An Energy Reduction Scheduling Mechanism for a High-Performance SoC Architecture

Slo-Li Chu

Department of Information and Computer Engineering,
Chung Yuan Christian University, Chung-Li, Taiwan, R.O.C.
slchu@cycu.edu.tw

Abstract. Continuous improvements in semiconductor technology are supporting new classes of System-on-a-Chip (SoC) architectures that combine extensive processing logic with high-density memory. Such architectures are generally called Processor-in-Memory (PIM) or Intelligent Memory (I-RAM) and can support high-performance computing by reducing the performance gap between the processor and the memory. The PIM architecture combines various processors in a single chip. These processors are characterized by their computation, memory-access and power consumption capabilities. Therefore, a novel parallelizing system, SAGE II, has been developed to identify their capabilities and dispatch the most appropriate jobs to them in order to exploit the advantages of PIM architectures. However, the SAGE II system only can deal with performance issues but power consumption is gradually becoming an important issue of current computing systems. This paper provides a new low-power transformation mechanism, called Energy-Oriented Power Reduction Scheduling (EOPRS), to extend the capability of SAGE II system. It can reduce the power consumption for the Processor-in-Memory system without losing execution performance. The detailed EOPRS transformation technique is presented later. The experimental results of several benchmarks are also discussed.

Keywords: EOPRS, SAGE II, SoC, Processor-in-Memory, Power Reduction

1 Introduction

In current high-performance computer architectures, processors run many times faster than the computer's main memory. This performance gap is often referred to as the Memory Wall [2]. This gap can be reduced using the System-on-a-Chip (SoC) strategy, which integrates the processor(s) and memory on a single chip. Accordingly, many researchers have considered integrating computing logic and high density DRAM on a single die [3, 5, 8, 10] as so-called Processor-in-Memory (PIM) or Intelligent RAM (IRAM). This class of architectures constitutes a hierarchical hybrid multiprocessor environment that involves host (main) and memory processors. The host processor (P.Host) is more powerful but have a deep cache hierarchy and higher latency when accessing memory. In contrast, memory processors (P.Mem) are typically less powerful but have a lower latency in memory access. With respect to energy consumption, host processor consumes more energy than memory processors when computing and accessing memory. These capability differences make parallelization and energy reduction more difficult.

The main problems addressed are how to dispatch appropriate tasks to these various processors according to their characteristics, to reduce their execution time, energy consumption, or both, and how to partition the original program so that it can be executed on these heterogeneous processor mixtures. Accordingly, an automatic source-to-source parallelizing and optimizing system, SAGE II (Statement- Analysis- Grouping- Evaluation II) is designed and implemented. The main difference between SAGE II and other parallelizing systems is that it uses statement rather than iteration as the basic unit of analysis. By integrating statement splitting, weight evaluation and several scheduling mechanisms, SAGE II can automatically analyze Fortran source program; partition it into statements; generate a Weight Partition Dependence Graph (WPG), determine the weight of each block in the WPG, and schedule these blocks to improve the performance by dispatching the most suitable blocks for execution on host and memory processors. To extend the capability of reducing power consumption of software running on the SoC architecture, this study presents a novel scheduling technique, called Energy-Oriented Power Reduction Scheduling (EOPRS), for to reduce the energy consumed for the one-P.Host and one-P.Mem configuration without losing performance. The remainder of this paper is organized as follows: Section 2 introduces the related work. Section 3 describes the PIM architecture we adopt. Section 4 describes SAGE II system. Section 5 provides the EOPRS mechanism. The experimental results are shown in Section 6. Finally, conclusion is given in Section 7.

2 Related Work

Since energy limited the growth of the mobile computers and embedded computing devices, it is very important to reduce the consumption of energy and increase battery lifetimes. Therefore the analysis and optimization of the energy consumed by all components become important not only for battery-powered personal devices, but also for large computing systems.

Current work on energy saving using software can be classified into two categories. In one, special architecture is required to support the software. For example, an additional mini cache between the I-Cache and the CPU core called the “L-Cache” or “I-Filter” can be used [1, 5, 6]. Such mini cache buffers only the reused instructions within loops. It consumes less energy than I-Cache and so eliminates the fetching of unnecessary instructions fetching, reduce signal switching activity and dissipated energy. It requires compilers to cooperate to the rearrange original codes to ensure that it fits the L-Cache. However, it also has the disadvantage of requiring a special architecture.

Another category of work considers instruction-level scheduling mechanisms to reduce energy consumption [4, 7, 9, 12]. If a previously executed instruction differs from the present instruction, the control circuit must activate different hardware functional units. Switching consumes energy, so appropriately reordering instructions in a program reduce the energy consumption. This method need to measure the inter-instruction effects and establishes an energy transition cost table, before performing instruction-level scheduling according to the cost of switching. However, such a method has two disadvantages. First, the source program must be written in a low-level language such that the scheduler cannot understand all of the behavior of this program, limiting the application of high-level transformation techniques, such as tiling. Second, the cost of switching instructions is only around 2% ~ 10% of the base cost of the instructions; therefore the energy reduction is not significant.

3 The Processor-in-Memory Architecture

Figure 1 depicts the organization of the PIM architecture evaluated in this study. It contains an off-the-shelf processor, P.Host, and a PIM chip. The PIM chip integrates one memory processor, P.Mem, with 64 Mbytes of DRAM. This architecture is derived from FlexRAM [5]. The tiny memory processors (P.Array) in the original FlexRAM are omitted to reduce the complexity of analysis. The techniques presented in this paper involve a one P.Host and one P.Mem configuration, and can be extended to support multiple P.Mems.

Table 1 and Table 2 list the main architectural parameters of the PIM architecture. P.Host is a six-issue superscalar processor that allows out-of-order execution and runs at 800MHz, while P.Mem is a two-issue superscalar processor with in-order capability and runs at 400MHz. There is a two-level cache in P.Host and a one-level cache in P.Mem. P.Mem has lower memory access latency than P.Host since the former is integrated with DRAM. Thus, computation-bound codes are more suitable for running on the P.Host, while memory-bound codes are preferably running on the P.Mem to increase efficiency.

The PIM chip is designed to replace regular DRAMs in current computer systems, and must therefore conform to a memory standard that involves additional power and ground signals to support on-chip processing. One such standard is Rambus, so the PIM chip is designed with a Rambus-compatible interface.

Table 1. Major parameters of the PIM architecture.

P.Host	P.Mem	Bus & Memory
Working Freq: 800 MHz Dynamic issue Width: 6 Integer unit num: 6 Floating unit num: 4 FLC_Type: WT FLC_Size: 32 KB FLC_Line: 64 B FLC_Replace policy: LRU SLC_Type: WB SLC_Size: 256 KB SLC_Line: 64 B Replace policy: LRU Branch penalty: 4 P.Host_Mem_Delay: 88	Working Freq: 400 MHz Static issue Width: 2 Integer unit num: 2 Floating unit num: 2 FLC_Type: WT FLC_Size: 16 KB FLC_Line: 32 B FLC_Replace policy: LRU SLC: N/A Branch penalty: 2 P.Mem_Mem_Delay: 17	Bus Freq: 100 MHz P.Host Mem RT: 262. 5 ns P.Mem Mem RT: 50. 5 ns Bus Width: 16 B Mem_Data_Transfer: 16 Mem_Row_Width: 4K

** FLC stands for the first level cache, SLC for the second level cache, BR for branch, RT for round-trip latency from the processor to the memory, and RB for row buffer.*

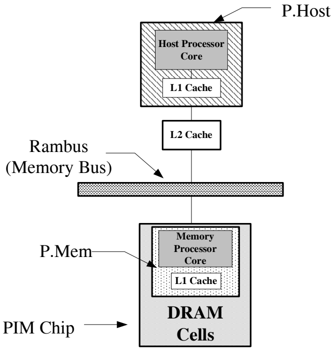

Fig. 1. Organization of the PIM architecture.

Table 2. Power consumption parameters of the PIM architecture.

	P.Host		P.Mem	
Integer Unit	ALU	194.46	ALU	97.23
	Shift	194.46	Shift	97.23
	Mult/div	210.45	Mult/div	105.23
	Branch	194.46	Branch	97.23
Load/Store Unit	Load	194.46	Load	97.23
	Store	194.46	Store	97.23
Floating point Unit	Add/Sub	210.45	Add/Sub	105.23
	Mult	210.45	Mult	105.23
	Div	210.45	Div	105.23
Clock energy	957		26.24	

4 The SAGE II System

Most current parallelizing compilers focus on transforming loops to execute iterations concurrently, in a so-called iteration-based approach. This approach is suited to homogeneous and tightly coupled multi-processor systems. However, it has an obvious disadvantage for heterogeneous multi-processor platforms because iterations have similar behavior but the capabilities of heterogeneous processors are diverse. Therefore, a different approach is adopted here, using the statements in a loop as a basic analysis unit, called statement-based approach, to develop the SAGE system.

SAGE II (Statement-Analysis-Grouping-Evaluation II) is an automatic parallelizing compiler, which partitions and schedules an original program to exploit the specialties of the host and the memory processor. At first, the source program is split into blocks of statements according to dependence relations. Then, the Weighted Partition Dependence Graph (WPG) is generated, and the weight of each block is evaluated. Finally, the blocks are dispatched to either the host or the memory processors, according to which processor is more suitable for executing the block. The major difference between SAGE II and other parallelizing systems is that it uses a statement rather than an iteration as the basic unit of analysis. This approach can fully exploit the characteristics of statements in a program and dispatch the most suitable tasks to the host and the memory processors.

Table 3 presents a simple example to demonstrate the advantages of statement-based parallelization. The program is fully parallelizable and can be partitioned into statements or iterations. The table lists the assumed statement weights for the P.Host and P.Mem. Table 4 shows five parallelization cases and their execution times. The first two involve executing the program solely on P.Host and P.Mem, respectively. Case 3 parallelizes the program using conventional parallelizing compilers, such as SUIF [2] or Polaris [2] to identify the parallelizable loops and dispatch them for execution on P.Host and P.Mem. This approach only achieves good speedup for processors with homogeneous capabilities (including memory access latency, computing power, and so on). In case 4, the iterations are dispatched to the processors according to the processors' capabilities, but the compiler does not consider the discrepancies among processors in executing statements. Case 5 uses the statement-based analysis approach (i.e., optimized by SAGE II). This approach outperforms all the others since it dispatches statements to P.Host and P.Mem by accounting for the characteristics of statements and the capabilities of processors, motivating the development of the SAGE II system for asymmetric multiprocessor environments. The compiling sequence of the SAGE II is shown in Figure 2. The major stages are described later.

Table 4. Five parallelizing cases and their execution times.

	Description	*Execution Time*
1	Host processor operates solely	Latency = [PH(S1)+PH(S2)+PH(S3)]*# of iterations = (3+5+6)N = 14 N
2	Memory processor operates solely	Latency =[PM(S1)+PM(S2)+PM(S3)]* # of iterations = (6+1+2)N = 9 N
3	Host and memory processors cooperate in symmetric workload	Latency = max((3+5+6)* 0.5N, (6+2+1)* 0.5 N) = 7 N
4	Host and memory processors cooperate in asymmetric workload by parallelizing the *iteration* space of the loop	The workload is dispatched based on the capability ratio of PH and PM obtained from Case 1 and Case2, i.e., PH: PM = 9:14. Hence, latency=14* (9/(9+14))N= 5.48 N
5	Host and memory processors cooperate in asymmetric workload by SAGE optimization	Latency = max (PH(S1) * N, PM(S2,S3)*N) = 3 N (Here S1 is more suitable for P.Host, but S2 and S3 are more suitable for P.Mem.)

Table 3. A fully parallelizable program.

Program	*Weight for *PH*	*Weight for *PM*
DO I = 1 to N		
S1: A= A mod B	3	6
S2: C= D[I]+E	5	1
S3: F= G[I]+H[I]	6	2
ENDDO		

**PH* stands for host processor and *PM* for memory processor.

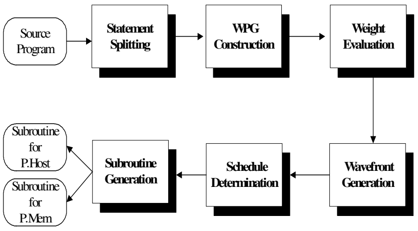

Fig. 2. The sequence of compiling stages in SAGE II.

4.1 Statement Splitting and WPG Construction [2]

Statement Splitting splits the dependence graph by Node Partitioning as introduced in [2]. WPG Construction constructs the Weighted Partition Dependence Graph (WPG), to be used in the subsequent stages of Weight Evaluation, Wavefront Generation and Schedule Determination.

The definitions relevant to Statement Splitting are introduced in the literature [2]. It is ignored for the page limitation but still listed the Statement Splitting algorithm (Algorithm 1) to partition the loops:

Algorithm 1. (Statement Splitting Algorithm)

```
Given a loop L = ($i_1, i_2, …. i_n$) ( $s_1, s_2, …. s_d$)
Step 1: Construct dependence Graph G by analyzing subscript expressions and index
        pattern.
Step 2: Establish a node partition Π on G as defined in Definition 2. If there are large
```

blocks caused by control dependence relations, convert control dependence into data dependence first [2], and then partition the dependence graph.

Step 3: On the partition Π, establish a weighted partition dependence graph WPG(B,E) defined in Definition 3.

4.2 Weight Evaluation

Two approaches to evaluating weight can be taken. One is to predict the execution time of programs by profiling the dominant parts [2]. The other considers the operations in a statement and estimates the program execution time by looking up an operation weight table [2]. The former method called code profiling may be more accurate, but the predicted result cannot be reused; the latter called code analysis can determine statements for suitable processors but the estimated program execution time is not sufficiently accurate. Hence, the Self-Patch Weight Evaluation scheme was designed to combine the benefits of both approaches. It integrates these two approaches together by analyzing code and searching weight table first to estimate the weight of a block. If the block contains unknown operations, the patch (profiling) mechanism is then activated to evaluate the weights of unknown operations. The obtained operation weights are added into the weight table for next look-up. For a detailed description of this scheme, please refer to [2]

5 Energy-Oriented Power Reduction Scheduling Mechanism

5.1 Algorithm Description

This paper has proposed an automatic source-to-source parallelizing compiler, SAGE II (Statement- Analysis- Grouping- Evaluation), which integrates statement splitting, weight evaluation, scheduling and optimizing mechanisms. However, the best execution schedule in terms of performance is not usually the same as the best execution schedule in terms of energy. This section focuses on reducing the energy consumption of PIM architecture. For the reason mentioned in the previous descriptions, statements are adopted as the analyzing units. Each statement unit has a delay weights and an energy weight for P.Host and P.Mem, respectively, and will be assigned to the most suitable processor according to these weights. Here we introduce the Energy-Oriented Power Reduction Scheduling, EOPRS, which allows the user to specify the energy limit. If the energy limit is met, the scheduler will use the performance-oriented principle to schedule the blocks. For simplicity, this paper considers only the system with a single P.Host and single P.Mem. The rest of this section presents the mechanisms in detail. Section 6 presents the experimental results.

This EOPRS approach enables the user to adjust the energy reduction ratio to get more speedup. In this approach, the energy weights and delay weights of each block in the WPG are determined first. Then the execution order for each block is determined according to their dependence relations. The blocks that can be executed simultaneously are assigned to a wavefront. Then the scheduler dispatches blocks in a single wavefront according to their energy weights, minimizing the energy consumption. According to the energy specified by the user, the maximum energy acceptable consumption, called the “deadline” is determined herein. The deadline is given by minimum energy consumption + (original energy consumption - minimum energy consumption) * (1 - energy reduced limit %). Then, the mechanism can automatically adjust schedule sequence to obtain the maximum potential speedup, such that the energy consumption is less than the maximum acceptable. The details mechanism of this approach is presented in Algorithm 2.

Algorithm 2: Energy-Oriented Power Reduction Scheduling

```
[Input]
    $WPG=(P, E)$: the order of blocks and weights are determined.
    Percent: the percent user want to constrain.
[Output]
    An execution wavefront schedule $WF= \{Wf_1, Wf_2, ....\}$ where $Wf_i=\{PH(b_a...b_b), PM(b_c...b_d)\}$ in which $PH(b_a...b_b)$
    means that blocks $b_a...b_b$ will be assigned to PHost in wavefront $i$, $PM(b_c...b_d)$ means that blocks $b_c...b_d$ will be assigned
    to PMem in wavefront $i$.
[Intermediate]
    W: a working set of blocks to be visited.
    max_pred_O($b_i$): the maximum execution order for all $b_i$'s predecessor blocks.
    min_pred_O($b_i$): the minimum execution order for all $b_i$'s predecessor blocks.
    wf_tmp: working sets of blocks for wavefront scheduling.
    ph_tmp(h), pm_tmp(m): working arrays to store the blocks for wavefront scheduling.
    ph_d_wei(h), pm_d_wei(m) working arrays to store the total delay weights of blocks .
    ph_e_wei(h), pm_e_wei(m) working arrays to store the total energy weights of blocks .
    max_wf: the maximum number of wavefront.
    PHEW_origin(max_wf): the energy weight of $b_i$ for Phost each wavefront.
    Deadline(max_wf): the deadline of each wavefront.
    PHDW($b_i$): the delay weight of $b_i$ for PHost.
    PMDW($b_i$): the delay weight of $b_i$ for PMem.
    PHEW($b_i$): the energy weight of $b_i$ for PHost.
    PMEW($b_i$): the energy weight of $b_i$ for PMem.
[Method]
for each  $b_i \in P$ do /*Initialization and weight determination for each blocks */
    $O_i = 0$; evaluate PHDW($b_i$), PMDW($b_i$), PHEW($b_i$) and PMEW($b_i$) by weight evaluation;
end for
W = P;
for each $b_i$ with no predecessors do /* Execution order assignment */
    $O_i = 1$; W = W – { $b_i$ };
end for
done = False; max_wf=0;
while done = False AND W $\neq \phi$ do
    done = True
    for each  $b_i \in$ W do
        if min_pred_O($b_i$) = 0  then done = False
        else
            $O_i$ = max_pred_O($b_i$)+1; W=W-{ $b_i$ }; max_wf = max(max_wf, $O_i$);
        end if
    end for
end while
energy_temp = 0
for j = 1 to max_wf
    wf_tmp=ph_tmp=pm_tmp={$\phi$};h=m=Opt_energy(j)=PHEW_origin(j)=Deadline(j)=0;
    ph_d_wei(j)=ph_e_wei(j)=pm_d_wei(j)=pm_e_wei(j)=0;
    store all of $b_i$ whose $O_i$ = j  in wf_tmp;
    for each  $b_i \in$ wf_tmp
        if  $PHEW(b_i) - PMEW(b_i) \leq 0$
            h = h + 1; Store $b_i$ in ph_tmp (h);
            ph_d_wei(j)=ph_d_wei(j)+PHDW($b_i$ ); ph_e_wei(j)=ph_e_wei(j)+PHEW($b_i$ );
        else
            m = m + 1; Store $b_i$ in pm_tmp (m);
            pm_d_wei(j)=pm_d_wei(j)+PMDW($b_i$ ); pm_e_wei(j)=pm_e_wei(j +PMEW($b_i$ );
        end if
        PHEW_origin(j)=PHEW_origin(j)+PHEW($b_i$ );
        Opt_energy(j) = Opt_energy(j) + Min(PHEW($b_i$ ), PMEW($b_i$ ));
        Sort ph_tmp (h) in decreasing order by PHDW($b_i$ );
        Sort pm_tmp (m) in decreasing order by PMDW($b_i$);
    end for
    Deadline(j)=Opt_energy(j)+(PHEW_origin(j)–Opt_energy(j))*(1–Percent);
    Deadline(j)=Deadline(j)+energy_temp; p = q = 0; done = FALSE;
    if  ph_d_wei(j) ≥ pm_d_wei(j)
        for p =1 to h
            ph_e_wei(j) = ph_e_wei(j) – PHEW ( ph_tmp(p))
            pm_e_wei(j) = pm_e_wei(j) + PMEW ( ph_tmp(p))
            ph_d_wei(j) = ph_d_wei(j) – PHDW ( ph_tmp(p))
            pm_d_wei(j) = pm_d_wei(j) + PMDW ( ph_tmp(p))
            if  ph_e_wei(j) + pm_e_wei(j) ≤ Deadline(j)
                ph_tmp = ph_tmp – {ph_tmp(p)}; pm_tmp = pm_tmp + {ph_tmp(p)};
            else
                pm_d_wei(j) = pm_d_wei(j) – PMDW ( ph_tmp(p))
                ph_d_wei(j) = ph_d_wei(j) + PHDW ( ph_tmp(p))
                pm_e_wei(j) = pm_e_wei(j) – PMEW ( ph_tmp(p))
                ph_e_wei(j) = ph_e_wei(j) + PHEW ( ph_tmp(p))
            end if
        end for
    else
        for q =1 to m
            ph_e_wei(j) = ph_e_wei(j) + PHEW ( pm_tmp(q))
            pm_e_wei(j) = pm_e_wei(j) – PMEW ( pm_tmp(q))
            ph_d_wei(j) = ph_d_wei(j) + PHDW ( pm_tmp(q))
            pm_d_wei(j) = pm_d_wei(j) – PMDW ( pm_tmp(q))
            if  ph_e_wei(j) + pm_e_wei(j) ≤ Deadline(j)
                pm_tmp = pm_tmp – {pm_tmp(q)}
                ph_tmp = ph_tmp + {pm_tmp(q)}
            else
                ph_d_wei(j) = ph_d_wei(j) – PHDW ( pm_tmp(q))
                ph_e_wei(j) = ph_e_wei(j) – PHEW ( pm_tmp(q))
                pm_d_wei(j) = pm_d_wei(j) + PMDW ( pm_tmp(q))
                pm_e_wei(j) = pm_e_wei(j) + PMEW ( pm_tmp(q))
            end if
        end for
    end if
    energy_temp=Deadline(j)–( ph_e_wei(j)+pm_e_wei(j)); $Wf_j$ ={ph_tmp, pm_tmp };
end for
```

6. Experimental Results

This experiment focuses on the one-P.Host and one-P.Mem configuration of PIM architecture to evaluate the energy reduction scheduling mechanism mentioned in Section 5. The energy consumption parameters of this PIM architecture can be found in the Table 2.

The evaluated applications include five programs - swim from SPEC95, cgemm from BLAS3, ft and cg from the serial version of NAS, and mdg from Perfect Benchmark. Table 5 shows the results of the experiment. "Std-" denotes that the applications are executed on P.Host alone; "Opt-" denotes that the applications are transformed by the one-P.Host and one-P.Mem performance scheduling mechanism mentioned in literature [2] for execution on one P.Host and one P.Mem simultaneously; "Ene-" denotes that the applications are transformed by "Energy-Oriented Energy Reduction Scheduling" as mentioned in Section 5 with the energy reduced ratio set to 70%.

In Table 5, swim, cg, and cgemm exhibit good speedup "Opt-" case because they all can be partitioned into many blocks for scheduling to P.Host and P.Mem, according to the characteristics of blocks and processors. Also, they can achieve a good energy reduction ratio; ft can obtain good speedup "opt-" case but some performance in "Ene-" case is lost because its block sizes are very different. Scheduling these blocks is difficult when the power reduction ratio is set to 70%. In contrast, mdg is intrinsically sequential. It can only be partitioned into several large blocks, preventing the generation of load-balance schedules, so the speedup of "Opt-" case is only 1.08. The EOPRS can further reduce the energy but the speedup is decreased to 0.45. Thus, the user can select different scheduling approaches according to their requirements.

Table 5. Experimental results of five benchmarks.

benchmark	form	Execution cycles	Speedup	Energy consumption (joule)	Energy Reduce Ratio
swim	Std-	228289321	1.00	43.49	0
	Opt-	116669760	1.96	35.29	18.85%
	Ene-	262401518	0.87	28.18	35.20%
cg	Std-	91111840	1.00	19.58	0
	Opt-	51230772	1.78	14.73	24.77%
	Ene-	67993910	1.34	12.66	35.34%
cgemm	Std-	257528272	1.00	50.26	0
	Opt-	147769271	1.74	38.63	23.14%
	Ene-	238641808	1.07	33.02	34.30%
ft	Std-	544099032	1.00	111.9	0
	Opt-	368981380	1.47	86.59	22.62%
	Ene-	549370943	0.99	74.41	33.50%
mdg	Std-	174506369	1.00	65.5	0
	Opt-	160335813	1.08	63.95	2.37%
	Ene-	385931357	0.45	41.68	36.37%

7. Conclusions

In this paper, we presented a new program transformation technology, Energy-Oriented Power Reduction Scheduling, EOPRS. Integrated with SAGE II system, EOPRS reduces the energy consumption on the PIM System. This approach allows the user to set the speedup limit. If the speedup limit is met, then the scheduler will use the energy reduction principle to schedule the blocks. Results of the experiments demonstrate the capabilities of the extended SAGE II system. In the experiment, the energy consumption was reduced by up to 36.37%. The author believe that the techniques proposed here can be extended to run on DIVA, EXECUBE, FlexRAM, and other high-performance SoC architectures by slightly modifying the code generator of the SAGE system.

Acknowledgement

This work is supported in part by the National Science Council of Republic of China, Taiwan under Grant 94-2213-E-033-032-

References

[1] Bajwa, R. S., et al.: Instruction Buffering to Reduce Power in Processors for Signal Processing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.5(4). IEEE, (1997) 417-424.

[2] Chu, S. L., Huang, T. C.: SAGE: A Comprehensive Parallelizing Framework for Processor-in-Memory Architectures. Journal of Systems Architecture. Vol. 50 (1). Elsevier Science, (2004) 1-15.

[3] M. Hall, et al,: Mapping Irregular Applications to DIVA, a PIM-Based Data-Intensive Architecture. Proc. of 1999 Conference on Supercomputing. ACM, (1999).

[4] Horowitz, M., Indermaur, T., Gonzalez, R.: Low-Power Digital Design. Proc. of Symposium on Low Power Electronics. 8-11, (1994).

[5] Kang, Y., et al.: FlexRAM: Toward an Advanced Intelligent Memory System. Proc. of International Conference on Computer Design. (1999).

[6] Kin, J., et al.: The filter cache: an energy efficient memory structure. Proc. of Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture. (1997) 184-193.

[7] Lee, L. H., Moyer, B., Arends, J.: Instruction Fetch Energy Reduction Using Loop Caches for Embedded Applications with Small Tight Loops. Proc. of 1999 International Symposium on Low Power Electronics and Design. (1999) 267-269.

[8] Oskin, M., Chong, F. T., Sherwood, T.: Active Page: A Computation Model for Intelligent Memory Computer Architecture. Proc. 25th Annual International Symposium on Computer Architecture. (1998) 192 –203.

[9] Parikh, A.., et al.: Energy-Aware Instruction Scheduling. Proc. of 7th International Conference on High Performance Computing-HiPC (2000) 335-344.

[10] Patterson, D., et al.: A Case for Intelligent DRAM. IEEE Micro, IEEE (1997) 33-44.

[11] Press,W. H., et al., Numerical Recipes in Fortran 77. Cambridge University Press (1992).

[12] Raghunathan, A., Jha, N. K., Dey, S.: High-Level Power Analysis and Optimization. Kluwer Academic Publishers, (1998).

[13] Tiwari, V., Malik, S., Wolfe, A.: Power Analysis of Embedded Software: a First Step Towards Software Power Minimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 2 (4). IEEE (1994) 437-445.