machine code.

2) Performance Degradation Evaluation: To evaluate the performance impact of the one-cycle overhead for switching between the drowsy and the normal active mode of the RF, Fig. 6 gives the execution time overhead for the two configurations of (8, 24) and (16, 16), for both MiBench and SPEC2000. The time overhead is small (on average 2.4% for SPEC2000 and 5.5% for MiBench on PISA). For most SPEC2000 applications, the (16, 16) configuration gives both large power saving and low execution time overhead. However, for MiBench, there is a trade-off between power consumption and performance under the two partition configurations.

[2]Available [online] at http://www.spec.org

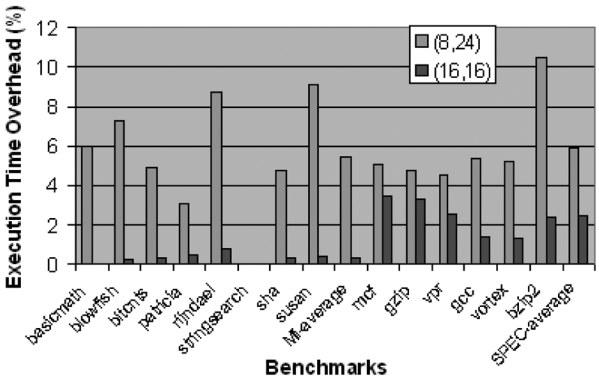

Fig. 6. Execution time overhead of MiBench and SPEC2000.

B. Experimental Results for MiBench on Alpha and X86 Architecture

We next experiment with MiBench on top of other two machines, Alpha and X86. For Alpha, a RISC architecture with 32 registers in total, in most cases partition (8, 24) gets the best saving, ranging from 49.1% to 61.8%. The X86 platform, a CISC architecture with only 16 registers, achieves the best power saving (56.1% on average) under partition (4, 12). We see that for MiBench, the 80/20 rule holds for all the three processor platforms, PISA, Alpha, and X86.

Mibench is also used in Ayala's work [8] for evaluating their power saving results on an embedded processor with 64 registers. Their average power saving reaches 65% when selecting two most time consuming functions to perform reconfiguration, which is slightly higher than our savings. The reason is that in our experiments, the target machines only have either 16 or 32 registers. Since their processor architecture is not publicly available, we cannot make direct comparisons on the same platform. We anticipate that our approach should achieve higher power saving than Ayala's if for the same number of registers (64).

VII. Conclusion

In this paper, we proposed a novel compiler-assisted RF partitioning and register reallocation approach to reduce the RF access power. The RF is partitioned into two regions without extra switching instructions. We applied the bit-line split technique to reduce the dynamic power consumption and switched the inactive region to a drowsy state at runtime to reduce its static power. The area overhead is negligible and the execution time overhead is acceptable. We formulated the problem into a graph partitioning optimization one and applied Kernighan–Lin algorithm for the optimal solution. Finally we tested our partitioning algorithm with MiBench and SPEC2000 applications on the PISA system, Alpha and X86. Compared to the non-partitioned register file, our approach reduces the RF access power consumption by about 58.3% on average for MiBench, and 54.4% for SPEC2000 on PISA.

References

[1] *Low Power Design Methodologies*, J. Rabaey and M. Pedram, Eds. Norwell, MA: Kluwer Academic, 1996.

[2] T. Schuster, B. Bougard, P. Raghavan, R. Priewasser, D. Novo, L. V. Perre, and F. Catthoor, "Design of a low power pre-synchronization ASIP for multimode SDR terminals," in *Proc. Int. Conf. Embedded Computer Systems: Architectures, Modeling Simulation*, Jul. 2007, pp. 322–332.

[3] A. Lambrechts, P. Raghavan, A. Leroy, G. Talavera, T. V. Aa, M. Jayapala, F. Catthoor, D. Verkest, G. Deconinck, H. Corporaal, F. Robert, and J. Carrabina, "Power breakdown analysis for a heterogeneous NoC platform running a video application," in *Proc. Application-Specific Systems, Architectures Processors*, Jul. 2005, pp. 179–184.

[4] J. H. Tseng and K. Asanovic, "Energy-efficient register access," in *Proc. Integrated Circuits Systems Design*, Sep. 2000, pp. 377–382.

[5] M. Gomez and V. Santonja, "Characterizing temporal locality in I/O workload," in *Proc. Int. Symp. Performance Evaluation Computer Telecommunication Systems*, Jul. 2002, pp. 76–82.

[6] R. A. Ravindran, R. M. Senger, E. D. Marsman, G. S. Dasika, M. R. Guthaus, S. A. Mahlke, and R. B. Brown, "Partitioning variables across register windows to reduce spill code in a low-power processor," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 998–1012, Aug. 2005.

[7] T.-J. Lin, S.-K. Chen, Y.-T. Kuo, C.-W. Liu, and P.-C. Hsiao, "Design and implementation of a high-performance and complexity-effective VLIW DSP for multimedia applications," *J. VLSI Signal Process.*, vol. 51, no. 3, pp. 209–223, Jun. 2008.

[8] J. L. Ayala, A. Veidenbaum, and M. L. Vallejo, "Power-aware compilation for register file energy reduction," *Int. J. Parallel Programm.*, vol. 31, no. 6, pp. 451–467, Dec. 2003.

[9] J. L. Ayala and A. Veidenbaum, "Reducing register file energy consumption using compiler support," presented at the Workshop on Application-Specific Processors, Istanbul, Turkey, Nov. 19, 2002.

[10] X. Guan and Y. Fei, "Reducing power consumption of embedded processors through register file partitioning and compiler support," in *Proc. Application-Specific Systems, Architectures Processors*, Jul. 2008, pp. 269–274.

[11] J. Park, J. Lee, and S. M. Moon, "Register allocation for banked register file," *ACM Special Interest Group on Programming Languages*, vol. 36, no. 8, pp. 39–47, Aug. 2001.

[12] X. M. Zhao and Y. Z. Ye, "Structure configuration of low power register file using energy model," in *Asia Pacific Conf. ASIC*, Aug. 2002, pp. 41–44.

[13] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage estimation considering power supply and temperature variations," in *Proc. Int. Symp. Low Power Electronics Design*, Aug. 2003, pp. 78–83.

[14] S. Borkar, "Getting gigascale chips: Challenges and opportunities in continuing Moore's law," *ACM Queue*, vol. 1, no. 7, pp. 26–33, Oct. 2003.

[15] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," in *Proc. Int. Symp. Computer Architecture*, May 2002, pp. 148–157.

[16] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. Singapore: World Scientific, 1999.