RC23280 (W0407-059) July 26, 2004
Computer Science

IBM Research Report

A High Perfomance, Energy Efficient GALS Processor Microarchitecture with Reduced Implementation Complexity

Yongkang Zhu
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627

David H. Albonesi
Department of Electrical and Computer Engineering
Cornell University
Ithaca, NY 14853

Alper Buyuktosunoglu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A High Performance, Energy Efficient GALS Processor Microarchitecture with Reduced Implementation Complexity

YongKang Zhu
Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, New York 14627
Email: yozhu@ece.rochester.edu

David H. Albonesi
Dept. of Electrical and Computer Engineering
University of Conell
Ithaca, New York 14853
Email: albonesi@ece.rochester.edu

Alper Buyuktosunoglu
IBM T. J. Watson Research Center
Yorktown Heights, New York 10598
Email: alperb@us.ibm.com

Abstract

As the costs and challenges of global clock distribution grow with each new microprocessor generation, a Globally Asynchronous, Locally Synchronous (GALS) approach becomes an attractive alternative. One proposed GALS approach, called a Multiple Clock Domain (MCD) processor, achieves impressive energy savings for a relatively low performance cost. However, the approach requires separating the processor into four domains, including separating the integer and memory domains which complicates load scheduling, and the implementation of 32 voltage and frequency levels in each domain. In addition, the hardware-based control algorithm, though effective overall, produces a significant performance degradation for some applications.

In this paper, we devise modifications to the MCD design that retain many of its benefits while greatly reducing the implementation complexity. We first determine that the synchronization channels that are most responsible for the MCD performance degradation are those involving cache access, and propose merging the integer and memory domains to virtually eliminate this overhead. We further propose significantly reducing the number of voltage levels, separating the Reorder Buffer into its own domain to permit front-end frequency scaling, separating the L2 cache to permit standard power optimizations to be used, and a new online algorithm that provides consistent results across our benchmark suite. The overall result is a significant reduction in the performance degradation of the original MCD approach and greater energy savings, with a greatly simplified microarchitecture that is much easier to implement.

Keywords GALS, Multiple Clock Domain (MCD) Processor, Complexity-Effective Design

A High Performance, Energy Efficient GALS Processor Microarchitecture with Reduced Implementation Complexity

YongKang Zhu*, David H. Albonesi† and Alper Buyuktosunoglu‡

*Department of Electrical and Computer Engineering, University of Rochester
†Department of Electrical and Computer Engineering, University of Cornell
‡IBM T. J. Watson Research Center

Abstract

As the costs and challenges of global clock distribution grow with each new microprocessor generation, a Globally Asynchronous, Locally Synchronous (GALS) approach becomes an attractive alternative. One proposed GALS approach, called a Multiple Clock Domain (MCD) processor, achieves impressive energy savings for a relatively low performance cost. However, the approach requires separating the processor into four domains, including separating the integer and memory domains which complicates load scheduling, and the implementation of 32 voltage and frequency levels in each domain. In addition, the hardware-based control algorithm, though effective overall, produces a significant performance degradation for some applications.

In this paper, we devise modifications to the MCD design that retain many of its benefits while greatly reducing the implementation complexity. We first determine that the synchronization channels that are most responsible for the MCD performance degradation are those involving cache access, and propose merging the integer and memory domains to virtually eliminate this overhead. We further propose significantly reducing the number of voltage levels, separating the Reorder Buffer into its own domain to permit front-end frequency scaling, separating the L2 cache to permit standard power optimizations to be used, and a new online algorithm that provides consistent results across our benchmark suite. The overall result is a significant reduction in the performance degradation of the original MCD approach and greater energy savings, with a greatly simplified microarchitecture that is much easier to implement.

1. Introduction

Advances in semiconductor technology, novel circuit techniques, and innovation in computer architecture have resulted in rapid improvements in microprocessor performance. Today, hundreds of millions of transistors are successfully harnessed to build these increasingly complex devices. However, during the next two or three generations, high end microprocessor designers will face several major challenges. Without argument, one of the biggest challenges will be to keep power dissipation to reasonable levels. Higher clock frequencies and transistor counts have made power dissipation a major microprocessor design constraint, so much so that it threatens to limit the amount of hardware that can be included on future microprocessors and how fast they can be clocked. Another impending limit will be the global clock distribution design due to larger die sizes and higher clock speed. Distribution of a high frequency global clock signal with low clock skew can be prohibitively expensive in terms of design effort, area, and power consumption under such circumstances. In addition, significant across chip and across wafer parameter variations are added sources of concern for future microprocessors.

In such an environment, globally asynchronous, locally synchronous (GALS) designs provide several benefits through their use of separate, autonomous units:

- *The capability to independently configure each domain to execute at frequency/voltage settings at or below the maximum values.* This allows domains that are not executing operations critical to performance to be configured at a lower frequency, and consequently, an GALS microarchitecture has the advantage that power can be saved.

- *Elimination of the need for careful design and fine tuning of a global clock distribution network.* Through local clock generation units, the problem of dealing with clock distribution can be confined into several smaller domains. For example, the impact of parameter variations on clock skew will be confined within a domain, and thus will require less design effort and cost for dealing with clock skew.

- *The ability for each domain frequency to track with parameter variations.* In the case of frequency, each domain can statically run at different frequencies (increasing effective *average* maximum frequency) by tracking the variations from V_{dd} noise, L_{eff}, as well as from temperature. For example, if one of the domains has one sigma slow L_{eff}, the frequency can be lowered for that domain while the other domains can run with a relatively higher frequency. On the contrary, if most of the domains have one sigma fast L_{eff}, the V_{dd} can be statically lowered for the same performance to save power.

For these reasons, there is a rapidly growing interest among high end microprocessor designers in adopting a GALS approach for future products. One major concern regarding the GALS processor efforts proposed to date [7, 12] is increased design and verification complexity. For instance, the Multiple Clock Domain (MCD) approach of [12] requires separating the processor into four domains (including separating the integer and memory domains which complicates the scheduling of integer load-dependent instructions), and the implementation of 32 voltage and frequency levels in each domain. In addition, the hardware-based control algorithm, though effective overall, produces a significant performance degradation for some applications. In order for GALS processor designs like MCD to be implemented in practice, *complexity effective* design simplifications [1] must be discovered. That is, design modification that significantly reduce design and verification complexity, while retaining virtually all of the original power/performance benefits, must be devised.

In this paper, we make significant simplifications to the MCD design, and discover additional optimizations that require very minor design changes. Using the framework developed by Semeraro et al. [12, 10, 8], we evaluate the performance and energy cost of modifications to the MCD design that greatly reduce the implementation complexity. We first determine that the synchronization channels that are most responsible for the MCD performance degradation are those involving cache access, and propose merging the integer and memory domains to virtually eliminate this overhead. Perhaps more significantly, this modification greatly simplifies the integer out-of-order scheduler, as load hit latencies are no longer subject to a domain crossing, and are thus deterministic. Load-dependent integer instructions can therefore be scheduled as in a fully synchronous design. We then propose significantly reducing the number of frequency/voltage levels, separating the Reorder Buffer into its own

domain to permit front-end frequency scaling, separating the L2 cache to permit it to be optimized using standard techniques, and a new online algorithm that provides consistent results across our benchmark suite. The overall result is a significant reduction in the performance degradation of the original MCD approach with greater energy savings, with a greatly simplified microarchitecture that is much easier to implement.

The rest of this paper is organized as follows. The concept of complexity effective design is briefly described in Section 2, as well as how we adopt this idea to our work. The microarchitecture of the original MCD design and potential modifications are described in Section 3. Section 4 elaborates in detail on each of our proposed MCD design modifications and presents experimental results. Section 5 discusses in detail the related work, and we finally conclude in Section 6.

2. Complexity-Effective Design

To our knowledge, the term *complexity-effective* was first used in the context of superscalar processors by Palacharla et al. [9] to describe a proposed *dependence-based* microarchitecture that organized execution resources into clusters. The goal was to permit the design to scale to a fast clock while still achieving high IPC performance.

Since then, the term has been modified for a series of workshops on Complexity-Effective Design held the last few years at ISCA. The Introduction in the 2004 workshop proceedings states the following: "A complexity-effective design feature or tool either (a) yields a significant performance and/or power efficiency improvement relative to the increase in hardware/software complexity incurred; or (b) significantly reduces complexity (design time and/or verification time and/or improved scalability) with a tolerable performance and/or power impact."

We adopt this definition in this paper, and strive to find design features that accomplish both (a) and (b) above for the MCD microarchitecture. Since the definition of "significant" is subject to interpretation, and there is no known metric for "complexity", we instead make qualitative arguments that our proposed modifications are indeed "complexity-effective".

3. MCD Microarchitecture and Overview of Potential Modifications

The MCD processor was first proposed by Semeraro et al. in [12]. The basic idea is to divide the chip into multiple domains each with its own clock and voltage generators to permit tuning each domain frequency and voltage independent of the other domains. Architectural queues that decouple different pipeline functions serve as the interfaces between domains, and are augmented with synchronization circuitry to ensure that signals on different time bases transfer correctly. Figure 1 shows the domain partitions proposed in [12].

Previous papers have described several control mechanisms to choose when, and to what values, to change domain frequencies and voltages. The *offline algorithm* [12] post-processes an application trace to find, for each interval, the configuration parameters that would have minimized energy, subject to a user-selected acceptable slowdown threshold. The application trace is built up gradually when running programs in a fully synchronous machine simulator. All primitive events (instruction fetch, decode, issue and commit) are collected with time stamps and added into a directed acyclic graph (DAG) with edges (with lengths) enforcing various data and structural dependences. At the end of each interval the constructed DAG is analyzed to calculate all the "slacks", and

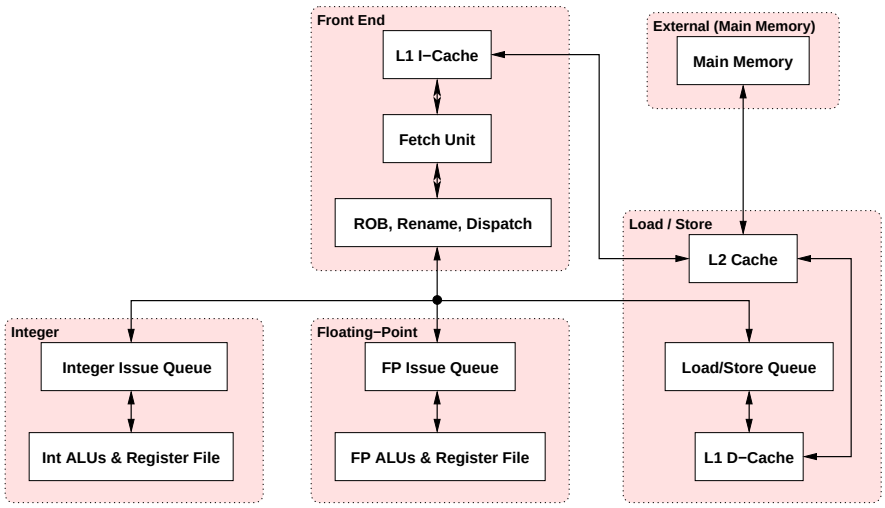


Figure 1. MCD architecture proposed in [12].

based on which, the final frequency settings are determined. Though impractical, this algorithm provides a target against which to compare more realistic alternatives. A more practical control scheme is the *on-line (attack/decay) algorithm* [10], which makes reconfiguration decisions dynamically during the execution of a program. However, compared to the *off-line* it achieves less energy efficiency and at times produces significantly high performance degradation.

From comparing a conventional microprocessor with an MCD design, and from reading the authors' own self-criticism regarding MCD [12, 10, 8], we have targeted several areas for complexity-effective design improvements:

Reducing the complexity of the online control algorithm, or improving its effectiveness. The online control algorithm has the advantage that it is transparent to software and therefore can be effective on legacy applications as well as new ones. Therefore, it is critical to the success of MCD, yet it is an additional complexity over the fully synchronous design. We take the viewpoint that simplifying the online algorithm is difficult with little impact on its effectiveness and therefore we seek to achieve the opposite: increase the algorithm's effectiveness, in particular its *robustness* across a variety of applications, with little impact on its complexity.

Reducing the number and complexity of the synchronization circuits. A second obvious complexity is the need to synchronize signals that cross domains. Such circuits need to be carefully designed to avoid metastability, and they also introduce considerable verification complexity. A reduction in the number of these circuits that are needed would simplify the MCD design. We perform an analysis to determine which synchronization channels are most responsible for the MCD performance degradation, in order to determine if they can be eliminated. Doing so would reduce the performance overhead and reduce the required number of synchronization circuits.

Reducing the dynamic scheduler complexity. A less obvious complication in MCD is created by placing the integer register file and execution units, and the load-store queue and L1 data cache, into separate domains. This means that for an integer load operation, a domain crossing must be incurred after the effective address (EA) is calculated to access the cache, and a second crossing must be made to place the data into the integer register file. Placing dedicated EA calculation logic in the memory domain does not remove either crossing as the register

file still needs to be accessed. The presence of these two domain crossings, together with the fact that the two domains may run at different frequencies, complicates the scheduling of load-dependent instructions. In many conventional synchronous designs, the out-of-order issue logic schedules load-dependent instructions assuming that the load will hit in the cache. Because the load hit latency is constant, this is easily achieved. In MCD, this latency is variable which requires some additional mechanism to know when a load is to return. To eliminate this problem for integer instructions (the complication remains for floating point instructions, although one domain crossing is removed for floating point loads), we propose to investigate the impact on MCD performance and energy efficiency of merging the integer and memory domains.

Saving front-end power. The authors of [12, 10, 8] state that they have had little success in reducing the front-end frequency without a large performance degradation. This leaves a considerable amount of potential power savings untapped. We surmise that the reason for this unfavorable outcome is due to the placement of the Reorder Buffer (ROB) in the front-end. This choice by the MCD designers means that whenever the front-end frequency is scaled down, the commit bandwidth is also scaled commensurately. We thus propose to move the ROB out of the front-end domain and always run it at full frequency. Because the ROB constitutes less than 1% of the total chip power [2], if this indeed does permit the front-end to scale then significant additional power savings can be realized with a small design change.

Separating out the L2 cache. The L2 cache by virtue of its size is a significant source of leakage power in today's aggressive process technologies. Many commercial designs solve this problem by using circuit-level power saving techniques, such as the use of thick oxides and high-V_t transistors, in the L2 cache, and running at a reduced speed, *e.g.*, at half of the processor core speed. This reduces the L2 bandwidth but many applications are unaffected. In effect, the L2 cache is already in its own "domain" in modern processors. We propose, therefore, to remove the L2 cache from the memory domain and run it at half speed. Our power calculations no longer take the L2 cache into account, as it is already low power and no longer part of the core MCD logic that we wish to optimize.

Simplifying the dynamic frequency and control circuits. An obvious complexity in MCD is that it assumes that voltage and frequency circuits having 32 levels can be implemented in each of its domains. Although local PLLs can potentially be devised to meet the frequency requirement, voltage regulators for dynamic microprocessor voltage scaling are implemented at the board level. Having four of these off-chip regulators would consume pins and create significant OEM problems. Proposals for on chip regulators assume linear circuits which have poor efficiency and thus would waste considerable power. We propose therefore to investigate MCD designs having four, or preferably two, levels in each domains. If this can be accomplished while still being profitable from an energy savings perspective, then simple voltage switch circuits, such as that proposed for drowsy caches [6], can be adopted.

In the next section, we describe and evaluate these proposed changes in detail.

Configuration Parameter	Value
Branch predictor:	
Level 1	1024 entries, history 10
Level 2	1024 entries
Bimodal predictor size	1024
Combining predictor size	4096
BTB	4096 sets, 2–way
Branch Mispredict Penalty	7
Decode/Issue/Retire Width	4/6/11
L1 Data Cache	64KB, 2–way set associative
L1 Instruction Cache	64KB, 2–way set associative
L2 Unifi ed Cache	1MB, direct mapped
L1 cache latency	2 cycles
L2 cache latency	12 cycles
Integer ALUs	4 + 1 mult/div unit
Floating–Point ALUs	2 + 1 mult/div/sqrt unit
Integer Issue Queue Size	20 entries
Floating–Point Issue Queue Size	15 entries
Load/Store Queue Size	64
Physical Register File Size	72 integer, 72 fbating–point
Reorder Buffer Size	80

Table 1. Alpha 21264–like architectural parameters.

Benchmark	Simulation Window
adpcm_decode	entire program (11.2M)
adpcm_encode	entire program (13.3M)
epic_decode	entire program (10.6M)
epic_encode	entire program (54.1M)
g721_decode	0 – 200M
g721_encode	0 – 200M
gsm_decode	entire program (122.1M)
gsm_encode	0 – 200M
jpeg_compress	entire program (153.4M)
jpeg_decompress	entire program (36.5M)
mpeg2_decode	0 – 200M
mpeg2_encode	0 – 200M
applu	650 – 850M
art	13,398 – 13,598M
equake	4,266 – 4,466M
gcc	2,000 – 2,200M
gzip	21,185 – 21,385M
mcf	1,325 – 1,525M
swim	575 – 775M
vpr	1,600 – 1,800M

Table 2. Benchmarks and simulation windows.

4. Analysis of Proposed Modifications

In this section, we analyze the proposed complexity-effective design modifications using the MCD simulation framework, which is based on the SimpleScalar/Wattch toolkit [3, 4]. The simulator includes the original online (attack/decay) algorithm and the offline algorithm proposed in [12]. We also implemented our modified attack/decay algorithm (as described in Section 4.1). The microarchitecture parameters were chosen to match those in prior MCD evaluations (see Table 1). In the original MCD simulator, Wattch calculated energy by adding up the power numbers cycle by cycle. This is fine for a fully synchronous machine but may over-estimate the energy savings in a MCD processor. In our experiments, we modified the energy accounting code to account for the dynamically varying clock cycle time. In all experiments, the voltage change is limited to a rate of 16.7 mV per μs.

In our experiments comparing our modified online algorithm with the original one, for convenience and fairness, we used the same set of benchmarks and the same simulation windows (see Table 2) as what was used in [8]. We added another benchmark (*gcc*) which was not used in [8], since it has much more memory references (over 65% of the total number of instructions in the simulated window) than other 7 SPEC2000 benchmark programs. Thus, our benchmark suite consists of 12 MediaBench programs and 8 SPEC2000 programs, of which four are floating-point programs and four are integer programs. In the other experiments, we only use the 8 SPEC2000 benchmarks.

4.1. Improving the attack/decay algorithm

The original attack/decay algorithm as proposed by Semeraro et al. in [10] periodically monitors the issue queue occupancy. Whenever the queue occupancy changes by more than a **fi xed** *Threshold*, the domain's frequency is changed by a **fi xed** *ReactionRatio*. This is the **attack** part. If the queue occupancy change is within the *Threshold*

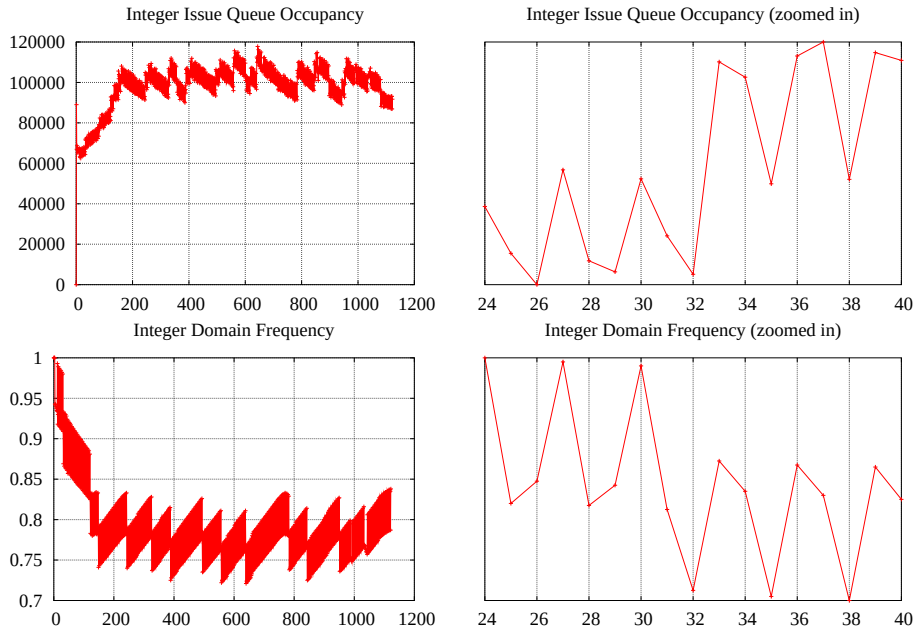


Figure 2. The mismatch between the curve of the integer issue queue occupancy (upper left) and the curve of the frequency (bottom left) selected by the original attack/decay algorithm, for *adpcm_decode*. The horizontal axis indicates the time intervals. Zoomed-in curves at the right side show where the mismatch starts to happen. From Interval 30 to 33, there are two consecutive small drops in the queue occupancy, followed by a large increase, and all these three changes are above the threshold, barely for the first two but greatly for the third. Because *the ReactionRatio is fixed*, the change on frequency triggered by a large change ends up *being the same* as that by a small change. As a result, at Interval 33, the frequency selected are much lower than what it should be.

and IPC does not increase by more than the pre-specified performance degradation, then the domain's frequency is lowered slightly. This is the **decay** part.

Suppose the queue occupancy accurately reflects the program demand at different phases of execution. We expect the frequency to change according to the variation in queue occupancy. But due to the **fixed** *ReactionRatio*, the magnitude of the frequency change due to a very large variation in queue occupancy would be the same as due to a smaller one which is just barely larger than the *Threshold*.

This uniform attack would build up the mismatch between the curves of queue occupancy and frequency gradually (see Figure 2), and finally after a long period, either higher performance degradation (see *adpcm_decode* in Figure 3) or less energy savings would occur.

We propose to modify the *attack* part of the algorithm, to react differently to different queue occupancy changes. The higher the queue occupancy changes, the greater we attack (that is, the more aggressively we adjust the frequency).

We assume the same *ReactionRatio* and *Threshold* used in the original algorithm, and propose three options on how to scale the actual reaction ratio, assuming V = queue_occupancy_change_ratio / *Threshold*:

- *case C*: the frequency changes *conservatively* upon the change of queue occupancy, and thus $ActualReactionRatio = ReactionRatio * \sqrt[2]{V}$;
- *case M*: the frequency changes *moderately* upon the change of queue occupancy, and thus $ActualReactionRatio = ReactionRatio * V$;
- *case A*: the frequency changes *aggressively* upon the change of queue occupancy, and thus $ActualReactionRatio = ReactionRatio * V^2$.

Figure 3 shows the effects of the modified online algorithms. On average, the modified algorithm achieves higher energy savings with lower performance degradation (except for case A, where the performance degradation is slightly higher than that of the original algorithm). Case M and A have almost the same energy efficiency as what the near-optimal offline algorithm can achieve.

For benchmarks *adpcm_decode* and *epic_decode* where the original algorithm produces high performance degradation, our modified schemes significantly reduced the performance degradation. Moreover, greater energy savings is achieved for *epic_decode*.

In comparing the three cases of the modified algorithm, Cases *M* and *A* achieve similar performance degradation and energy savings, while Case *C*, due to its conservative reactions to queue occupancy change, results in a lower performance degradation and hence lower energy savings, but is still superior to the original algorithm. Our new algorithms are very robust across our benchmark suite. Overall, Case *M* achieves a 18% energy (*not* power) savings for a 6% performance degradation. We favor Case *M* as the modified algorithm for its design simplicity, and use it in our further investigations.

4.2. Synchronization channel analysis

Since the different domains of the MCD processor may operate at different frequencies, inter-domain crossings incur synchronization penalties [12]. More specifically, if the data to be transferred is latched at the rising clock edge of the transmitting domain, and the next rising clock edge of the receiving domain is too close in time, then an extra clock cycle is assumed for the data to be latched at the receiving domain. This assumption is consistent with the operation of inter-domain synchronization circuits [5]. To reduce design complexity, we would like to simplify such circuits if the impact is limited. Even better, we seek to remove channels with a large performance impact and where the energy cost is low. Therefore, in this section, we study the performance impact of each inter-domain channel to determine the sensitivity to performance in order to focus on the most important channels.

Figure 4 shows all the synchronization channels. We use this opportunity to separate the ROB and L2 cache out to study these effects as well. This adds two additional channels (the original MCD design has 15 synchronization channels; see [11] for details): Channel 16, where synchronization occurs when an instruction is being dispatched; and Channel 17, for synchronizing transfers between the L1 data cache and L2 unified cache.

Figure 5 shows what percentage of the total synchronization penalty occurs in each channel for our benchmark suite. We place *vpr* in the floating point group due to its large number of floating-point loads and because its distribution has a greater resemblance to the floating point benchmarks. For the floating point benchmarks, Channel 5

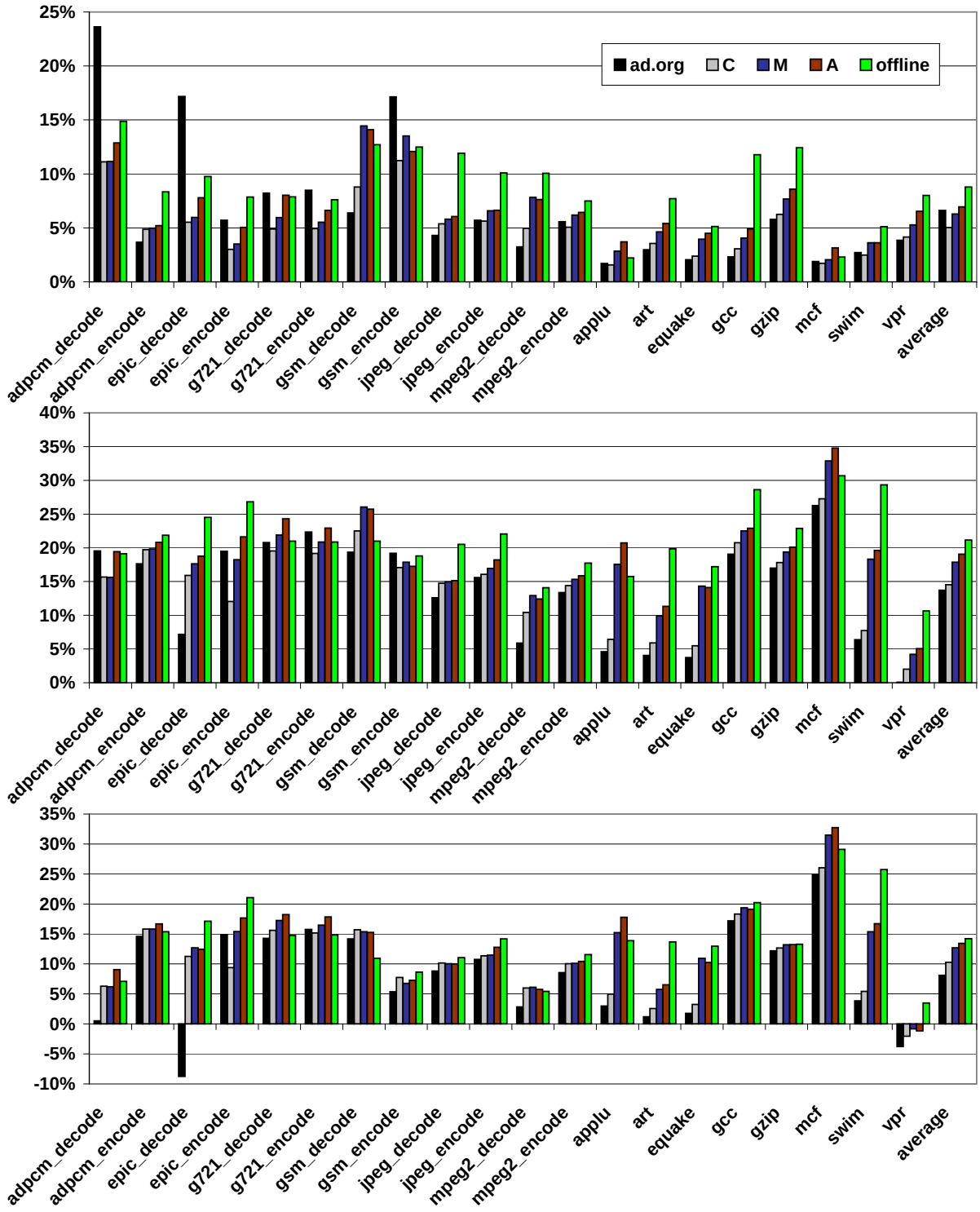
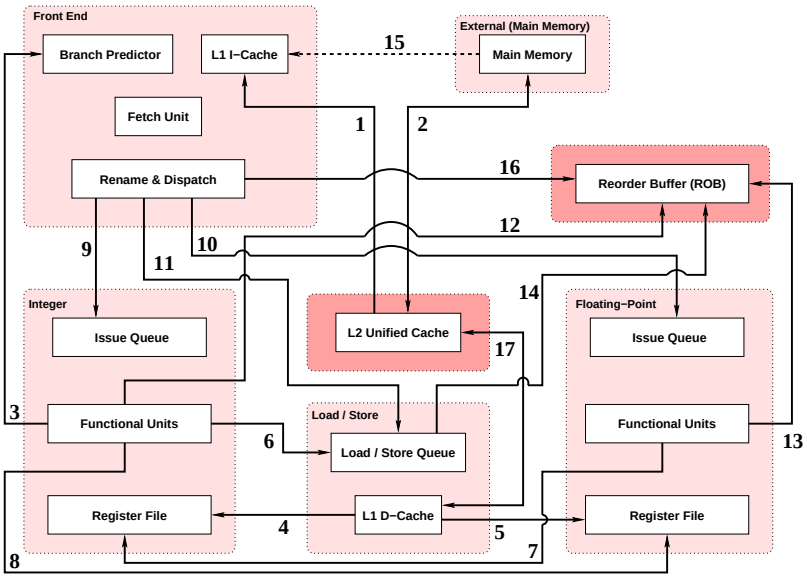


Figure 3. Performance degradation (upper), energy savings (middle), and energy delay product improvement (bottom) using the original (*ad.org*) and modified attack/decay (with 3 cases: C, M and A) algorithms and the offline algorithm, on the original MCD processor. The baseline is a fully synchronous machine with the same architectural parameters.



Synchronization channels:

- *Channel 1*: UL2 Cache ⇒ IL1 Cache
- *Channel 2*: Main Memory ⇔ UL2 Cache
- *Channel 3*: Integer ALU ⇒ Branch Predictor
- *Channel 4*: DL1 Cache ⇒ Integer Register File
- *Channel 5*: DL1 Cache ⇒ FP Register File
- *Channel 6*: Integer Result Bus ⇒ Load/Store Queue
- *Channel 7*: FP Result Bus ⇒ Integer Register File
- *Channel 8*: Integer Result Bus ⇒ FP Register File
- *Channel 9*: Fetch Queue ⇒ Integer Issue Queue
- *Channel 10*: Fetch Queue ⇒ FP Issue Queue
- *Channel 11*: Fetch Queue ⇒ Load/Store Queue
- *Channel 12*: Integer Functional Units ⇒ Reorder Buffer
- *Channel 13*: FP Functional Units ⇒ Reorder Buffer
- *Channel 14*: Load/Store Queue ⇒ Reorder Buffer
- *Channel 15*: Main Memory ⇒ IL1 Cache
- *Channel 16*: Fetch Queue ⇒ Reorder Buffer
- *Channel 17*: UL2 Cache ⇔ DL1 Cache

Figure 4. Our proposed MCD domain partition scheme, along with all possible synchronization channels among different domains.

is the hot spot where roughly 50% of the total synchronization penalty occurs, whereas there is no such dominant channel for the integer benchmarks that on average contributes more than 25% of the total.

However, the magnitude of the synchronization penalties may not directly reflect the performance impact. Rather, the tolerance of the application to added latency on each of the channels is a more significant factor. To determine the performance impact of each channel's synchronization penalty, we individually remove the channel penalties and measure the resulting performance improvement. We only present those channels for which, for at least one benchmark program, a performance impact of at least 0.5% was achieved when the synchronization penalties were removed. Results for Channel 16 and 17 are also presented (although no benchmark program sees more than 0.5% impact), since they are the extra channels resulted from our new domain partition scheme. As is shown in Figure 6, Channels 6, 4, and 5 have the highest performance impact. These three channels are related to load operations: Channel 6 for effective address transfer between the integer and memory domains, and Channels

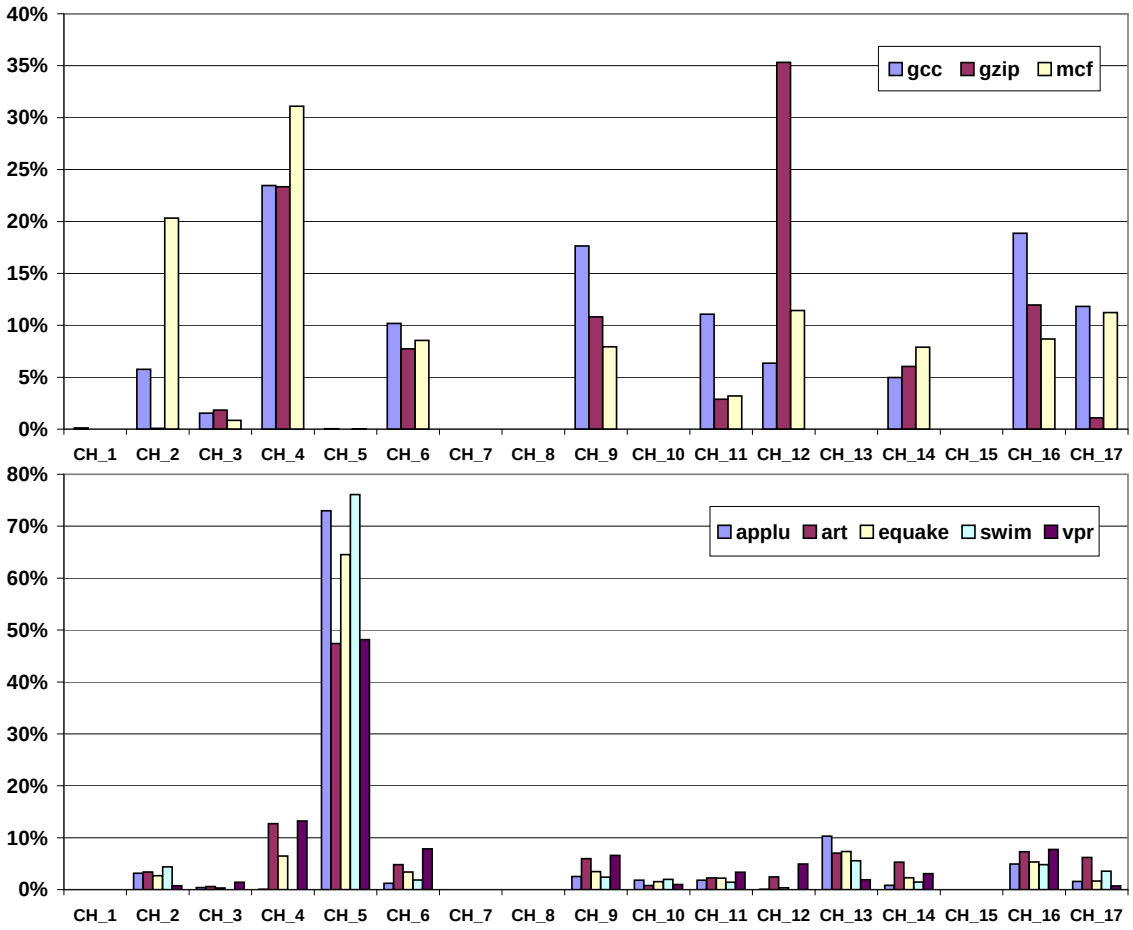


Figure 5. Percentage of synchronization penalties occurred in each individual channel as a percentage of the total synchronization penalty incurred.

4 and 5 for the return of cache data to the integer and floating-point register files, respectively. The impact of synchronization on Channels 4 and 6 has a particularly acute impact on *gzip*, due to its very small memory footprint that fits into the L1 data cache. The result is that the synchronization penalty as a fraction of the average load latency is highest for *gzip*, and therefore has the largest performance impact. An obvious way to greatly reduce the MCD synchronization penalty is to merge the integer and memory domains into a single combined domain. This has the added benefit of simplifying the scheduling of integer load-dependent instructions.

Recall that Channels 16 and 17 are the extra synchronization channels introduced by separating the ROB and L2 cache. Figure 6 shows that, although not zero, the performance impact of these penalties are relatively small compared with those on Channels 3, 4, 5 and 6. There is therefore the potential for significant gains in moving the ROB and L2 cache.

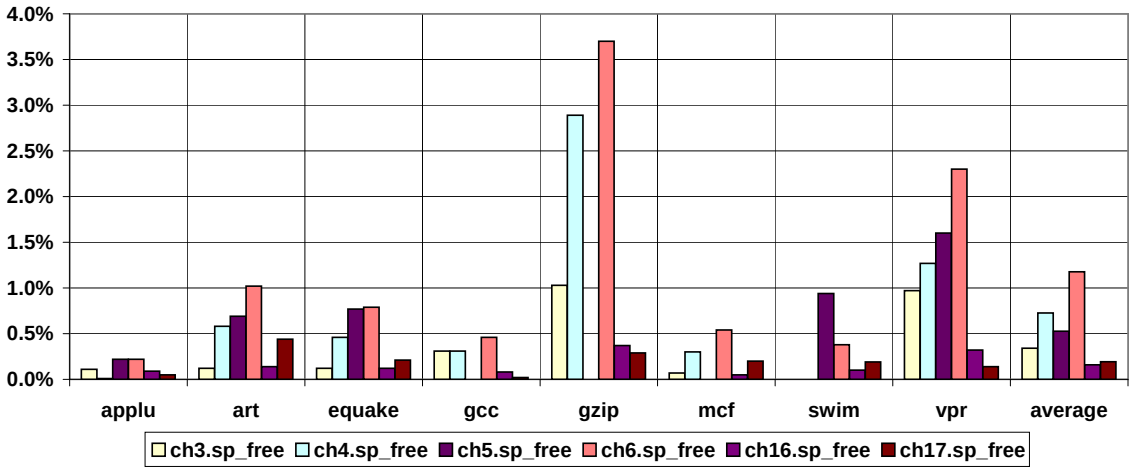


Figure 6. Performance improvement (compared to a realistic MCD processor) when channel x's penalties are removed (*chx.sp_free*).

4.3. Dynamically scaling the front-end frequency and separating the L2 cache

The MCD designers found that the frequency of the front-end could not be scaled without incurring a large performance degradation [12]. This indicates that a hardware unit within this domain always lies in the critical path of program execution. We conjecture that this critical structure is the Reorder Buffer (ROB) for the following reasons. The ROB connects the front-end and back-end of the execution pipeline. When an instruction is dispatched from the front-end, a free ROB entry needs to be allocated; when an instruction finishes execution and is ready to commit, the ROB needs to commit the architectural state changes made by this instruction. Since at any moment the execution bottleneck must be in either the front-end or the back-end, the ROB is always the key structure that needs to be run at the full frequency to maintain high performance.

Based on this conjecture, we propose to remove the ROB from the *front-end* domain and make it a new standalone domain always running at the full frequency. We have already observed in the prior section that doing so should have little performance impact. The effect on energy of not being able to scale the ROB should be small as well, as it comprises about 1% of the total chip power [2]. Thus, we create a five domain MCD processor called *5d* in the following discussion.

Figure 7 shows the frequency curves in the *front-end* domain before and after separating the ROB out, for *Swim*, using the offline algorithm. We find that, while the frequency curves are almost identical in the other domains (due to space limit, we did not include the frequency curves for the other domains here), more opportunities are exposed for scaling the frequency of the *front-end* domain with the separation of the ROB. Thus, the simple design change of making the ROB a full-frequency standalone domain should yield even greater MCD savings. Although we do not evaluate the energy savings of removing the L2 from the memory domain (as it uses a different energy saving approach than MCD), the prior section showed that the performance impact would be minor. We call the resulting six domain MCD processor *6d*.

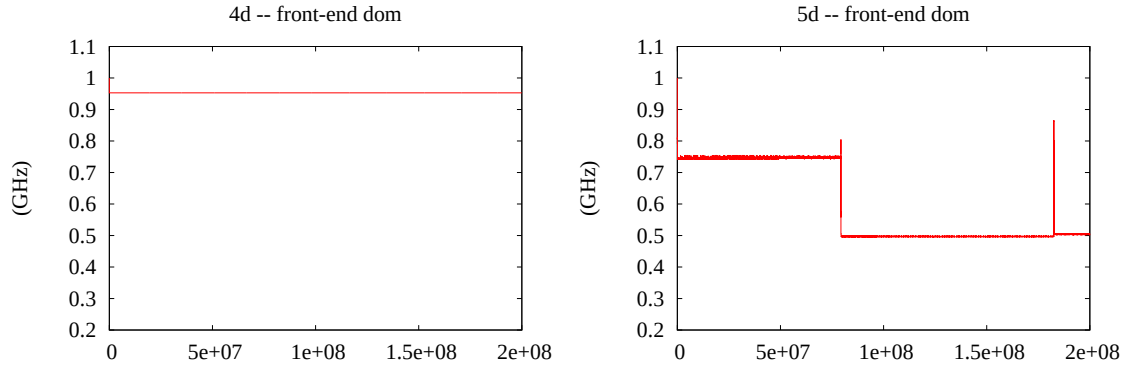


Figure 7. The frequency curves in the *front-end* domain, before and after separating the ROB out, for *Swim*, with the offline algorithm.

Figure 8 compares the effects of separating out the ROB and L2 cache using the modified online algorithm. The overall performance impact as expected from prior results is small in both cases, and is somewhat impacted by the online algorithm taking advantage of the ability to scale the front-end domain. As expected, the energy savings improves with the separate ROB, but to a small degree. This is due to the fact that, in our simulation model, the power consumption of the front-end is only about 12% of the total chip power. Thus, the new algorithm is recouping a large fraction of this potential savings. The potential impact is higher for designs that have greater front-end power. Finally, there is a small performance and energy cost for separating out the L2 cache, but this would be easily overridden by the ability to use thick oxides and high-V_t transistors.

4.4. Merging the integer and memory domains

Recall that we have three major motivations for exploring the merging of the integer and memory domains. First, we found that the channels between these domains that involve cache access were those that had the highest contribution to the MCD performance degradation. Second, removing those channels reduces the number of synchronization circuits. Finally, merging the domains removes a major source of MCD complexity: the issue complications of scheduling dependent instruction base on a load that makes two domain crossings and is therefore non-deterministic.

Figure 9 shows the effects of merging these domains using the modified online algorithm. Note that the online algorithm needs to monitor queue occupancy to make reconfiguration decisions. In the merged domain, we have two queues – the integer and load store queues. In our experiment we formed the new queue occupancy by adding the two occupancies together with each weighted by 0.5. We experimented with weightings between 0 and 1 and found that equally accounting for both queues gave the best result. As is shown in Figure 9, merging greatly reduces the performance degradation as expected from our prior results, with a small enough reduction on energy savings to maintain a similar energy delay product. Note that the worst case performance degradation for these benchmarks is reduced from 6.5% to 4.7%. With the obvious impact on the MCD design, merging the domains is a very complexity-effective design decision.

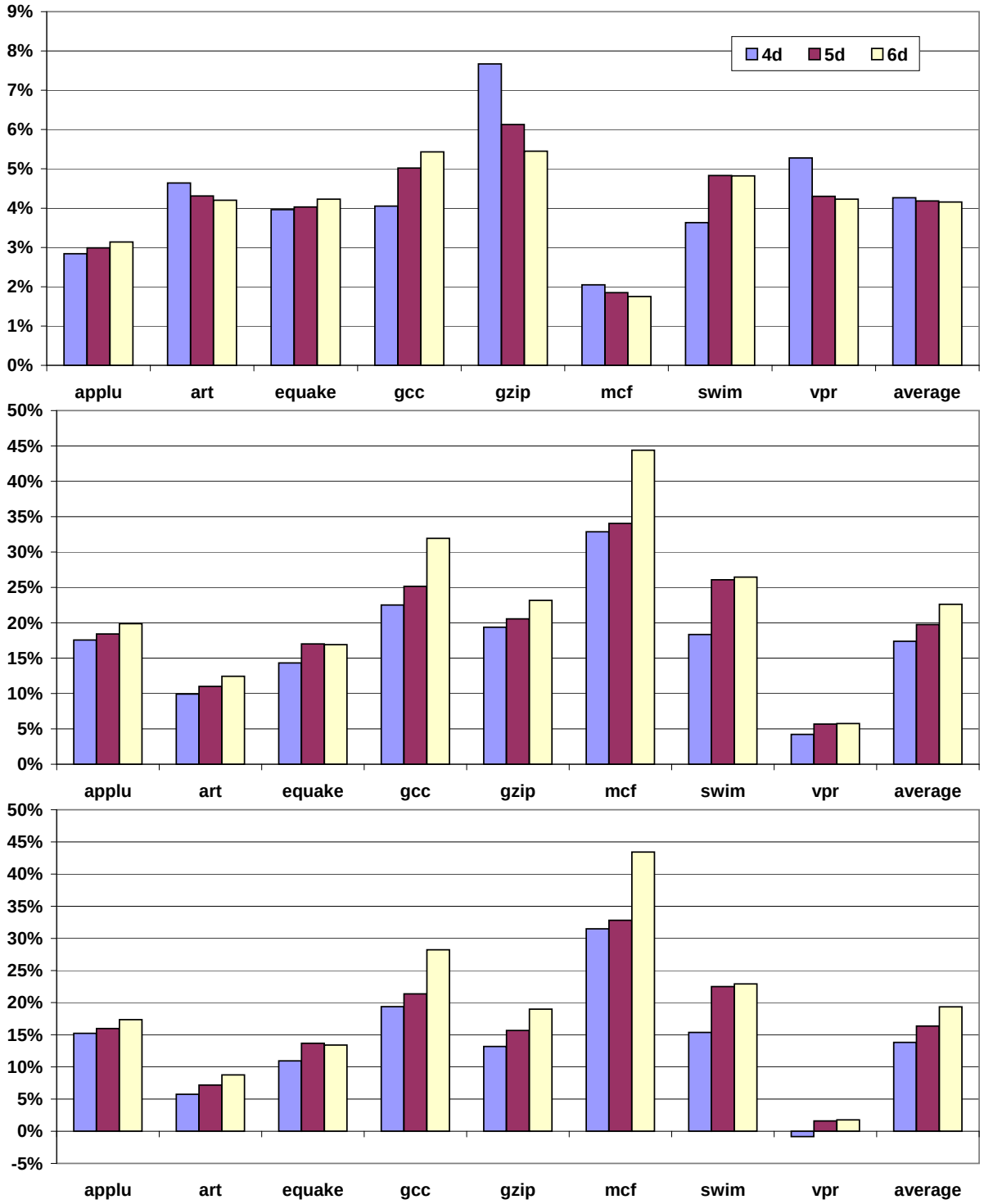


Figure 8. Performance degradation (upper), energy savings (middle) and energy delay product improvement (bottom), for 4, 5 and 6-domain MCD schemes, using the modified (Case M) attack/decay algorithm. The baseline is a fully synchronous machine with the same architectural parameters.

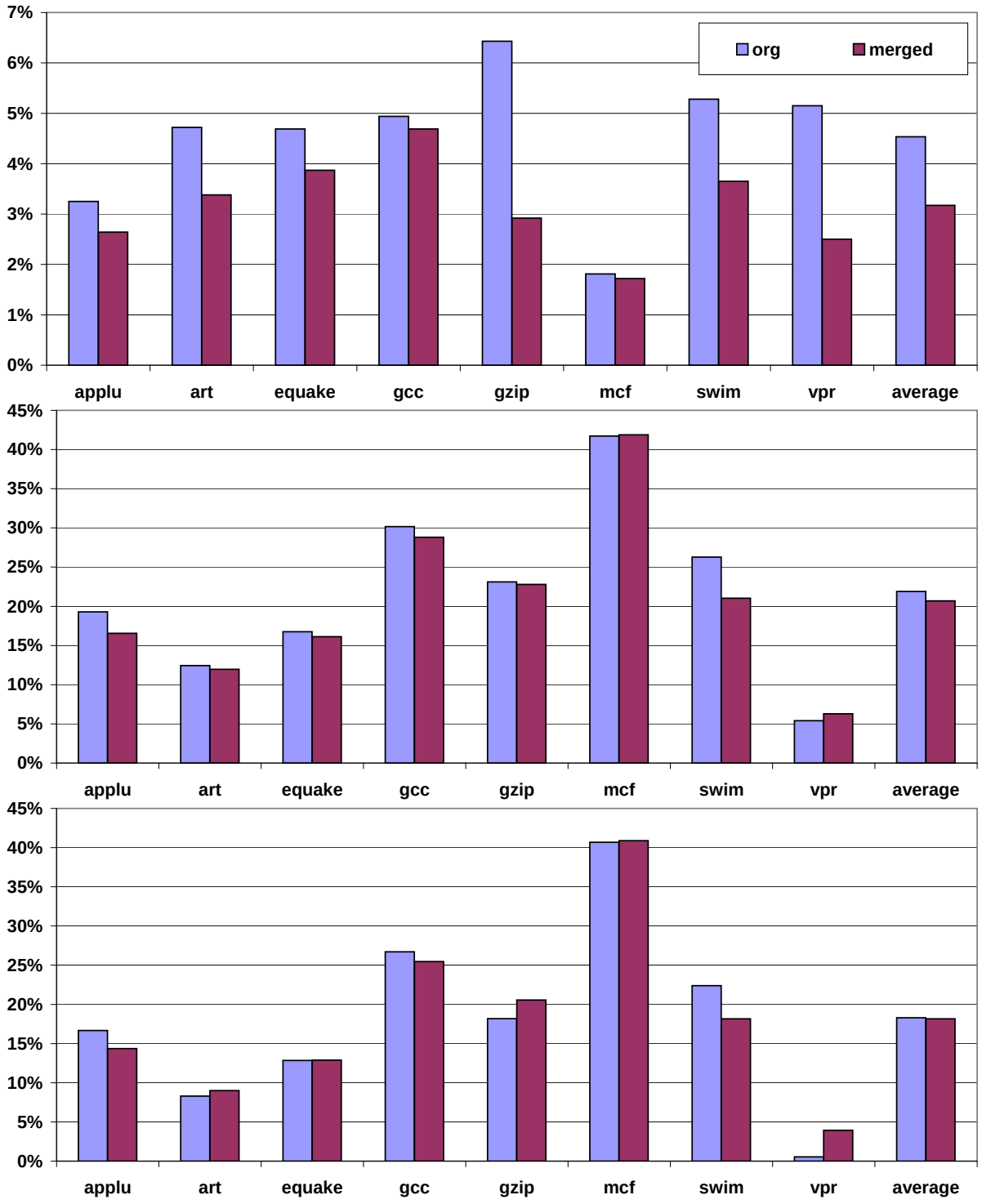


Figure 9. Performance degradation (upper), energy savings (middle) and energy delay product improvement (bottom), for 6-domain MCD schemes, before (*org*) and after (*merged*) merging the integer and memory domains, using the modified online algorithm. The baseline is a fully synchronous machine with the same architectural parameters.

Based on the analysis above, we propose to partition the chip into the following 5 domains:

- *ROB domain* — always running at full frequency;
- *L2 cache domain* — always running at half frequency;
- *fetch domain* — same as the original MCD design, except ROB;
- *floating-point domain* — same as the original MCD design;
- *integer and memory domain* — same as combining the integer and memory domains in the original MCD design, except the L2 cache.

4.5. Simpler frequency scheme

The original MCD design uses 32 different frequencies evenly distributed from 250 MHz to 1.0 GHz – which requires a complicated scheme to generate and switch among these 32 frequency levels, as well as the accompanying voltages. In this section, we study how much energy efficiency would be compromised by simpler, more easily implementable schemes, requiring only a few frequencies/voltages.

Assuming all 32 frequencies are equally important, we conducted experiments using 7 evenly-spaced frequencies (from 250 MHz, with step of 125 MHz, all the way up to 1.0 GHz). However, we find that the 32 frequencies are not equally important. For frequencies that are below 500 MHz, only the lowest (250 MHz) is heavily used when there are no activities in one domain (like the floating-point domain when an integer benchmark is running); the other low frequency values are seldom used. Based on this observation, we then conducted experiments on using only 4 frequencies: 250 MHz, 500 MHz, 750 MHz and 1.0 GHz.

Figure 10 compares the effects of using full-range frequency and using a fewer number. A 4-frequency scheme achieves much the same energy efficiency as the full-frequency scheme, without seriously degrading performance. As expected, the energy savings is reduced as the algorithm identifies less opportunity for energy savings. However, such a small number of frequencies and voltages could be generated using simple switches as with drowsy caches [6], making the design much easier to realize.

Two frequencies, however, are not enough to yield significant improvements in both performance and energy. The performance degradation is reduced, but this is only because the offline algorithm is unable to find many opportunities for energy savings with only two frequencies. For *vpr*, *applu* and *equake*, the energy-delay product is even worse than that of the full frequency scheme.

4.6. Putting it all together

Figure 11 compares the original MCD with the online algorithm of [10] with our modified MCD that separates the ROB and L2, combines the integer and floating point domains, uses four frequencies in each of the three domains (front-end, int+mem, and floating point), and the modified online algorithm. The overall performance degradation is reduced from roughly 2.9% to 2.5%, and the energy savings increases from 10% to 18%, with our proposed modifications. The performance degradation to energy (*not* power) savings ratio of the new scheme

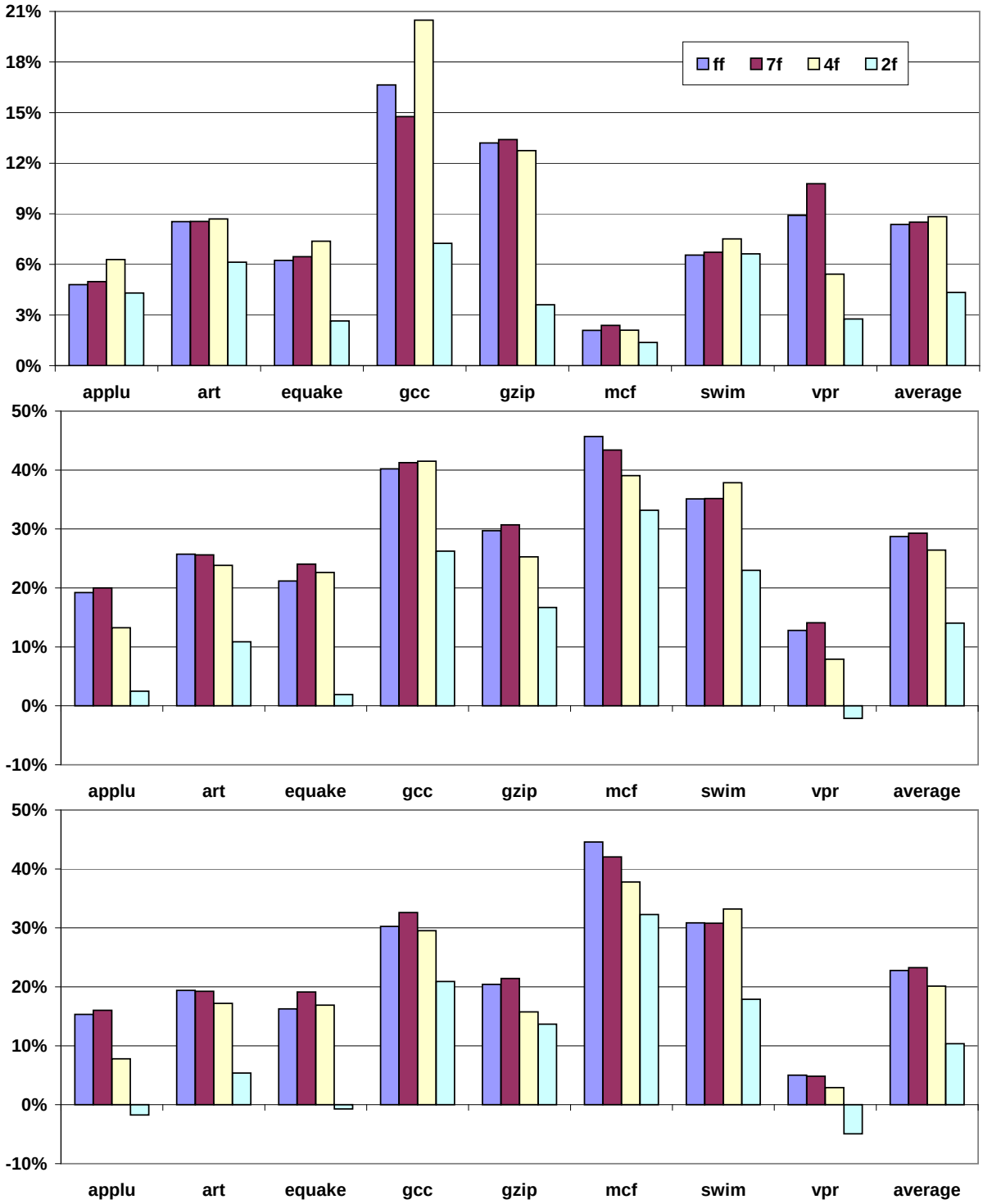


Figure 10. Performance degradation (upper), energy savings (middle) and energy x delay product improvement (bottom), for 6-domain MCD schemes, with the offline algorithm, using full range frequency and simpler frequency schemes. The baseline is a fully synchronous machine with the same architectural parameters.

Figure 11. Performance degradation (upper), energy savings (middle) and energy x delay improvement (bottom), over a fully synchronous machine with the same architectural parameters. *org* stands for the original MCD design with the original online algorithm using full-frequency scheme; *mod* stands for the new domain partition scheme with our modified online algorithm and four frequencies/voltages.

is a very favorable 7.2. Furthermore, our approach requires fewer synchronization circuits due to merging two domains, much simpler dynamic frequency and voltage scaling circuitry (only four levels), and a simpler integer instruction scheduler. Moreover, the modest changes that we proposed to the MCD microarchitecture (moving the ROB and modifying the online algorithm) have yielded significant benefits.

5. Related Work

A GALS approach to processor microarchitecture was proposed by Semeraro et al. in [12] and Iyer and Marculescu in [7]. Our work is built on the design presented in [12]; we propose a new domain partition scheme that achieves the same energy efficiency with lower performance degradation.

The performance impact of synchronization has been studied in [12, 7], and it has been demonstrated that a MCD processor with an out-of-order superscalar core would be more capable of tolerating the performance effects of synchronization penalties than with an in-order issue core [11]. While these works studied the overall impact of *all* the synchronization penalties occurred in *all* synchronization channels, our work identified the most important channels where the penalties occurred would have the biggest performance impact (though their amount is not necessarily the highest). We proposed to remove those penalties by domain merging and showed the new domain partition scheme achieves essentially same energy savings but with lower performance degradation, compared to the original design.

A number of schemes have been devised to control a MCD processor. In [12], an offline algorithm is proposed. It analyzes program trace built from execution on a fully synchronous machine and makes decisions on when and to what value the frequency should change. Though not practical in some application environments, it sets an optimal limit against which more practical schemes can be targeted. We used this algorithm in our work to see the effects of our modified algorithm and to test the effects of implementing fewer frequencies. Magklis et al. proposed a more practical profile-based control scheme [8], which applies the offline algorithm only to those "important" procedures and loop nests identified by profiling runs. These two approaches are both software-based. The first hardware-based approach—the attack/decay algorithm is proposed by Semeraro et al. in [10]. In Section 4.1, we analyzed the potential problems with this algorithm and proposed modifications which better bound the performance degradation while saving additional energy. Another hardware-based approach was recently proposed by Wu et al. [13]. They use a PI controller to make reconfiguration decisions. The input to the controller is the difference between the previous interval queue occupancies and the *pre-specified* target queue occupancy. The pre-publication version of this paper did not appear in enough time for us to make a comparison with our own improved algorithm. However, in addition to a new control algorithm, we propose a number of additional modifications to the MCD microarchitecture that improve its effectiveness while simplifying the overall design.

6. Conclusions

In this paper, we presented our work towards achieving a more complexity-effective MCD design. We first presented our modification to the original attack/decay algorithm, which remedies the problem of high performance degradation in some cases of the original algorithm and achieves better energy efficiency, while adding

little complexity to the control circuitry. We then studied the domain partition problem and proposed to separate the ROB and L2 cache, which results in a dynamically scalable front-end domain, and the freedom to use standard L2 cache low-power techniques. Based on the new domain partition scheme, we studied the performance sensitivity to synchronization penalties on every channel and found that, the channel involving cache access are the ones for which performance is the most sensitive. Based on this observation we proposed to merge the integer and memory domains to remove these hot channels. Finally, we found that a simpler 4-frequency scheme reaps most of the energy efficiency that could be achieved with the 32-frequency scheme. A 4-frequency designing greatly simplifies the implementation of the frequency and voltage controller. We finally compared our new MCD design (new domain partitioning, the modified online algorithm and a simpler 4-frequency scheme) with the original MCD design and found that, the new approach improves both performance and energy efficiency. The net result of all these optimizations is a design with more compelling performance and energy results yet which is much more implementable as compared to the original MCD design.

References

[1] P. Bose, D. H. Albonesi, and D. Marculescu. Complexity-Effective Design. Available at http://www.ece.rochester.edu/users/albonesi/wced03/review_article.html, 2003.

[2] P. Bose, D. Brooks, A. Buyuktosunoglu, P. W. Cook, K. Das, P. G. Emma, M. Gschwind, H. Jacobson, T. Karkhanis, P. Kudva, S. Schuster, J. Smith, V. Srinivasan, V. V. Zyuban, D. H. Albonesi, and S. Dwarkadas. Early Stage Definition of LPX: A Low Power Issue-Execute Processor. In *Proc., 2nd Intl. Workshop on Power-Aware Computer Systems*, 2002.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level analysis and optimization. In *Proc., 27th Intl. Symp. on Computer Architecture*, 2000.

[4] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, Dept. of Computer Science, University of Wisconsin, 1997.

[5] A. Chakraborty and M. R. Greenstreet. Efficient Self-Timed Interfaces for Crossing Clock Domains. In *Proc., 9th Intl. Symp. on Asynchronous Circuits and Systems*, 2003.

[6] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proc., 29th Intl. Symp. on Computer Architecture*, 2002.

[7] A. Iyer and D. Marculescu. Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors. In *Proc., 29th Intl. Symp. on Computer Architecture*, 2002.

[8] 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 processor. In *Proc., 30th Intl. Symp. on Computer Architecture*, 2003.

[9] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *Proc., 24th Intl. Symp. on Computer Architecture*, 1997.

[10] 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 *Proc., 35th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, 2002.

[11] G. Semeraro, D. H. Albonesi, G. Magklis, M. L. Scott, S. G. Dropsho, and S. Dwarkadas. Hiding synchronization delays in a gals processor microarchitecture. In *Proc., 10th Intl. Symp. on Asynchronous Circuits and Systems*, 2004.

[12] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proc., 8th Intl. Symp. on High-Performance Computer Architecture*, 2002.

[13] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *Proc., 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2004.