Integrated CPU Cache Power Management in Multiple Clock Domain Processors

Nevine AbouGhazaleh, Bruce Childers, Daniel Mossé and Rami Melhem
{nevine, childers, mosse, melhem}@cs.pitt.edu

Department of Computer Science, University of Pittsburgh

Abstract. Multiple clock domain (MCD) chip design addresses the problem of increasing clock skew in different chip units. Importantly, MCD design offers an opportunity for fine grain power/energy management of the components in each clock domain with dynamic voltage scaling (DVS). In this paper, we propose and evaluate a novel integrated DVS approach to synergistically manage the energy of chip components in different clock domains. We focus on embedded processors where core and L2 cache domains are the major energy consumers. We propose a policy that adapts clock speed and voltage in both domains based on each domain's workload and the workload experienced by the other domain. In our approach, the DVS policy detects and accounts for the effect of inter-domain interactions. Based on the interaction between the two domains, we select an appropriate clock speed and voltage that optimizes the energy of the entire chip. For the Mibench benchmarks, our policy achieves an average improvement over no-power-management of 15.5% in energy-delay product and 19% in energy savings. In comparison to a traditional DVS policy for MCD design that manages domains independently, our policy achieves an 3.5% average improvement in energy-delay and 4% less energy, with a negligible 1% decrease in performance. We also show that an integrated DVS policy for MCD design with two domains is more energy efficient for simple embedded processors than high-end ones.

1 Introduction

With the increase in number of transistors and reduced feature size, higher chip densities create a problem for clock synchronization among chip computational units. With a single master clock for the entire chip, it has become difficult to design a clock distribution network that limits clock skew among the chip components. Several solutions have been proposed to this problem using globally-asynchronous locally synchronous (GALS) design. In GALS design, a chip is divided into multiple clock domains (MCD), where individual chip units are associated with a particular domain. Each domain operates synchronously with its own clock and communicates with other domains asynchronously through queues.

In addition to addressing clock skew, MCD design offers important benefits to reducing power consumption with dynamic voltage scaling (DVS) at the domain level. Such fine-grain power management is important for embedded systems, which often have especially tight constraints on power/energy requirements. Indeed, National

Semiconductor has recently developed a technology, called PowerWise, that uses multiple domains to manage the power consumption of ARM-based system-on-a-chip designs [1]. Since each domain maintains its own clock and voltage independently of other domains, DVS can be applied at the domain level, rather than at the chip level. Power and energy consumption can be reduced by dynamically adjusting an individual domain's clock and voltage according to domain activity. Throughout this paper we use the term *speed* to collectively refer to voltage and frequency.

Several power management policies have been proposed to incorporate DVS into MCD chips. For example, Magklis et al.'s seminal online power management policy [2] monitors queue occupancy of a domain and computes the change in the average queue length in consecutive intervals. When queue length increases, the domain speed is increased; when queue length decreases, the speed is decreased. In general, policies in the literature [3][4][5][6] focus on each domain in *isolation* without considering possible inter-domain effects when varying speed.

In this paper, we propose an **integrated** power management policy for embedded processors with multiple clock domains. Unlike other techniques, our policy takes into account activity and workload in all domains to decide the best set of speed settings. Our policy stems from our observation that current online DVS policies for MCD chips have a localized view and control of the DVS in each domain and do not account for domain interactions. For the Mibench and the SPEC2000 benchmarks, our policy improves the energy-delay product by 15.5% and 18.5% on average (up to 26%) while energy savings are 19% and 23.5% on average (up to 32%). The performance penalty is less than 5% and 6.5%, respectively. Compared to a well-known online MCD DVS policy [3], we show an additional improvement in the energy-delay product of 3.5% and 7%, on average (up to 13%), with minimal performance degradation. Our policy requires no additional hardware beyond what is already available in MCD design.

The contribution of this paper is threefold. First, we identify a significant inefficiency in current online DVS policies, and show the sources and implications of this inefficiency. Second, we propose a new DVS policy that adapts the core and L2 cache speeds in a way that avoids these inefficiencies, taking into account domain interactions. Third, we show positive gains of our policy against a well-known online DVS policy [3].

The remaining portion of this paper is organized as follows. As background, we first describe application characteristics and MCD hardware design in Section 2. Section 3 compares independent and integrated DVS policies for MCD in terms of design and implementation. Section 4 presents our integrated DVS policy and identifies scenarios where it performs better than an independent DVS policy. Evaluation and sensitivity analysis of our policy against a well-known DVS policy is presented in Section 5. Other related work is presented in Section 6 and concluding remarks are in Section 7.

2 Application and MCD Chip Models

In this paper, because of the focus on embedded systems, we first consider a simple MCD processor with two domains (see Figure 1), namely the *core* and the *L2 cache*, and later expand it to include processors with more domains. We consider the core and

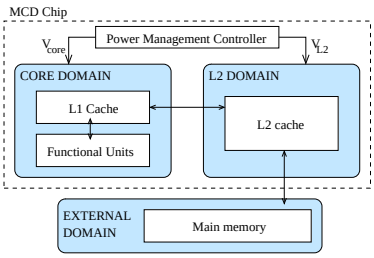


Fig. 1. MCD processor with two domains

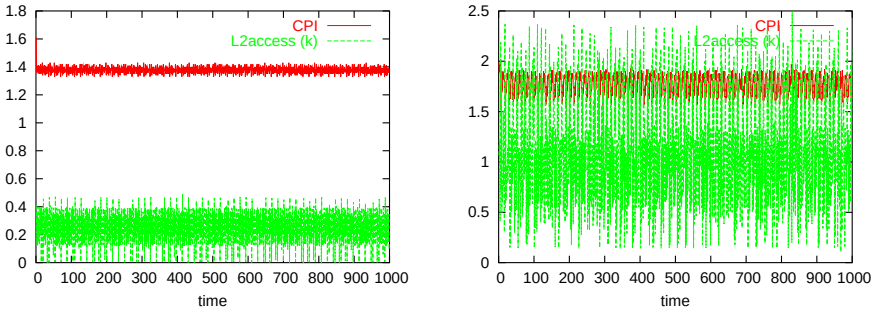
the L2 cache domains due to their high influence on the overall performance and energy consumption. The core domain includes all computational units such as the register file, functional units, issue unit, decode unit and L1 cache. In the core domain, each individual unit consumes a small fraction of the total power, but when grouped, that domain consumes a large fraction of the total chip power. On the other hand, caches consume a large fraction of the total chip power. For example, caches consume 50% power for ARM10TDMI running at 400MHz [7] . Moreover, it is predicted that the L2 cache will continue to be one of the major consumers of energy (due to increasing on-chip L2 cache sizes) [8].

A typical application goes through phases during its execution. An application has varying cache/memory access patterns and CPU stall patterns. In general, application phases correspond to loops, and a new phase is entered when control branches to a different code section. Since we are interested in the performance and energy of the CPU core and L2 cache, we characterize each code segment in a program using performance monitors that relate to the activity in each of these domains [3]. Figure 2 shows the variations in two performance counters (cycle-per-instruction and number of L2 accesses) as examples of monitors that can be used to represent a program behavior. We obtain these traces from running the shown benchmarks on Simplescalar with a StrongArm-like processor configuration (see Section 5). From these graphs, applications go through varying phases, which cause varying activity in different chip domains.

3 DVS in Multiple Clock Domains

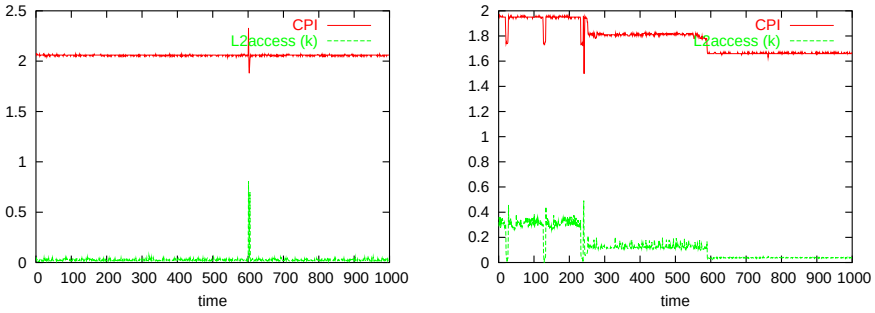
As briefly mentioned above, there are two categories of DVS policies for MCD processors that can be implemented in hardware. We discuss them in the context of a two-domain MCD design shown in Figure 1.

The first is called *Independent DVS policy*. This policy periodically sets the speed of each domain independently based on the activity of the domain, which is measured through performance counters in that domain. For example, we may use the number of instructions-per-cycle (IPC) and the number of L2 cache accesses as an indication of the activity in the core and L2 cache domains. IPC encompasses the effects of several factors affecting performance that occur within the core such as number and type (INT/FP) of issued instructions, branch mispredictions, L1 and TLB accesses. Higher



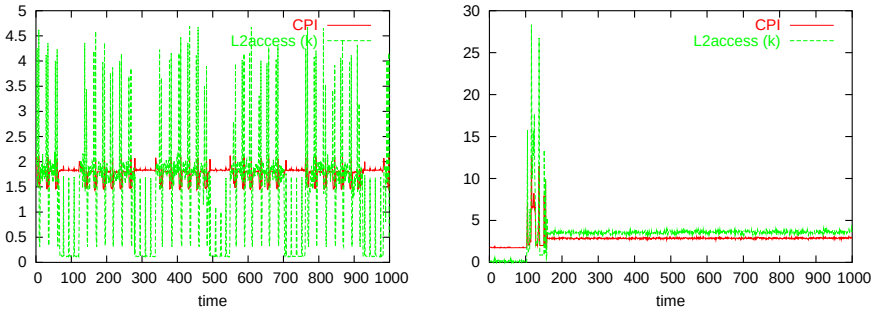
(a) *gsm-toast* phases

(b) *lame* phases



(c) *rsynth* phases

(d) *bzip* phases



(e) *gzip* phases

(f) *twolf* phases

Fig. 2. Variations in Cycle-per-Instruction (CPI) and L2 accesses (L2access) of six Mibench and SPEC2000 benchmarks.

(lower) IPC indicates that more (less) instructions finished execution and the presence of fewer (more) stall cycles in the different core units. Similarly, in the L2 cache domain higher (lower) L2 requests indicate higher (lower) activity in the cache's different sections. The policy periodically monitors the variations in IPC and L2 accesses across interval periods. The IPC and number of L2 accesses can be monitored through commonly available event counters in most modern architectures. Based on the trend in a counter, the policy decides to change the speed of the corresponding domain. This scheme is efficient because it can be done locally and with very little overhead.

The second is the *Integrated DVS policy*, which takes into account the effect of speed changes of one domain on the other domain. For example, reducing speed in a non-critical domain may result in an indirect slow-down in the performance-critical domain. This slowdown is not necessarily due to a change in application behavior, but rather a reaction to the other domain's slowdown. Detailed discussion of different domain interactions is described in Section 4.1. Domain interaction is the driving force behind our approach.

Our goal is to design an integrated core- L2 cache DVS policy that (1) selects appropriate speeds for each domain, adapting to application's run-time behavior (phases) and (2) minimizes the overall energy-delay product[1].

The general idea of our integrated MCD DVS approach is the use of a power management controller that collects information about the workload of *all* domains, and sets the speed of each appropriately. The power management controller uses the combined behavior in all domains in a given interval to decide the speed of each domain in the following interval. Details of our proposed policy are presented in Section 4.2.

4 Domain interaction-aware DVS

In this section, we discuss the inefficiency of independent online DVS policies (Section 4.1), and propose an interaction-aware DVS policy to overcome this inefficiency (Section 4.2).

4.1 MCD inter-domain interactions

Applying DVS independently in an MCD processor creates domain interactions that may negatively affect the performance and/or energy of other domains. We present an example to illustrate the cause of these effects and their undesired implications; the reader is encouraged to follow the numbered steps in Figure 3. (1) Assume an MCD processor is running an application that experiences some pipeline stalls (e.g., due to branch misprediction). The increased number of stalls results in reduced IPC. (2) The independent DVS policy triggers a lower speed setting in the core domain. Slowing down the core will reduce the rate of issuing instructions, including L2 accesses. (3) Fewer L2 accesses per interval causes the independent policy to lower the speed in the L2 cache domain. (4) This, in turn, increases the cache access latency, which (5) causes

[1] The metric to optimize can vary; we have experimented with the usual metrics, namely energy, delay, and the most used, the energy-delay product.

more stalls in the core-domain. Hence, this interaction starts a vicious cycle, which spirals downward.

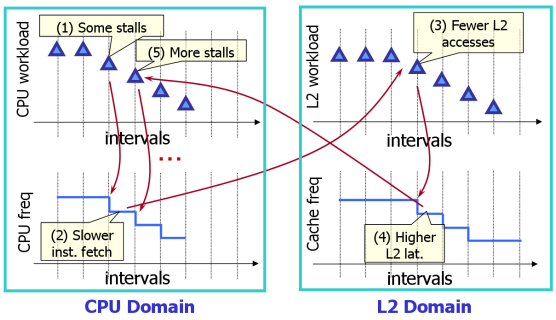


Fig. 3. Example of positive feedback in independent power management in each domain

The duration of this positive feedback[2] depends on the application behavior. For benchmarks with low activity/load variations per domain, this feedback scenario results in low speeds for both domains. While these low speeds reduce power, they clearly hurt performance and do not necessarily reduce total energy-delay product. Analogously, positive feedback may cause increased speeds in both domains, which potentially improves delay at the expense of increasing energy consumption. These two scenarios illustrate that the independent policy may not properly react to a domain's true workload.

These undesired positive feedback scenarios arise from the fact that the independent policy monitors only the local performance of a given domain to set its speed. This local information does not identify whether the source of the load variability is local to a domain or induced by other domains. As a result, the policy cannot take the correct action. In our example, the variation in IPC can be induced by local effects such as executing a large number of floating point instructions or suffering many branch mispredictions. Alternatively, effects from other domains such as higher memory and L2 access latency can induce variations in IPC. Although the effect on IPC is similar, the DVS policy should behave differently in these two cases.

To find out how often applications experience such undesired positive feedback, we analyzed applications under Semeraro et al.'s independent DVS policy [3]. Table 1 illustrates the percentage of time intervals where positive feedback occurs in some Mibench and SPEC2000 benchmarks. The data is collected over a window of 500M instructions (after fast-forwarding simulations for 500M instructions). We divide the execution into 100K instruction intervals then count the percentage of consecutive intervals that experience positive feedback in both the CPU and L2 domains simultaneously. The table shows that some applications experience high rates of positive feedback, while others are largely unaffected. In the former (e.g., *gsm*, *lame*, *rsynth*, *bzip*, *parser*, and *vpr*), we

[2] A positive feedback control is where the response of a system is to change a variable in the same direction of its original change.

Table 1. Percentage of time intervals that experience positive feedback scenarios in some Mibench and SPEC2000 benchmarks.

adpcm_dec	adpcm_enc	basicmath	crc32	gsm_toast	gsm_untoast	lame	rsynth
0.28%	1.73%	0.24 %	0.18%	27.7%	20.09%	22.56%	47.56%
bzip	equake	gcc	gzip	parser	twolf	vortex	vpr
26.22%	13.98%	23.35 %	21.07%	26.44%	23.69%	12.38%	23.73%

expect that the independent policy will result in relatively high delays or high energy because it reacts with inappropriate speed setting for more than 20% of the time.

To have a better indication of the core and L2 cache workloads, the policy has to be aware of the status of both domains, because each domain may indirectly influence the workload in the other domain. This motivates the need for run-time policies that take into account the core and the L2 cache interactions to appropriately set the speeds for both domains in a way that minimizes total energy, delay or energy-delay product.

4.2 Integrated Core and L2 cache DVS policy

In our integrated policy, we monitor the IPC and the number of L2 accesses with performance counters. The speeds are driven by the change in the *combined* status of IPC and number of L2 accesses in a given execution interval. The rate of increase or decrease in speed is based on the rate of increase or decrease in the monitored counter subject to exceeding a threshold as proposed by Zhu et al. [9]. We introduce a new set of rules (listed in Table 2) to be executed by the DVS policy for controlling the speeds. The symbols $\Uparrow$, $\Downarrow$, and $-$ depict an increase, decrease, and no-change in the corresponding metric. Columns 2 and 3 in the table show the change in the monitored counters while columns 4 and 5 (columns 6 and 7) show the action taken by independent (our) policy on the corresponding domain speeds.

Table 2. Rules for adjusting core and L2 cache speeds in independent and proposed policies.

rule #	Event to monitor		Action by independent policy		Action by our integrated policy	
	IPC	L2access	V_c	$V_\$$	V_c	$V_\$$
1	$\Uparrow$	$\Uparrow$	$\Uparrow$	$\Uparrow$	$\Downarrow$	$\Uparrow$
2	$\Uparrow$	$\Downarrow$	$\Uparrow$	$\Downarrow$	$\Uparrow$	$\Downarrow$
3	$\Uparrow$	$-$	$\Uparrow$	$-$	$\Uparrow$	$-$
4	$\Downarrow$	$\Uparrow$	$\Downarrow$	$\Uparrow$	$\Downarrow$	$\Uparrow$
5	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$-$	$\Downarrow$
6	$\Downarrow$	$-$	$\Downarrow$	$-$	$\Downarrow$	$-$
7	$-$	$\Uparrow$	$-$	$\Uparrow$	$-$	$\Uparrow$
8	$-$	$\Downarrow$	$-$	$\Downarrow$	$-$	$\Downarrow$
9	$-$	$-$	$-$	$-$	$-$	$-$

Given the evidence from Table 1, we decided to focus on the positive feedback cases described in Section 4.1. These cases only cause a change in rules 1 and 5 in Table 2, and maintain the other rules exactly the same. It only changes the rules when there is a simultaneous increase or decrease in IPC and L2 cache accesses. As a result, our policy requires minimal changes to existing policies (i.e., it can be readily supported without any additional cost), yet it achieves better energy savings. Contrary to the independent policy, which seems intuitive, our integrated policy does **not** increase (or decrease) the speed if both the counters show an increase/decrease during a given interval. Instead, the policy changes speeds as shown in the table.

Next, we describe both cases and the reasons behind the counter-intuitive actions of our policy.

Simultaneous increase in IPC and L2 cache access (rule 1): Our approach reacts to the first positive feedback case by reducing the core speed rather than increasing it, as in the independent policy. This decision is based on the observation that the increase in IPC was accompanied by an increase in the number of L2 cache accesses. This increase may indicate a start of a program phase with high memory traffic. Hence, we preemptively reduce the core speed to avoid overloading the L2 cache domain with excess traffic. In contrast, increasing the core speed would exacerbate the load in both domains. We choose to decrease the core speed rather than keeping it unchanged to save core energy, especially with the likelihood of longer core stalls due to the expected higher L2 cache traffic.

Simultaneous decrease in IPC and L2 cache access (rule 5): We target the second undesired positive feedback scenario where the independent policy decreases both core and cache speeds. From observing the cache workload, we deduce that the decrease in IPC is not due to higher L2 traffic. Thus, longer core stalls are a result of local core activity such as branch misprediction. Hence, increasing or decreasing the core speed may not eliminate the source of these stalls. By doing so, we risk unnecessarily increasing in the application's execution time or energy consumption. Hence, we choose to maintain the core speed without any change in this case, to break the positive feedback scenario without hurting delay or energy.

5 Evaluation

In this section, we evaluate the efficacy of our integrated DVS policy, which considers domain interactions, on reducing a chip's energy and energy-delay product. We use the Simplescalar and Wattch architectural simulators with an MCD extension by Zhu et al. [9] that models inter-domain synchronization events and speed scaling overheads. To model the MCD design in Figure 1, we altered the simulator kindly provided by Zhu et al. by merging different core domains into a single domain and separating the L2 cache into its own domain. In the independent DVS policy, we monitor the instruction fetch queue to control the core domain, and the number of L2 accesses to control the L2 cache domain.

Since our goal is to devise a DVS policy for an embedded processor with MCD extensions, we use *Configuration A* from Table 3 as a representative of a simple embedded processor (Simplescalar's StrongArm configuration [10]). We use Mibench benchmarks with the *long* input datasets. Since Mibench applications are relatively short, we fast-forward only 500 million instructions and simulate the following 500 million instructions or until benchmark completion.

Table 3. Simulation configurations

Parameter	Config. A (simple embedded)	Config. B (high-end embedded)
Dec./Iss. Width	1/1	4/6
dL1 cache	16KB, 32-way	64KB, 2-way
iL1 cache	16KB, 32-way	64KB, 2-way
L2 Cache	256KB 4-way	1MB DM
L1 lat.	1 cycles	2 cycles
L2 lat.	8 cycles	12 cycles
Int ALUs	2+1 mult/div	4+1 mult/div
FP ALUs	1+1 mult/div	2+1 mult/div
INT Issue Queue	4 entries	20 entries
FP Issue Queue	4 entries	15 entries
LS Queue	8	64
Reorder Buffer	40	80

To extend our evaluation and check whether our policy can be extended to different arenas (in particular, higher performance processors), we also use the SPEC2000 benchmarks and a high-end embedded processor [9] (see *Configuration B* in Table 3). We run the SPEC2000 benchmarks using the *reference* data set. We use the same execution window and fastforward amount (500M) for uniformity.

Our goal is twofold. First, to show the benefit of accounting for domain interactions, we compare our integrated DVS policy with the independent policy described in Section 3. For a fair comparison, we use the same policy parameters and thresholds used by Zhu et al. [9]. The power management controller is triggered every 100K instructions. Moreover, our policy implementation uses the same hardware used in [9], in addition to trivial (low overhead) addition in the monitoring and control hardware of an MCD chip. Second, to quantify the net savings in energy and delay, we compare our policy to a *no-DVS* policy, which runs all domains at highest speed. We show all results normalized to the no-DVS policy.

We first evaluate the policies using an embedded processor (Configuration A in Table 3). Figure 4-a shows that for the Mibench applications, the improvement in the energy-delay product is 15.5% on average (up to 21% in *rsynth*) over no-DVS policy. For the SPEC2000 benchmarks, the improvement in the energy-delay product is 18% on average (up to 26% in *twolf*) over no-DVS policy. Most of the improvement is a result of energy savings (an average of 21% across applications) as seen in Figure 4-b,

with much less performance degradation as seen in Figure 4-c (note different Y-axis scale).

The integrated, interaction-aware policy achieves an extra 7% improvement in energy-delay product above the independent policy gains. These savings are beyond what the independent policy can achieve over the no-DVS policy[3]. The improvement over the independent policy comes from avoiding the undesired positive feedback scenarios by using coordinated DVS control in the core and L2 cache domains. However, the energy-delay product improvement beyond the gain achieved by the independent policy is highly dependent on the frequency of occurrence of the positive feedback scenarios, the duration of the positive feedback and the change in speed during these positive feedback cases throughout the application execution. From Table 1, we notice that the frequency of the positive feedback in *adpcm*, *basicmath*, and *crc32* is almost negligible; accordingly, there are significantly smaller benefits from our policy as shown in Figure 4. On the other hand, applications like *gsm*, *rsynth*, *gcc*, *parser*, and *twolf* show high energy savings due to repeated occurrence of positive feedback cases.

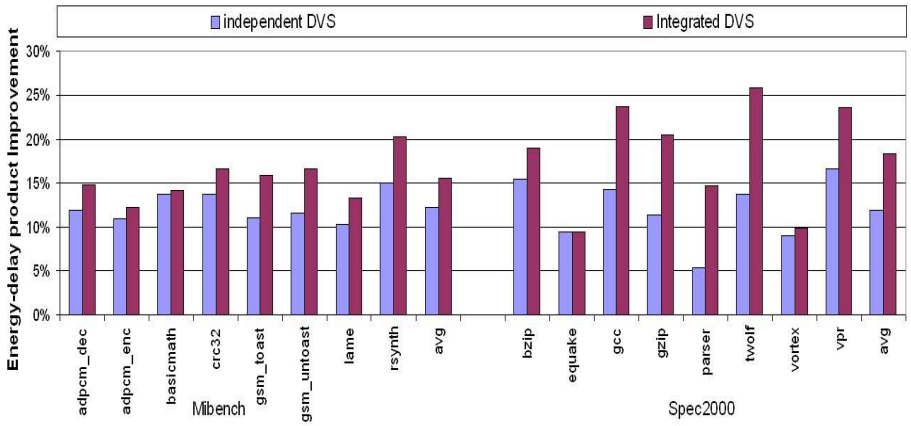
With respect to performance, we note that our proposed integrated policy has a slow-down of 5% on average for Mibench (7% on average for SPEC2000). This slowdown is only 1% more than the slowdown of the independent policy.

To test whether a different policy that avoids the undesired positive feedback scenarios using alternative actions (specifically, different actions for rules 1 and 5 in Table 2) would perform better, we experimented with different rules for these two cases. Table 4 shows the actions of our proposed policy and seven policy variants, in addition to our proposed integrated policy P0. Figure 5 shows the average degradation in energy-delay product relative to the independent policy. It is clear that other actions for dealing with positive feedback scenarios are not as effective in reducing the energy-delay product. The degradation in energy-delay product of the policy variants ranges from 2% to 12% over our proposed policy.

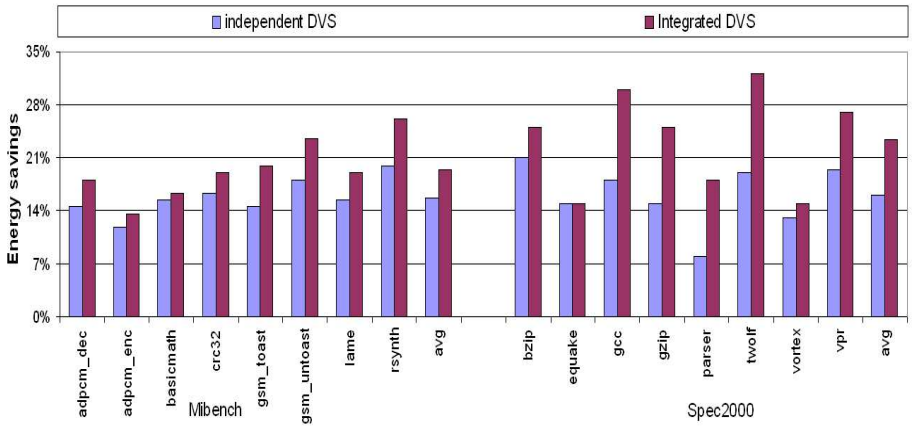
Table 4. Variants of our proposed policy: actions of setting the core voltage (V_c) and the cache speed ($V_\$$) in rules 1 & 5 from Table 2.

rule	P0		P1		P2		P3		P4		P5		P6		P7	
#	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$
1	$\Downarrow$	$\Uparrow$	$\Downarrow$	$-$	$\Downarrow$	$-$	$\Downarrow$	$-$	$-$	$\Uparrow$	$-$	$-$	$-$	$\Uparrow$	$-$	$\Uparrow$
5	$-$	$\Downarrow$	$-$	$\Downarrow$	$-$	$-$	$\Uparrow$	$-$	$-$	$\Downarrow$	$\Uparrow$	$-$	$\Uparrow$	$-$	$-$	$-$

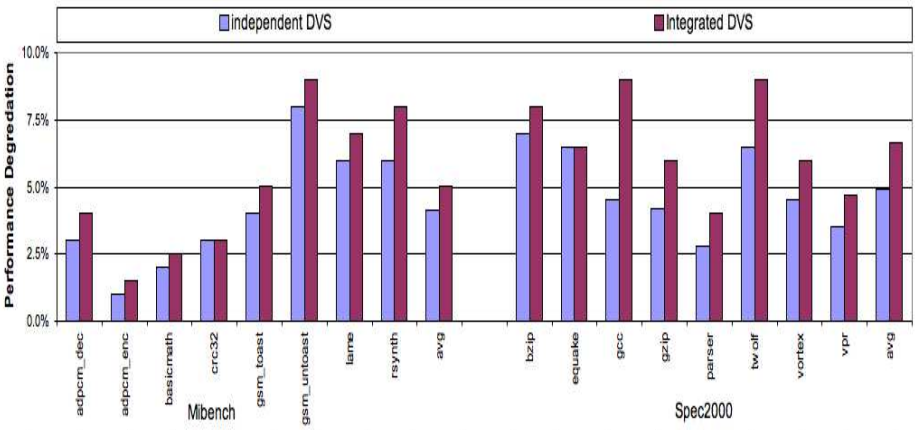
[3] Reported results of the independent policy are not identical to the one reported in [3] due to few reasons: (a) The latest distribution of the MCD simulation tool set has a different implementation of the speed change mechanism. (b) We simulate two-domain processor versus five-domain processor in the original independent policy. (c) We execute applications with different simulation window, as well.



(a) Normalized improvement in energy-delay product



(b) Normalized Energy Savings



(c) Performance degradation

Fig. 4. Energy and delay of independent policy (Independent DVS) and our policy (Integrated DVS) relative to no-DVS policy in configuration A and two voltage domains processor.

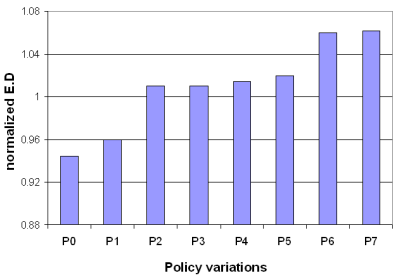
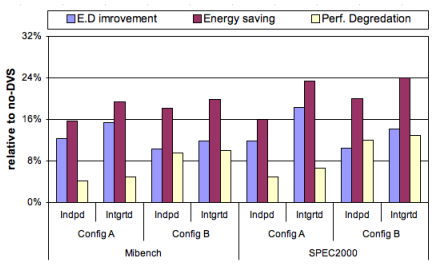


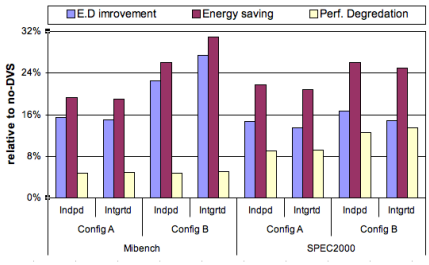
Fig. 5. Average degradation in energy-delay product relative to the independent policy

Sensitivity analysis

We study the benefit of using a domain interaction-aware DVS policy under different system configurations. We explore the state space by varying key processor configurations and the granularity of DVS control (that is, number of MCD domains). In addition to a simple embedded single-issue processor (configuration A in Table 3), we experiment with a more complex embedded processor (configuration B, the same processor configuration used in [9]). This test should identify the benefit of interaction-aware policy in a simple embedded processor versus a more powerful one. This more powerful processor, such as Intel's Xeon 5140 and Pentium M, are used in portable medical, military and aerospace applications [11]. Figure 6-a compares configuration A versus configuration B in terms of energy-delay product, energy saving, and performance degradation. The figure shows the average values over the Mibench and SPEC2000 benchmarks for 2 domains. One observation is that we achieve larger energy-delay improvement in embedded single-issue processor (Config A) than the more complex one (Config B). This larger improvement is mainly due to higher energy savings. In single-issue processors, cache misses cause more CPU stalls (due to lower ILP) than in higher-issue processors. This is a good opportunity for the DVS policy to save energy by slowing down domains with low workloads while having small impact on performance.



(a) 2 domains



(b) 6 domains

Fig. 6. Energy and delay for independent policy (Indpnd) and our integrated policy (Intgrtd) relative to no-DVS policy for processors with (a) two domains and (b) six domains.

Comparing our results with the independent policy, we notice that the average benefit of considering domain interactions decreases with the increase in issue width. This is because processors with small issue width are more exposed to stalls from the memory hierarchy, which makes it important to consider the core and L2 cache domain interaction. In contrast, with wider issue width, these effects are masked by the core's higher ILP. This result shows that applying the integrated policy can benefit simple embedded processors. Whereas energy-delay savings in high-end embedded processor do not favor the use of the integrated policy over independent counterpart.

Because we are also interested in the effect of interactions across multiple domains on energy savings, we examined the effect of increasing the number of clock domains. To perform this test, we simulated the five domains used in [3], but added a separate domain for the L2 cache. The resultant domains are: reorder buffer domain, fetch unit, integer FUs, floating point FUs, load/store queue, and L2 cache domains. We use our policy to control the fetch unit and L2 domains, and set the speeds of the remaining domains using the independent policy [3] [9].

Figure 6-b shows the results for the two processor configurations when dividing the chip into 6-domains. Comparing Figures 6-a and 6-b, we find that DVS in processors with large number of domains enables finer-grain power management, leading to larger energy-delay improvements. However, for embedded systems, a two-domain processor is a more appropriate design choice when compared to a processor with a larger number of domains (due to its simplicity). Figure 6 shows that increasing the number of domains had little (positive or negative) impact on the difference in energy-delay product between our policy and the independent policy. This indicates that the core-L2 cache interaction is most critical in terms of its effect on energy and delay, which yielded higher savings in the two-domain case. We can conclude that a small number of domains is the most appropriate for embedded processors, not only from a design perspective but also for improving energy-delay.

6 Related Work

MCD design has the advantages of alleviating some clock synchronization bottlenecks and reducing the power consumed by the global clock network. Semeraro et al. explored the benefit of the voltage scaling in MCD versus globally synchronous designs [3]. They find a potential 20% average improvement in the energy-delay product. Similarly, Iyer at al. analyzed the power and performance benefit of MCD with DVS [4]. They find that DVS provides up to 20% power savings over an MCD core with single voltage.

In industrial semiconductor manufacturing, National Semiconductor in collaboration with ARM developed the PowerWise technology that uses Adaptive Voltage Scaling and threshold scaling to automatically control the voltage of multiple domains on chip [1]. The PowerWise technology can support up to 4 voltage domains [12]. Their current technology also provides power management interface for dual-core processors.

Another technique by Magklis et al. is a profile-based approach that identifies program regions that justify reconfiguration [5]. This approach involves extra overhead of profiling and analyzing phases for each application. Zhu et al presented architectural optimizations for improving power and reducing complexity [9]. However, these poli-

cies do not take into account the cascading effect of changing a domain voltage on the other domains.

Rusu et al. proposed a DVS policy that controls the domain's frequency using machine learning approach [13][14]. They characterize applications using performance counter values such as cycle-per-instruction and number of L2 accesses per instruction. In a training phase, the policy searches for the best frequency for each application phase. During runtime, based on the values of the monitors performance counters, the policy sets the frequency for all domains based on their offline analysis. The paper shows improvement in energy-delay product close to a near-optimal scheme. However, the technique requires an extra offline training step to find the best frequencies for each domain and application characterization.

Wu et al. present a formal solution by modeling each domain as a queuing system [6]. However, they study each domain in isolation and incorporating domain interactions increases the complexity of the queuing model. Varying the DVS power management interval is another way to save energy. Wu et al. adaptively vary the controlling interval to react to changes in workload in each domain was presented in [15]. They do not take into account the effect induced by voltage change in one domain on the other domains.

MCD design is applied for the multicore and simultaneous multithreading processors such as in [16][17][18]. In [16][17], each core has its own clock network, and the DVS policy independently controls each core's voltage. Lopez et al. studies the trade-off between adapting the L2 cache capacity and speed based on the number of active threads in the core domain [18].

7 Conclusion

In MCD processors, applying DVS in each domain can significantly reduce energy consumption. However, varying the voltage and clock independently in each domain indirectly affects the workload in other domains. This results in an inefficient DVS policy. In this paper, we identify these inefficiencies in online MCD-DVS policies, and propose a simple DVS policy that accounts for inter-domain interactions. Our policy separately assigns the voltage and clock of the core and L2 cache domains based on activity in **both** domains. We show that our policy achieves higher energy and energy-delay savings than an MCD DVS policy that is oblivious to domain interactions. Our policy achieves average savings in energy-delay product of 18.5% for the SPEC2000 and 15.5% for the Mibench suites. Moreover, our policy achieves higher savings in energy-delay product over past independent DVS approaches (7% for SPEC2000 and 3.5% for Mibench benchmarks) using the same hardware. We also show that processors with narrow issue widths have a larger improvement in the energy-delay product with our integrated DVS policy. Finally, our results show that a simple MCD design using two domains is more energy efficient for simple embedded processors than for high-end ones.

References

1. National Semiconductor, "PowerWise Technology", 2007, http://www.national.com/appinfo/power/powerwise.html.
2. G. Magklis, G. Semeraro, D.H. Albonesi, S.G. Dropsho, S. Dwarkadas and M.L. Scott, "Dynamic Frequency and Voltage Scaling for a Multiple-Clock-Domain Microprocessor", *IEEE Micro*, vol. 23, n. 6, pp. 62–68, 2003.
3. G. Semeraro, D.H. Albonesi, S.G. Dropsho, G. Magklis, S. Dwarkadas and M.L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture", in *MICRO 35: Proc Intl Symp on Microarchitecture*, pp. 356–367, 2002.
4. A. Iyer and D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors", in *ISCA'02: Proc Intl Symp on Computer architecture*, pp. 158–168, 2002.
5. G. Magklis, M.L. Scott, G. Semeraro, D.H. Albonesi and S. Dropsho, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor", in *ISCA'03: Proc Intl Symp on Computer Architecture*, pp. 14–27, 2003.
6. Q. Wu, P. Juang, M. Martonosi and D.W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors", in *ASPLOS-XI: Proc Intl Conf on Architectural support for programming languages and operating systems*, pp. 248–259, 2004.
7. M. Ben Naser and C.A. Moritz, "A Step-by-Step Design and Analysis of Low Power Caches for Embedded Processors", 2005, http://www.lems.brown.edu/ iris/BARC2005/Webpage/BARCpresentations/ben-naser.pdf.
8. S. Kaxiras, Z. Hu and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power", in *ISCA'01: Proc Intl Symp on Computer Architecture*, pp. 240–251, 2001.
9. Y. Zhu, D.H. Albonesi and A. Buyuktosunoglu, "A High Performance, Energy Efficient GALS Processor Microarchitecture with Reduced Implementation Complexity", in *ISPASS'05: Proc Intl Symp on Performance Analysis of Systems and Software*, 2005.
10. SimpleScalar/ARM, "SimpleScalar-Arm Version 4.0 Test Releases", http://www.simplescalar.com/v4test.html/.
11. Intel Embedded products, "High-Performance Energy-Efficient Processors for Embedded Market Segments", 2006, http://www.intel.com/design/embedded/downloads/315336.pdf.
12. J. Pennanen, "Optimizing the Power for Multiple Voltage Domains", 2006, Spring Processor Forum, Japan.
13. C. Rusu, N. AbouGhazaleh, A. Ferreria, R. Xu, B. Childers, R. Melhem and D. Mossé, "Integrated CPU and L2 cache Frequency/Voltage Scaling using Supervised Learning", in *Workshop on Statistical and Machine learning approaches applied to ARchitectures and compilation (SMART)*, 2007.
14. N. AbouGhazaleh, A. Ferreria, C. Rusu, R. Xu, B. Childers, R. Melhem and D. Mossé, "Integrated CPU and L2 cache Voltage Scaling using Supervised Learning", in *LCTES '07: Proc. of ACM SIGPLAN on Language, compiler, and tool for embedded systems*, 2007.
15. Q. Wu, P. Juang, M. Martonosi and D.W. Clark, "Voltage and Frequency Control With Adaptive Reaction Time in MCD Processors", in *HPCA'05: Proc Intl Symp on High-Performance Computer Architecture*, pp. 178–189, 2005.
16. J. Oliver, R. Rao, P. Sultana, J. Crandall, E. Czernikowski, L.W. Jones IV, D. Franklin, V. Akella and F.T. Chong, "Synchroscalar: A Multiple Clock Domain, Power-Aware, Tile-Based Embedded Processor", in *ISCA'04: Proc Intl Symp on Computer Architecture*, 2004.
17. P. Juang, Q. Wu, L.S. Peh, M. Martonosi and D.W. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors", in *ISLPED'05: Proc Intl Symp on Low Power Electronics and Design*, pp. 127–130, 2005.
18. S. Lopez, S. Dropsho, D. Albonesi, O. Garnica and J. Lanchares, "Dynamic Capacity-Speed Tradeoffs in SMT Processor Caches", in *High Performance Embedded Architecure and Compilation (HiPEAC)*, 2007.