IEICE TRANSACTIONS

on Information and Systems

VOL. E99-D NO. 2
FEBRUARY 2016

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Performance of Dynamic Instruction Window Resizing for a Given Power Budget under DVFS Control

Hideki ANDO[†a)] *and* Ryota SHIOYA[†b)], *Members*

SUMMARY Dynamic instruction window resizing (DIWR) is a scheme that effectively exploits both memory-level parallelism and instruction-level parallelism by configuring the instruction window size appropriately for exploiting each parallelism. Although a previous study has shown that the DIWR processor achieves a significant speedup, power consumption has not been explored. The power consumption is increased in DIWR because the instruction window resources are enlarged in memory-intensive phases. If the power consumption exceeds the power budget determined by certain requirements, the DIWR processor must save power and thus, the performance previously presented cannot be achieved. In this paper, we explore to what extent the DIWR processor can achieve improved performance for a given power budget, assuming that dynamic voltage and frequency scaling (DVFS) is introduced as a power saving technique. Evaluation results using the SPEC2006 benchmark programs show that the DIWR processor, even with a constrained power budget, achieves a speedup over the conventional processor over a wide range of given power budgets. At the most important power budget point, i.e., when the power a conventional processor consumes without any power constraint is supplied, DIWR achieves a 16% speedup.

key words: *microprocessor, superscalar processor, memory-level parallelism, instruction-level parallelism, power consumption*

1. Introduction

The large speed gap between the processor and main memory severely degrades the performance of current processors. If a last-level cache (LLC) miss occurs, the processor stalls waiting for the requested data to be retrieved from main memory, the latency of which is hundreds of clock cycles. To mitigate against this, aggressive out-of-order execution is an effective approach [1]. In this approach, the instruction window, composed of the reorder buffer (ROB), the issue queue (IQ), and the load/store queue (LSQ), is extensively enlarged and instructions are aggressively reordered at issue time. This aggressive reordering allows an LLC-miss-causing load to be issued as early as possible, allowing multiple memory accesses to overlap. This effectively reduces the memory latency per missed load. This type of parallelism is called *memory-level parallelism* (MLP).

Although the basic idea of the aggressive out-of-order execution is simple, a straightforward implementation adversely affects the clock cycle time, because enlarging the instruction window resources causes an increase in the delay. This problem can be solved by pipelining the resources, but this in turn prevents instruction-level parallelism (ILP) from being exploited effectively. This is mainly due to pipelining the IQ, which prevents dependent instructions from being issued back-to-back.

To solve this problem, Kora *et al.* proposed *dynamic instruction window resizing* (DIWR), which appropriately configures the size of the instruction window to the available parallelism (MLP or ILP) [2], [3]. During a period where MLP is valuable to performance, the instruction window size is enlarged, whereas during a period where ILP is valuable, it is reduced. A large window allows MLP to be exploited effectively by aggressively reordering instructions, whereas a small window allows ILP to be exploited effectively with a single cycle operation of the window resources.

DIWR achieves a significant performance improvement, and good *energy* efficiency (i.e., performance per energy or inverse of energy-delay-product (EDP)). According to their evaluation, the energy efficiency is improved by 8%. However, *power* (energy per unit time) consumption increases owing to the enlarged instruction window resources. These large resources consume extensive dynamic and static power. Heat generated in a chip is due to the consumption of power. Figure 1 shows the evaluation results of power consumption of conventional (base) and DIWR processors, the configuration of which is given in Tables 1 and 3*. The vertical axis represents the power consumption of the base and DIWR processors relative to the average power consumption of the base processor. To save space, we show the results for only several selected programs from SPEC2006. “AVG mem”, “AVG comp”, and “AVG all” represent the average values for all memory-intensive, all compute-intensive, and all programs, including the non-selected programs, respectively. As shown in the figure, although the increase in power consumption of the DIWR processor is not so large in compute-intensive programs, it is considerable in memory-intensive programs. The power consumption of several memory-intensive programs even exceeds the largest power consumption in compute-intensive programs, despite the low activity of the processor due to LLC misses.

Manuscript received August 18, 2015.
Manuscript revised October 16, 2015.
Manuscript publicized November 12, 2015.

[†]The authors are with Department of Electrical Engineering and Computer Science, Nagoya University, Nagoya-shi, 464–8603 Japan.
a) E-mail: ando@nuee.nagoya-u.ac.jp
b) E-mail: shioya@nuee.nagoya-u.ac.jp

DOI: 10.1587/transinf.2015EDP7325

*In this evaluation, DVFS is not applied to either base and DIWR processors.

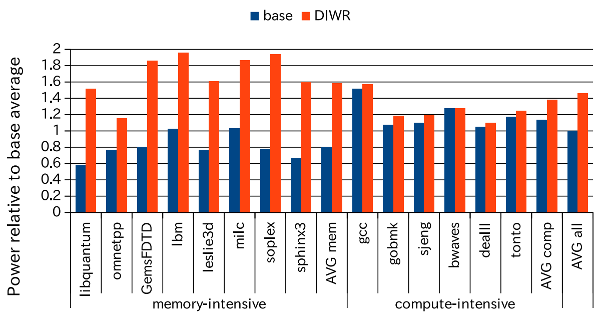


Fig. 1 Power consumption of conventional (base) and DIWR processors.

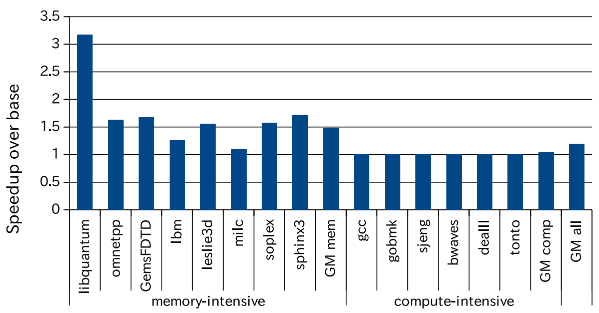


Fig. 2 Speedup of DIWR over the base without any power constraint.

If the power consumption exceeds a power budget given by certain requirements, the processor with DIWR must save power in a certain way. A common and widely introduced method for saving power is *dynamic voltage and frequency scaling* (DVFS). Although this is effective in power saving, it degrades performance.

In this paper we explore whether the DIWR processor can achieve better performance than a conventional processor with a limited power budget, and if so, what the extent of the improved performance is for a given power budget.

The remainder of this paper is organized as follows. Section 2 gives an overview of DIWR, while Sect. 3 defines the research question this study aims to answer. The evaluation results are presented in Sect. 4. Related work is described in Sect. 5, and our conclusions are given in Sect. 6.

2. Overview of Dynamic Instruction Window Resizing

This section overviews DIWR; for further details see [2], [3].

In this scheme, the Intel P6-type architecture [4] is assumed as the base architecture, where the instruction window resources are the ROB[†], IQ, and LSQ, all of which have a FIFO structure. Therefore, when at a particular time the region from the head to a particular entry is used, resizing is carried out by moving the boundaries of the used and unused regions. This resizing takes several cycles to configure the pipeline latches for the instruction window, and to enable or disable the circuits in the used or unused region, respectively. During this period, the processor pipeline stalls affecting performance slightly.

Values of the size and pipeline depth of the resource are referred to as the *instruction window resource level* (henceforth simply called the *level*) (i.e., *level* = {*size, pipeline-depth*}). As the level number increases, so the corresponding size also increases. Each resource with a particular size is pipelined so that it does not increase the clock cycle time.

The scheme predicts that once one LLC miss has occurred, further misses will occur continuously for a while, because LLC misses tend to be clustered with respect to time. Thus, the scheme enlarges the window resources once one LLC miss has occurred. In other words, the level of each window resource increases by one (if it is already at its maximum, it does not change).

[†]We assume that the processor has a map table that translates a logical register into a physical register field in the ROB to implement the ROB with a RAM organization in evaluation of Sect. 4.

In contrast, the scheme shrinks the window resources once the main memory latency has lapsed from the time at which the last LLC miss occurred. Specifically, the level of each window resource is reduced by one (if it is currently at level 1, the level does not change). Note that the shrinking of the window resources is delayed by stopping the resource allocation until the regions of the ROB, IQ, and LSQ that are to be removed by shrinking simultaneously become vacant.

DIWR significantly increases performance. Figure 2 shows the speedup over the base processor, with the configuration of the evaluated processors given in Tables 1 and 3. Again, we show the results for only several selected programs from SPEC2006. “GM mem”, “GM comp”, and “GM all” represent the geometric means for all memory-intensive, all compute-intensive, and all programs, including the non-selected programs, respectively.

As shown in the figure, in memory-intensive programs, the performance is significantly increased by exploiting MLP with a large instruction window. Although the instruction window is pipelined and thus, ILP is lost to a certain degree, the benefit of MLP exploitation significantly outweighs the loss of ILP. At the same time, in compute-intensive programs, the scheme exploits ILP with a small instruction window, where the window resources operate with a single cycle as in a conventional processor. Thus, a nearly identical amount of ILP is exploited.

3. Research Question

As shown in Fig. 2, DIWR achieves significant performance improvements. Also, as shown in [2], [3], energy efficiency is improved, mainly because execution time is significantly reduced. Unfortunately, power consumption increases owing to the enlarged instruction window. For example, the peak power consumption of the IQ at its maximum size (256 entries) (see Table 3) in the DIWR processor is 3.7 times larger than that in the base processor (64 entries), according to an evaluation using the McPAT [5].

The evaluation results for the power consumption of the base and DIWR processors are shown in Fig. 1. Although power consumption does not increase so much in compute-intensive programs, because the instruction window is infrequently enlarged, it does increase considerably in memory-intensive programs. The enlarged instruction window causes large dynamic and static power consumption in memory-intensive programs, despite the processor not being as busy owing to LLC misses.

Therefore, the question arises whether DIWR can really improve performance, even with a limited power budget. Here, the power budget is determined by several factors such as device heat and battery life requirements. This paper explores the performance of DIWR for a wide range of given power budgets, with DVFS introduced as a power saving technique.

4. Evaluation

We first describe the basic environment for evaluation, including the base processor configuration and the assumptions of the size and pipeline depth of the instruction window resources for each level in DIWR in Sect. 4.1. Then, we explain the assumptions of DVFS and the experimental methodology in Sect. 4.2 and 4.3, respectively. In Sect. 4.4, we evaluate the performance of DIWR for a given power budget. Section 4.5 discusses the results for each program. Finally, we show the evaluation results of energy efficiency in Sect. 4.6.

4.1 Basic Environment and Assumptions of Evaluation

The configuration of the base processor used in our evaluation is given in Table 1. For performance evaluation, we built a simulator based on the SimpleScalar Tool Set version 3.0a [6]. The instruction set used is Alpha ISA. We used all programs from the SPEC2006 except *wrf*, which does not currently run correctly on our simulator. The programs were compiled using gcc ver.4.5.3 with option -O3. In Table 2, the benchmark programs are categorized in terms of whether they are memory-intensive or compute-intensive with a threshold for the average load latency of 10 cycles. This category is used to show the following evaluation results. We simulated 100M instructions after the first 16G instructions were skipped with the *ref* inputs.

For the DIWR processor, we assume that it physically has window resources that are four times larger than those in the base processor, and that two levels can be selected with the DIWR scheme, as shown in Table 3. The pipeline depth of each resource is based on that in [2], [3]. In Table 3, we also include the assumption of the penalty at level transition [2], [3].

For power evaluation, we used the McPAT [5] modified for our assumptions in DVFS, assuming 22nm LSI technology.

4.2 DVFS Assumptions

We assume that the maximum supply voltage and clock frequency are 1.0V and 3.6GHz, respectively. As a scaling

Table 1 Configuration of base processor.

Pipeline width	four instructions wide for each of fetch, decode, issue, and commit
ROB	128 entries
Issue queue	64 entries
LSQ	64 entries
Branch prediction	16-bit history 64K-entry PHT gshare, 2K-set 4-way BTB, 10-cycle misprediction penalty
Function unit	4 iALU, 2 iMULT/DIV, 2 Ld/St, 4 fpALU, 2 fpMULT/DIV/SQRT
L1 I-cache	64KB, 2-way, 32B line
L1 D-cache	64KB, 2-way, 32B line, 2 ports, 2-cycle hit latency, non-blocking
L2 cache	2 MB, 4-way, 64B line, 12-cycle hit latency
MSHRs	32 for each of L1 and L2 caches
Main memory	300-cycle min. latency, 8B/cycle bandwidth
Data prefetcher	stride-based, 4K-entry, 4-way table, 16-data prefetch to L2 cache on miss

Table 2 Benchmark programs.

category	programs
memory-intensive	int: hmmer, libquantum, mcf, omnetpp, xalancbmk
	fp: GemsFDTD, lbm, leslie3d, milc, soplex, sphinx3
compute-intensive	int: astar, bzip2, gcc, gobmk, h264ref, perlbench, sjeng
	fp: bwaves, cactusADM, calculix, dealII, gamess, gromacs, namd, povray, tonto, zeusmp

Table 3 Number of entries and pipeline depths of window resources at each level and the assumption of a cycle penalty at level transition.

resource	parameter	level	
		1	2
IQ	entries	64	256
	pipeline depth	1	2
ROB	entries	128	512
	pipeline depth	1	2
LSQ	entries	64	256
	pipeline depth	1	2
Level transition penalty		10 cycles	

scenario for voltage and frequency, we introduce linear scaling as in several previous studies (e.g., [7], [8]), where the supply voltage is scaled linearly with the clock frequency. However, we assume that the supply voltage cannot be reduced below 0.65V (as assumed in [8]), which is the lowest supply voltage of the Intel Sandy Bridge [9]. That is, supply voltage V is scaled with clock frequency f according to the following equation:

$$V = \max\left(V_{min}, \frac{f}{f_{max}} V_{max}\right) \quad (1)$$

where f_{max}, V_{max}, and V_{min} are the maximum clock frequency (3.6GHz), maximum supply voltage (1.0V), and minimum supply voltage (0.65V), respectively.

Although an assumption based on the relationship in real products provides more realistic results, the latest reliable data available to us are those for the Intel Pentium M fabricated with 90nm LSI technology [10]. Because this is too old and does not reflect the current relationship, we chose linear scaling.

The DVFS transition rate is assumed to be 2.5V/μs.

Table 4 DVFS parameters.

parameter	value
clock frequency (GHz)	1.6 – 3.6
supply voltage (V)	0.65 – 1.00, scaled according to Eq. (1)
transition rate (V/μs)	2.5
MPKI threshold	0.25 – 4.0

This rate is based on the performance of the integrated voltage regulator presented recently, which is implemented in the Intel Haswell [11]. According to Intel, the transition of the supply voltage from 0.8V to 1.05V for the turbo boost takes 100ns. Although the turbo boost and DVFS have an opposite direction (higher performance and larger power consumption vs. lower performance and smaller power consumption), they use the same technology. Therefore, we used this rate in our experiment. Although this rate is approximately 100 times higher than the conventional off-chip voltage regulator, we believe that many processor vendors will introduce this type of voltage regulator in the near future. We assume that the processor stalls during the transition period.

The assumptions described above are those obtained from our best efforts to adapt our evaluation to recent real processors like Intel's Sandy Bridge and Haswell.

We assume that the DVFS is applied only to the processor. Therefore, the main memory latency and the burst rate in the cycle is scaled appropriately according to the scaled clock frequency in our cycle-based performance simulation.

Table 4 summarizes the DVFS parameters and their values in our experiment.

4.3 Experimental Methodology

In our experiment, we explored the best voltage/frequency to maximize performance for a given power budget independently in memory- and compute-intensive phases for both base and DIWR processors, as shown in Fig. 3. Henceforth, we denote the voltage/frequency in memory- and compute-intensive phases as V_{mem}/f_{mem} and V_{comp}/f_{comp}, respectively. We measure L2 misses per kilo instructions (MPKI) on an interval basis to determine memory intensity. If this value is greater than a predetermined threshold in an interval, we predict that the next interval will be memory-intensive; otherwise we predict that it will be compute-intensive. We determined the interval to be 10,000 cycles and varied MPKI threshold from 0.25 to 4.0 in our experiment as indicated in Table 4.

The reason that we consider memory intensity in the DVFS control is that an L2 miss causes a long (300 cycles or more) low activity phase, and the only event that causes such a phase is an L2 miss in our experiment. Decreasing the clock frequency during the memory-intensive phase does not degrade performance significantly; however, considerable power can be saved by reducing the voltage/frequency. In contrast, the clock frequency strongly affects performance in compute-intensive phases.

The following two steps are carried out in the experiment:

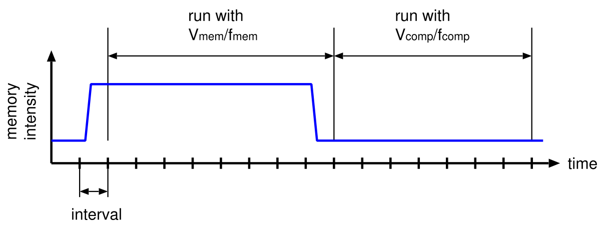


Fig. 3 DVFS control according to memory intensity.

- **Step 1:** Through simulations, we evaluate performance (in terms of throughput, which is committed instructions per second) and power consumption for every combination of DVFS parameters, i.e., V_{mem}/f_{mem}, V_{comp}/f_{comp}, and MPKI threshold, with varying their values.
- **Step 2:** We attempt to find the best DVFS parameter value combination that maximizes throughput for a given power budget by searching the simulation results.

The procedure described above quasi-ideally finds the maximum performance strictly satisfying a given power budget. We do not introduce real DVFS control on commercial processors, because it is based only on the present activity, and thus does not ensure satisfying a given power budget. Unless the DVFS controller knows the future activity of the processor, it does not know how much it should shift the voltage/frequency at present to satisfy the given power budget and simultaneously maximize performance.

The specific experimental procedure is outlined in Fig. 4 using pseudo code, where lines 1 to 19 represent step 1, and lines 21 to 52 represent step 2.

In step 1, for each DVFS parameter (lines 9 to 11) (see Table 7 for voltage/frequency combination we explored), we carry out a performance and power simulation for each program, and output the throughput and power consumption results (lines 13 to 15). Note that the DVFS parameters (i.e., V_{mem}/f_{mem}, V_{comp}/f_{comp}, and MPKI threshold) are not changed during a single simulation in lines 13 and 14. Then, in step 2, for each given power budget (line 27), we attempt to find the best throughput for each program in the following way: For each DVFS parameter (lines 32 to 34), we check whether the power consumption satisfies the power budget (line 36). If it does, and if throughput under the current DVFS parameters is greater than the best throughput previously found, the best throughput and power consumption combination is updated (lines 39 and 40). If there is at least one set of DVFS parameters that satisfies the power budget, the procedure outputs the best throughput and power consumption combination (line 47). Otherwise, it outputs "NG" (line 49) (note that only meaningful results (i.e., not "NG" outputs) were obtained in the evaluation in the following sections).

Here, we define two power budgets. First, we determine the *full power budget* to be the maximum power con-

```
# Step 1: Simulations
f_comp: scaled clock frequency in compute-intensive interval;
f_mem:  scaled clock frequency in memory-intensive interval;
V_comp: scaled supply voltage in compute-intensive interval;
V_mem:  scaled supply voltage in memory-intensive interval;
MPKI_threshold: MPKI threshold to determine memory intensity;
MPKI_threshold_list: MPKI thresholds explored (0.25, 0.5, 1, 2, 4);

foreach (f_comp, V_comp) in DVFS Table 7 {
  foreach (f_mem, V_mem) in DVFS Table 7 {
    foreach MPKI_threshold in MPKI_threshold_list {
      foreach program in SPEC2006_suite {
        carry out performance simulation and obtain resource usage statistics and throughput;
        carry out power simulation using resource usage statistics and obtain power;
        output "program,throughput,power,f_comp,V_comp,f_mem,V_mem,MPKI_threshold";
      }
    }
  }
}

# Step 2: Find best throughput for a given power budget
budget_full: maximum power consumption in DIWR processor;
budget_min:  minimum budget rate explored (0.3);
budget_max:  maximum budget rate explored (1.0);
budget_step: change step of budget rate (0.02);

foreach budget_rate in (budget_min..budget_max) with steps of budget_step {
  budget = budget_full * budget_rate;
  throughput_best = -1;
  budget_satisfied = 0;
  foreach program in SPEC2006_suite {
    foreach (f_comp, V_comp) in DVFS Table 7 {
      foreach (f_mem, V_mem) in DVFS Table 7 {
        foreach MPKI_threshold in MPKI_threshold_list {
          curr_parameters = (f_comp, V_comp, f_mem, V_mem, MPKI_threshold);
          if (power under curr_parameters <= budget) {
            budget_satisfied = 1;
            if (throughput under curr_parameters > throughput_best) {
              throughput_best = throughput under curr_parameters;
              power_at_best = power under curr_parameters;
            }
          }
        }
      }
    }
    if (budget_satisfied) {
      output "budget_rate,program,throughput_best,power_at_best";
    } else {
      output "budget_rate,program,NG";
    }
  }
}
```

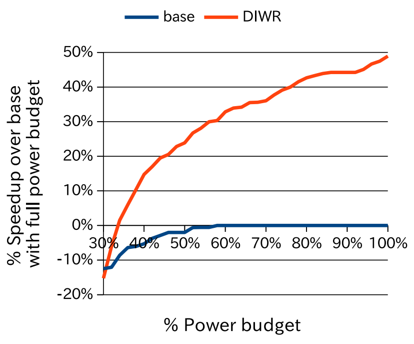
Fig. 4 Experimental procedure to obtain the best throughput for a given power budget.

sumption of the DIWR processor without power constraints during the execution of all programs. Specifically, we let the full power budget be the power consumption in *lbm* (see Fig. 1). Next, we call the maximum power consumption of the base processor without any power constraints during the execution of all programs, the *base power budget*, which is the power consumption in *gcc* (see Fig. 1), and is 77% of the full power budget. The evaluation result at the base power budget is particularly important, because the the thermal design power (TDP) of the base processor should be at least this number.

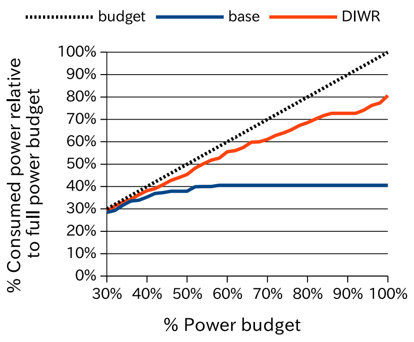
4.4 Performance Evaluation Results

Figures 5 and 6 show performance (in terms of throughput as defined above) and power consumption in memory-intensive and compute-intensive programs, respectively. Each figure includes two graphs: (a) the percentage average speedup of the base and DIWR processors over the base processor with the full power budget; and (b) the percentage average consumed power of the base and DIWR processors relative to the full power budget.

As shown in Fig. 5(a), performance steadily decreases as the power budget decreases in the DIWR processor, because the power consumption in the execution of several programs reaches the limit of the power budget, and DVFS needs to lower the voltage/frequency. This is seen in Fig. 5(b), where the power consumption steadily decreases as the power budget decreases. In the base processor performance barely deteriorates down to 50% of the full power budget, because the maximum consumed power is the 77% budget as mentioned previously, and the power consumption in few programs reaches the power budget in this range. Below the 50% budget, performance does deteriorate, but the rate is slower compared with that for the DIWR processor. Therefore, the performance of DIWR relative to the base is

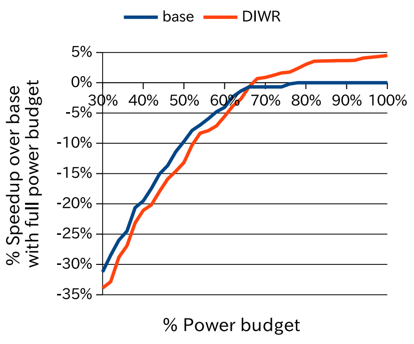


(a) Speedup over the base with full power budget

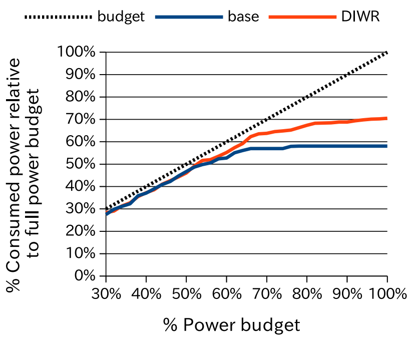


(b) Consumed power relative to maximum power budget

Fig. 5 Throughput and consumed power in *memory-intensive* programs.



(a) Speedup over the base with full power budget



(b) Consumed power relative to maximum power budget

Fig. 6 Throughput and consumed power in *compute-intensive* programs.

Table 5 Summary of speedup of DIWR over the base.

program category	power budget		
	full	base	minimum (30%)
memory-intensive	49%	41%	−3%
compute-intensive	4%	2%	−4%
all	20%	16%	−1%

always decreased as the power budget decreases. Not surprisingly, the maximum speedup of DIWR over the base (49%) is achieved at the full power budget, while the minimum speedup in the experimental range is −3% at the 30% power budget. At the base power budget, which is the most important point, the speedup is still as much as 41%.

In compute-intensive programs, the DIWR and base processors exhibit similar curves for all power budgets, as shown in Fig. 6(a). The power consumption of DIWR is only slightly larger than that of the base, as shown in Fig. 6(b). These results are due to the fact that the DIWR scheme adaptively keeps the window size small to exploit the ILP included in compute-intensive programs.

Table 5 summarizes the evaluated speedup.

4.5 Discussion of Results for Each Program

Figure 7 shows the speedup of DIWR over the base for various power budgets (denoted PB in the figure legend) for each benchmark program. Table 6 lists the DVFS table indices corresponding to the selected voltage/frequency for each given power budget in each program. See Table 7 for the actual values for each DVFS table index. The index labels for C and M represent those for the voltage/frequency of compute- and memory-intensive intervals, respectively (i.e., V_{comp}/f_{comp} and V_{mem}/f_{mem}). For each index pair in the table, the left and right indices represent those for the base and DIWR processors, respectively.

As described in Sect. 4.4, the speedup of DIWR is significantly reduced as the power budget decreases in memory-intensive programs based on geometric mean. Such sensitivity to the power budget is unexceptionally found in all memory-intensive programs, as shown in Fig. 7. As seen in the DVFS table indices for memory-intensive programs given in Table 6(a), the frequency of both f_{comp} and f_{mem} is reduced more in DIWR than in the base for small power budgets (the indices in such the case are highlighted by boldface). This means that the instruction window cannot be enlarged without reducing the voltage/frequency in DIWR for a small power budget. However, as seen in Fig. 7, the speedup over the base is still significant over a wide range of our experimental power budgets in most memory-intensive programs.

In contrast, speedup sensitivity to the power budget is very small in all compute-intensive programs as shown in Fig. 7, and DIWR exhibits a similar performance to that of the base. Although the power consumption does reach the

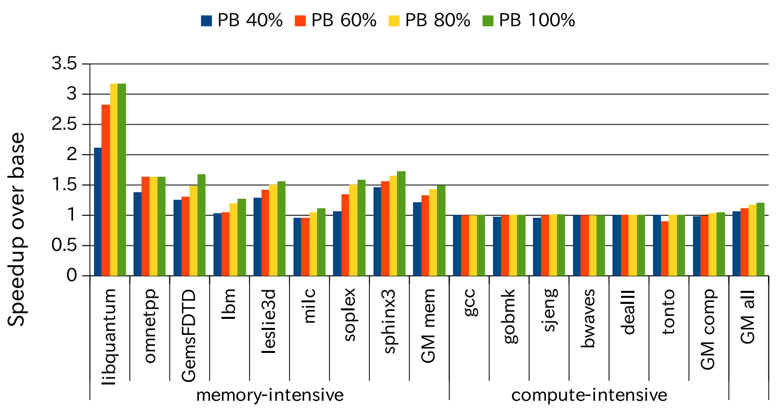


Fig. 7 Speedup over base for various power budgets.

Table 6 Selected DVFS table indices for various power budgets. Left and right indices represent those for base and DIWR, respectively.

program	power budget							
	40%		60%		80%		100%	
	C	M	C	M	C	M	C	M
libquantum	0, 0	0, **3**	0, 0	0, **1**	0, 0	0, 0	0, 0	0, 0
omnetpp	0, **1**	0, **3**	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
GemsFDTD	0, **3**	1, **3**	0, **2**	0, **2**	0, **1**	0, **1**	0, 0	0, 0
lbm	0, 0	2, **3**	0, **1**	0, **2**	0, 0	0, **1**	0, 0	0, **1**
leslie3d	0, 0	0, **3**	0, 0	0, **2**	0, 0	0, **1**	0, 0	0, 0
milc	0, **1**	2, **3**	0, 0	0, **2**	0, 0	0, **1**	0, 0	0, **3**
soplex	0, **2**	0, **4**	0, 0	0, **3**	0, 0	0, **1**	0, 0	0, 0
sphinx3	0, 0	0, **3**	0, 0	0, **2**	0, 0	0, **1**	0, 0	0, 0

(a) Memory-intensive programs

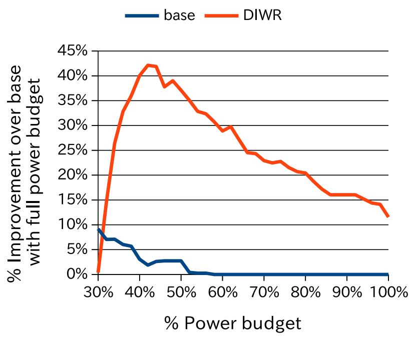
program	power budget							
	40%		60%		80%		100%	
	C	M	C	M	C	M	C	M
gcc	3, 3	3, 3	1, 1	1, **2**	0, 0	0, **1**	0, 0	0, 0
gobmk	1, **2**	3, 2	0, 0	0, **1**	0, 0	0, 0	0, 0	0, 0
sjeng	1, **2**	2, 2	0, 0	0, **1**	0, 0	0, 0	0, 0	0, 0
bwaves	2, 2	2, 2	1, 1	1, 1	0, 0	0, 0	0, 0	0, 0
dealII	2, 2	2, 2	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
tonto	2, 2	2, 2	0, **1**	0, **1**	0, 0	0, 0	0, 0	0, 0

(b) Compute-intensive programs

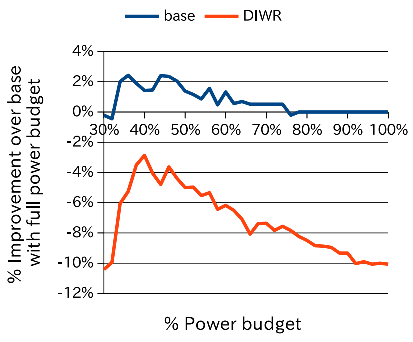
Table 7 DVFS table of supply voltage and clock frequency.

index	frequency (GHz)	voltage (V)
0	3.6	1.000
1	3.2	0.889
2	2.8	0.778
3	2.4	0.667
4	2.0	0.650
5	1.6	0.650

limit of the power budget with small power budgets, as seen in Table 6(b), the DVFS table indices for the base and DIWR processors are mostly the same for all power budgets and all programs. However, in a few programs, the performance is slightly lower than that of the base processor in small power budget cases. This arises from a lower clock frequency being selected (see boldface index numbers in Table 6(b)). Since L2 misses occur even in compute-intensive programs and DIWR increases the level with responses to the misses, although it is infrequent. This increases the power consumption, and thus a lower clock frequency is selected. As shown



(a) Memory-intensive programs



(b) Compute-intensive programs

Fig. 8 Energy efficiency relative to that of the base processor with a full power budget.

in Fig. 6, the average performance of DIWR is slightly lower than that of the base in small power budget cases with this reason.

Finally, note that owing to a noise level difference, M indices in *lbm* and *milc* are not 0 at 100% power budget. The differences in throughput compared with the case of an index of 0 are only 0.004% and 0.2%. A noise level difference generally arises because of constant memory latency in time.

4.6 Energy Efficiency

Figure 8 shows the energy efficiency, i.e., the performance per unit of energy (which is proportional to 1/EDP (energy-delay product)). The y-axis represents the percentage improvement in energy efficiency of the base and DIWR processors over the base processor with a full power budget.

As shown in Fig. 8(a), DIWR improves energy efficiency for almost all the given power budgets over the base in memory-intensive programs. The maximum improvement is 40% at the 42% power budget. In contrast, DIWR exhibits lower energy efficiency compared with the base in compute-intensive programs, as shown in Fig. 8(b). However, the degradation, 10% at maximum, is acceptable. The improvement in the evaluation of all the programs ranges from a minimum of −9% to a maximum of 10%. At the base power budget point, the improvement is 4%.

5. Related Work

In this section, we first discuss studies on MLP, focusing on those that consider power. We then review research on reducing the power consumption of instruction window resources.

5.1 Studies Considering Power on MLP Exploitation

Many studies on MLP have been carried out. In general, to exploit MLP, distant miss loads in the program order must be brought closer in terms of time in some way. This extra work requires additional energy. However, to the best of our knowledge, few studies have been carried out focusing on power consumption.

Early studies on MLP extract the instructions necessary for the generation of memory accesses as a thread, and then spawn the thread at a certain point in the program execution to a different context of the processor [12]–[19]. Among these studies, those considering power consumption are, to the best of our knowledge, [16] and [19]. In [16], Collins *et al.* focus on *delinquent loads*, which cause frequent LLC misses, and extract pre-execution instructions related only to these. In [19], Petric *et al.* estimate performance benefit and energy consumption analytically, and then select good threads in terms of the performance/energy tradeoff.

The disadvantage of these thread-based schemes, including those of Petric and Collins, is that they require a multithreaded environment such as simultaneous multithreading or chip multiprocessors, and the context is consumed. In other words, the opportunity to allocate other independent threads to the context for better throughput is lost.

However, several studies that do not need a multithreaded environment have been carried out more recently [20]–[24]. These studies commonly pre-execute instructions when the instruction window resources run out. A typical cause of running out of instruction window resources is an LLC miss. Pre-execution is carried out when an LLC miss occurs, and other LLC misses can be overlapped with the original LLC miss, if misses occur during pre-execution.

Among these studies, those considering power consumption are [25] and [26]. In [25], Hyodo *et al.*, like [16], focus on delinquent loads, and pre-execute only instructions related to these. In [26], Mutlu *et al.* proposed several techniques for reducing the number of pre-executed instructions. Since these techniques are tailored to pre-execution or their base technique for MLP exploitation, they are not applicable to DIWR.

5.2 Research on Reducing Power Consumption of Instruction Window Resources

Since instruction window resources consume large amounts of power, a number of studies have focused on saving power.

Issue queue: Ponomarev *et al.* and Folegnani *et al.* proposed resizing the IQ based on demand [27], [28]. If the demand is high, the IQ is enlarged; otherwise it is reduced, reducing power consumption. Demand is measured by occupation of the IQ or contribution to performance. DIWR differs from these studies, because the resizing policy is MLP-aware.

Michaud *et al.* proposed prescheduling instructions before inserting instructions into the IQ [29]. In their study, the IQ comprises a shift register and small conventional CAM-type IQ. The scheme predicts issue timing of an instruction and places the instruction into the shift register according to the prediction. Replacing a part of the CAM comprising the conventional IQ with a simple shift register reduces power consumption. The downside of this scheme is that the small CAM part is easily clogged by an LLC miss load, and consequently issuing instructions is completely blocked. Thus, this scheme is not suitable for exploiting MLP.

Load/store queue: A number of studies to reduce the complexity and consequently, also the power consumption of the LSQ have been carried out (e.g., [30], [31]). Many of these attempt to eliminate the CAM in the LSQ circuit. These schemes are orthogonal to DIWR, and can deliver improved performance for a given power budget.

Cain *et al.* proposed *value-based replay* that re-executes loads in program order to check whether speculatively executed loads preserved or violated dependence [30]. This scheme eliminates the associative search function of the load queue, although re-execution of loads consumes additional power.

Sha *et al.* proposed a scheme that predicts the store queue index for store-load forwarding [31]. This scheme eliminates the associative search function of the store queue.

6. Conclusions

A previous study showed that DIWR achieves significant performance improvement with a very modest change in the existing processor architecture and very small additional cost. Unfortunately, power consumption (energy per unit time) is considerably increased. The question that arose is whether DIWR can improve performance with a limited given power budget. In this paper, we explored the performance of both DIWR and conventional processors with a given power budget, incorporating power consumption saving using DVFS quasi-ideally. Our evaluation results show that DIWR still delivers better performance than the conventional processor over a wide range of given power budgets. At the most important power budget point, i.e., the power a conventional processor consumes without power constraints, DIWR achieves a 16% speedup.

References

[1] A. Cristal, O.J. Santana, F. Cazorla, M. Galluzzi, T. Ramírez, M. Pericàs, and M. Valero, "Kilo-instruction processors: Overcoming the memory wall," IEEE Micro, vol.25, no.3, pp.48–57, May/June 2005.

[2] Y. Kora, K. Yamaguchi, and H. Ando, "MLP-aware dynamic instruction window resizing for adaptively exploiting both ILP and MLP," Proceedings of the 46th Annual International Symposium on Microarchitecture, pp.37–48, Dec. 2013.

[3] Y. Kora, K. Yamaguchi, and H. Ando, "MLP-aware dynamic instruction window resizing in superscalar processors for adaptively exploiting available parallelism," IEICE Trans. Inf. & Syst., vol.E97-D, no.12, pp.3110–3123, Dec. 2014.

[4] Intel, P6 Family of Processors - Hardware Developer's Manual, Sept. 1998.

[5] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," Proceedings of the 42nd Annual International Symposium on Microarchitecture, pp.469–480, Dec. 2009.

[6] http://www.simplescalar.com/

[7] C. Isci, A. Buyuktosunoglu, C.Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," Proceedings of the 39th Annual International Symposium on Microarchitecture, pp.347–358, Dec. 2006.

[8] R. Miftakhutdinov, E. Ebrahimi, and Y.N. Patt, "Predicting performance impact of DVFS for realistic memory systems," Proceedings of the 45th Annual International Symposium on Microarchitecture, pp.155–165, Dec. 2012.

[9] D. Kanter, "Sandy bridge ISSCC update," Real World Technologies, April 2011. http://www.realworldtech.com/sandy-bridge-circuits/

[10] Intel Corporation, "Intel Pentium M processor on 90nm process with 2-MB L2 cache datasheet," Jan. 2006.

[11] N. Kurd, M. Chowdhury, E. Burton, T.P. Thomas, C. Mozak, B. Boswell, M. Lal, A. Deval, J. Douglas, M. Elassal, A. Nalamalpu, T.M. Wilson, M. Merten, S. Chennupaty, W. Gomes, and R. Kumar, "Haswell: A family of IA 22nm processors," 2014 International Solid-State Circuits Conference Digest of Technical Papers, pp.112–113, Feb. 2014.

[12] R. Chappell, J. Stark, S. Kim, S. Reinhardt, and Y. Patt, "Simultaneous subordinate microthreading (SSMT)," Proceedings of the 26th Annual International Symposium on Computer Architecture, pp.186–195, May 1999.

[13] C. Zilles and G.S. Sohi, "Master/slave speculative parallelization," Proceedings of the 35th Annual International Symposium on Microarchitecture, pp.85–96, Nov. 2002.

[14] Z. Purser, K. Sundaramoorthy, and E. Rotenberg, "A study of slipstream processors," Proceedings of the 33rd Annual International Symposium on Microarchitecture, pp.269–280, Dec. 2000.

[15] A. Roth and G.S. Sohi, "Speculative data-driven multithreading," Proceeding of the Seventh Annual International Symposium on High Performance Computer Architecture, pp.37–48, Jan. 2001.

[16] J.D. Collins, D.M. Tullsen, H. Wang, Y. Lee, D. Lavery, J.P. Shen, and C. Hughes, "Speculative precomputation: Long-range prefetching of delinquent loads," Proceedings of the 28th Annual International Symposium on Computer Architecture, pp.14–25, July 2001.

[17] J.D. Collins, D.M. Tullsen, H. Wang, and J.P. Shen, "Dynamic speculative precomputation," Proceedings of the 34th Annual International Symposium on Microarchitecture, pp.306–317, Dec. 2001.

[18] C.B. Zilles and G.S. Sohi, "Execution-based prediction using speculative slices," Proceedings of the 28th Annual International Symposium on Computer Architecture, pp.2–13, July 2001.

[19] V. Petric and A. Roth, "Energy-effectiveness of pre-execution and energy-aware p-thread selection," Proceedings of the 32nd Annual International Symposium on Computer Architecture, pp.322–333, June 2005.

[20] A.R. Lebeck, J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg, "A large, fast instruction window for tolerating cache misses," Proceedings of the 29th Annual International Symposium on Computer Architecture, pp.59–70, May 2002.

[21] O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt, "Runahead execution: An effective alternative to large instruction windows," Proceedings of the Nineth Annual International Symposium on High-Performance Computer Architecture, pp.129–140, Feb. 2003.

[22] S.T. Srinivasan, R. Rajwar, H. Akkary, A. Gandhi, and M. Upton, "Continual flow pipelines," Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.107–119, Oct. 2004.

[23] A. Yamamoto, Y. Tanaka, H. Ando, and T. Shimada, "Data prefetching and address pre-calculation through instruction pre-execution with two-step physical register deallocation," Proceedings of the Eighth Workshop on Memory Performance: Dealing with Applications, Systems and Architectures, pp.33–40, Sept. 2007.

[24] A. Yamamoto, Y. Tanaka, H. Ando, and T. Shimada, "Two-step physical register deallocation for data prefetching and address pre-calculation," IPSJ Transactions on Advanced Computing Systems, vol.1, no.2, pp.94–106, Aug. 2008.

[25] K. Hyodo, K. Iwamoto, and H. Ando, "Energy-efficient pre-execution techniques in two-step physical register deallocation," IEICE Trans. Inf. & Syst., vol.E92-D, no.11, pp.2186–2195, Nov. 2009.

[26] O. Mutlu, H. Kim, and Y.N. Patt, "Techniques for efficient processing in runahead execution engines," Proceedings of the 32nd International Symposium on Computer Architecture, pp.370–381, June 2005.

[27] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," Proceedings of the 34th Annual International Symposium on Microarchitecture, pp.90–101, Dec. 2001.

[28] D. Folegnani and A. González, "Energy-effective issue logic," Proceedings of the 28th Annual International Symposium on Computer Architecture, pp.230–239, June 2001.

[29] P. Michaud and A. Seznec, "Data-flow prescheduling for large instruction windows in out-of-order processors," Proceedings of the Seventh International Symposium on High-Performance Computer Architecture, pp.27–36, Jan. 2001.

[30] H.W. Cain and M.H. Lipasti, "Memory ordering: A value-based approach," Proceedings of the 31st Annual International Symposium on Computer Architecture, pp.90–101, June 2004.

[31] T. Sha, M.M.K. Martin, and A. Roth, "Scalable store-load forwarding via store queue index prediction," Proceedings of the 38th Annual International Symposium on Microarchitecture, pp.159–170, Nov. 2005.

Hideki Ando received his B.S. and M.S. degrees in Electronic Engineering from Osaka University, Suita, Japan, in 1981 and 1983, respectively. He received his Ph.D. degree in Information Science from Kyoto University, Kyoto, Japan, in 1996. From 1983 to 1997 he was with Mitsubishi Electric Corporation, Itami, Japan. From 1991 to 1992 he was a visiting scholar at Stanford University. In 1997 he joined the faculty of Nagoya University, Nagoya, Japan, where he is currently a Professor in the Department of Electrical Engineering and Computer Science. He received IPSJ best paper awards in 1998 and 2002, and a best paper award at the Symposium on Advanced Computing Systems and Infrastructures in 2013. His research interests include computer architecture and compilers.

Ryota Shioya was born in 1981. He received his M.E. and Ph.D. in Information and Communication Engineering from the University of Tokyo in 2008 and 2011, respectively. He was a research fellow of the Japan Society for the Promotion of Science from 2009 to 2011. Since 2011, he is an assistant professor at the Graduate School of Engineering, Nagoya University. He is a member of IEICE, IPSJ, and IEEE.