Providing Predictable Performance vs. Energy/Temperature Tradeoffs for Multimedia Applications [*]

Christopher J. Hughes, Ruchira Sasanka, Jayanth Srinivasan, and Sarita V. Adve
Department of Computer Science
University of Illinois at Urbana-Champaign
{cjhughes,sasanka,srinivsn,sadve}@cs.uiuc.edu

Abstract

Multimedia applications are an increasingly important workload for general-purpose processors on a growing number of systems (e.g., future handheld, laptop, and desktop systems). These systems often operate in energy and/or thermally constrained environments. Recently, a great deal of work has been done on dynamic energy and thermal management for general-purpose processors through processor hardware adaptation. These techniques save energy and allow designing for thermal limits that are lower than the possible peak temperature, but often at the cost of lower performance. Inherent to these techniques is the ability to operate at different points in the energy/temperature vs. performance tradeoff spectrum. There has been much work showing that such techniques result in high energy savings and better thermal efficiency. For multimedia applications, however, performance predictability is a crucial requirement, to ensure that their soft real-time constraints will be met and to exploit opportunities afforded by these applications to trade off performance for resource use.

This paper revisits energy and thermal management algorithms from the viewpoint of predictability. Viewed in this context, we show that a class of algorithms, which we call predictive algorithms, are able to provide predictable energy/temperature vs. performance tradeoffs within a reasonable tolerance. In contrast, a more widely discussed class of algorithms (in the context of general-purpose computing), which we call reactive algorithms, were not designed for predictability. In terms of absolute energy savings, predictive algorithms have some advantages, but a combination of predictive and reactive algorithms does best – while this algorithm appears to provide good predictability as well, it requires a lot of off-line tuning. For thermal efficiency, predictive algorithms always showed lower performance degradation than the reactive algorithms in all cases studied.

1 Introduction

Multimedia applications are an increasingly important workload for a growing number of systems (e.g., future handheld, laptop, and desktop systems). As these applications become more complex and follow evolving standards, considerations such as programmability, upgradability, and portability become important. As

[*]This work is supported in part by an equipment donation from AMD, a gift from Motorola Inc., the National Science Foundation under Grant No. CCR-0096126, CCR-0209198, CCR-0205638, and EIA-0224453, and the University of Illinois. Sarita V. Adve is also supported by an Alfred P. Sloan Research Fellowship and Christopher J. Hughes was supported by an Intel graduate fellowship.

a result, general-purpose programmable architectures become increasingly attractive over special-purpose designs. However, multimedia applications impose many new challenges for general-purpose processors, including the need for high energy and thermal efficiency and high predictability.

Multimedia applications often run on mobile and small form factor devices, demanding high energy and thermal efficiency. Mobile devices generally use a battery for their power supply and thus have a limit on the energy they can consume, often requiring a tradeoff in performance. Their relatively small size and cost additionally limit the ability to cool the processor; e.g., requiring cheaper packaging, heat sinks, and no fans. The result is a reduced maximum thermal limit for the processor (i.e., the temperature that the processor must not exceed), again at the cost of performance.

Recently, a great deal of research has been done on improving energy and thermal efficiency for general-purpose processors through processor hardware adaptation. For energy, a key observation made is that often slowing down processing can reduce processor energy, albeit at the cost of performance. Further, processor resources are often under-utilized in different parts of the execution; deactivating such resources can save energy without affecting performance. Thus, depending on current resource usage and application performance requirements, the processor hardware is dynamically adapted to acceptably trade off energy and performance.

To reduce thermal costs, the key observation made is that the peak temperature is very rarely reached by a processor and is much higher than the typical temperature. Therefore, the processor and consequent thermal solutions are designed for a thermal limit lower than the peak temperature – if during execution, the processor appears to reach this thermal limit, a hardware adaptation is triggered to reduce the temperature, generally at the expense of performance.

For both energy and thermal management, most processor adaptations fall in one of two categories: dynamic voltage and frequency scaling (DVS) or architecture adaptation. DVS reduces the voltage of the processor to save energy, but requires a corresponding decrease in frequency, which degrades performance. Architecture adaptation changes the amount of active microarchitectural resources on the processor (e.g., ALUs) by powering down (deactivating) some of them. The effects of architecture adaptation on energy, temperature, and performance depend on how the adaptation is controlled or the *adaptation control algorithm*.

A large number of adaptation control algorithms have been proposed and shown to be effective at saving energy and reducing thermal costs for both multimedia and other general-purpose workloads. However, a critial distinguishing feature of multimedia applications is that they require high predictability. These are soft real-time periodic applications, where at each fixed time interval (called the period), a new discrete piece of data, typically called a *frame*, becomes available for processing. This processing, called a *job*, must complete before the start of the next period. Thus, real-time operating systems must carefully schedule these applications, allocating a fixed amount of time for each job, so that all jobs can finish within the required time. The scheduling algorithms are often accompanied by admission control algorithms, which determine if a newly arrived application's jobs can be successfully scheduled; if not, the application is denied admission into the real-time partition of the system. These systems, therefore, require an priori prediction of the execution time for each job. Since these applications are soft (vs. hard) real-time, it is acceptable to miss a small fraction of the deadlines; therefore, while a high level of execution time predictability is desired, it does not have to be perfect.

As energy becomes increasingly important, real-time operating systems have begun to consider energy as a first-class resource and begun to schedule energy as well, requiring predictability of energy consumption. The use of adaptive systems has made predictability important in yet another way, by opening up a new space of trading off performance for energy or temperature. Multimedia applications can be run with lower performance (which allows energy savings) without affecting the user experience, as long as the real-time constraint is met. Furthermore, multimedia applications offer the opportunity to operate at different quality points with different resource usage. These traits enable multimedia applications to exploit tradeoffs in performance and energy in an intelligent way. As more layers of the system become adaptive, (e.g., , network and scheduler), the possible ways in which energy and performance tradeoffs can be made increases and the operating system must decide how to make these tradeoffs for the multiple applications in the system.[1] Making these decisions in an intelligent way requires high predictability of such energy vs. performance tradeoffs.

Unfortunately, work on general-purpose processors has not paid much heed to the issue of predictability. On the contrary, many performance enhancing techniques (e.g., out-of-order and speculative execution) appear to reduce predictability, and many adpatation control algorithms for energy and thermal management are oblivious to this issue.

In this paper, we revisit hardware adaptation control algorithms in the context of predictability for multimedia applications. Viewed in this context, we divide the space of adaptation control algorithms into reactive algorithms and predictive algorithms. A majority of the algorithms proposed in the literature for architecture adaptation are reactive algorithms – they invoke adaptations at a fine granularity (order of a few hundred instruction long intervals), assuming that the behavior in the previous interval is representative of that in the next interval. These algorithms typically do not make any predictions or guarantees of time or energy since they do not a priori predict the behavior of the execution. In contrast, our previous work has shown that many features of multimedia applications can be predicted at a frame granularity. The other class of algorithms, predictive algorithms, exploit this property by working on a frame granularity. Previous work has shown that these predictive algorithms are often better at energy and temperature management than reactive algorithms because the frame granularity allows them to exploit high overhead adaptations and the predictability of frame properties allows the development of these algorithms without much application specific tuning. This paper shows that in addition to the above advantages for energy and temperature man-

[1]Another paper submitted to this special issue deals with such a cross-layer adaptive system called GRACE [13], and uses the results of this paper.

agement, the predictive algorithms additionally allow a priori predictions of tradeoffs between execution time, energy, and temperature, a key requirement for real-time operating systems.

2 Adaptation Control for Real-Time Multimedia Applications

As mentioned earlier, in this paper, we consider periodic soft real-time multimedia applications. We assume that the real-time operating system allocates a certain amount of processing time for each frame (the same amount for each frame). The allocation depends upon the application's needs, the load on the system, and perhaps predicted energy-performance or thermal-performance tradeoffs provided by the adaptation control algorithm; therefore, the allocation is unknown at application development time. Each frame must be processed within its processor allocation, which we henceforth refer to as the *deadline*. (Note that this is a slightly different meaning from the usual real-time notion of a deadline, which would be the period of the application.) We assume soft real-time applications, for which a small fraction of deadline misses is acceptable. Finally, for some frames, the processor may finish before the deadline. We refer to the leftover processing time as *slack*. If there is slack for a frame, we can slow the processor down without the user noticing, as long as we do not use more than the available slack.

Several important characteristics of the hardware behavior when running these applications are easily predictable at the granularity of a frame: performance, energy consumption, and peak temperature reached. Therefore, we have previously proposed algorithms that adapt at the frame granularity [7, 10, 12]. For each frame, our algorithms use accurate predictions about the behavior of the different possible hardware configurations to choose the one that maximizes some desired property (e.g., energy savings) while still meeting certain other requirements (e.g., meeting real-time constraints). We call this approach to adaptation control *predictive*.

Previous algorithms for controlling DVS alone exploit slack for real-time multimedia applications; they are also predictive and run at the frame granularity. However, most previous algorithms for controlling architecture adaptation use a *reactive* approach and adapt at a sub-frame granularity (at regular intervals during a frame). We refer to sub-frame adaptation as temporally local, as opposed to frame-level, or temporally global, adaptation. The reactive algorithms do not rely on accurate predictions of hardware behavior since obtaining such predictions is difficult for general applications. Instead, they assume that hardware behavior changes little from one interval of a frame to the next. These algorithms monitor certain hardware behavior and choose a configuration for the next interval frame based on a set of heuristics. These heuristics generally require a large amount of tuning effort, which can make them hard to implement practically. They also can give undesirable behavior for applications outside of the set used for tuning.

Another approach is to use an algorithm grounded in control theory. Such an algorithm targeting thermal efficiency has been proposed for controlling a single architecture adaptation [11]. While this approach is very promising, we do not yet know of work showing how to control multiple, interacting architecture adaptations and DVS together using a control theoretic technique.

Besides the predictability issue explored here, our predictive adaptation control algorithms have two advantages over the best reactive ones. First, the predictive algorithms allow the use of adaptations with large switching times, such as DVS. Reactive algorithms, which are inherently temporally local, adapt too frequently to control adaptations like DVS (which can take up to 10μs to switch [4]). Second, the predictive algorithms require less tuning than the reactive algorithms.

Reactive algorithms do have an advantage over our predictive algorithms: they can exploit variability in hardware behavior during a frame due to their temporally local nature. To combine the benefits of both predictive and reactive algorithms, we have previously proposed a combined predictive+reactive algorithm [10] (described later).[2]

Section 3 describes the best reactive algorithms targeting energy and thermal efficiency evaluated for multimedia applications, and Section 4 gives our predictive algorithms. Section 5 describes our experimental methodology. Section 6.1 demonstrates our predictive algorithms' ability to predict the energy-performance or temperature-performance tradeoffs they provide. Finally, Section 6.2 compares the energy and thermal efficiency of our predictive algorithms to that of the best reactive algorithms.

3 Reactive Adaptation Control

3.1 Reactive Control for Energy

A number of reactive algorithms to control individual adaptive architectural resources have been proposed. Adaptations considered include changing the instruction window size (or issue queue and reorder buffer sizes) [2, 3, 10], changing the number of functional units and/or issue width [1, 9, 10], changing the amount of speculation allowed [8], and others. In this paper, we focus on changing the instruction window size and the number of active functional units (and issue width). We assume that the instruction window is composed of equal-sized segments, and that an arbitrary number can be active at any time. We also assume that an arbitrary number of ALUs and FPUs can be active at any time.

Most reactive control algorithms for these resources use two heuristics: one each to indicate if some of the resource should be deactivated or reactivated. Previously, we evaluated a number of reactive heuristics, and proposed some of our own, for these two architecture resources [10]. The heuristics all consider adapting at the end of small, fixed intervals (256 cycles here), and deactivate or reactivate only the smallest part of a resource at each interval boundary (e.g., one ALU). We now describe the combination of heuristics for each resource that provides the most energy savings for the applications studied.

To decide whether to increase the size of the instruction window, we proposed a heuristic that estimates

[2]Purely predictive algorithms, however, are not inherently temporally global. We have concurrently developed a (more complex) predictive algorithm which also exploits variability during a frame [5]. This work is concurrently under review, so we do not discuss that algorithm here.

the performance loss from having some of the window deactivated (the details of the elaborate estimation method are not important for this paper). If the performance loss in an interval is larger than a *threshold*, the window size is increased for the next interval. To decide whether to decrease the size, Folegnani et al. proposed a heuristic that counts the number of instructions issued from the youngest part of the window [3]. If this is below a *threshold* for an interval, there is likely little benefit from having so large a window, and its size is decreased for the next interval.

To decide whether to increase the number of active functional units, we proposed a heuristic that counts the number of times instructions are ready to be issued to a functional unit, but no unit is free (i.e., issue hazards). The larger the number of issue hazards, the larger the performance loss from deactivated units (most likely). If the number of issue hazards is above a *threshold* for an interval, the number of active units is increased. To decide whether to decrease the number of active units, Maro et al. proposed a heuristic that counts the number of cycles all units are utilized [9]. If this number is smaller than a *threshold* for an interval, there is likely little benefit from having so many units active, and the number of units is decreased for the next interval.

All four of the heuristics described above collect a statistic during each interval and compare that to a threshold at the end of the interval to decide whether or not to adapt. These thresholds must be carefully tuned to achieve the best tradeoff between performance and energy. However, it is difficult to tune even a single threshold since the behavior of the heuristics is application-dependent – one value of the threshold may work well for some applications, but poorly for others. Even worse, adaptation of the resources considered here will interact, making it desirable to tune thresholds together. This makes the number of design points (combinations of the threshold values) explode. We attempted to automate the tuning process, but with little success. Instead, we rely on manual tuning, which, given the large design space, takes an enormous amount of effort. However, even though we do find a good set of thresholds this way, we use only a small set of applications for the process. We cannot easily predict the behavior of the heuristics with the thresholds we choose for applications outside our set. This issue is explored in more detail in [5].

We call the combination of the above two algorithms *reactive*.

3.2 Reactive Control for Thermal Efficiency

In our simulations, the register file was the hottest structure for all of our applications. We find that the reactive adaptations used by the reactive energy algorithm – instruction window adaptation and functional unit adaptation – are both effective in handling the register file hot spot. Deactivating functional units targets the register file temperature directly by reducing the number of active register file ports. Instruction window adaptation indirectly impacts the register file temperature by reducing the number of instructions in the processor pipeline which results in fewer register file accesses. We found that these responses were more effective than the previous state-of-the-art of processor toggling [12].

Our reactive thermal algorithm works by deactivating a fixed amount of the functional units or the instruction window whenever the processor temperature exceeds a set thermal trigger temperature [12]. Once the processor temperature goes below the thermal trigger, the functional units and instruction window are fully powered up again.

Finally, different response mechanisms used on different applications require different thermal trigger temperatures to ensure safe and efficient operation. A fixed thermal trigger for all purposes will either result in a loss of performance for many applications (due to the trigger temperature being too conservative) or will put some applications in thermal emergency (due to the trigger temperature being too aggressive). For the fairest comparison of our reactive thermal algorithm against our predictive thermal algorithm, we manually tune the trigger temperature used for each combination of application and thermal limit to ensure the best possible performance that was also thermally safe.

4 Predictive Adaptation Control

In this section we describe our previously proposed predictive algorithms targeting energy efficiency and thermal efficiency (one algorithm for each) [7, 12]. Both are based on the ability to predict certain behavior of the different hardware configurations (power, performance, and maximum temperature) at the frame level. Our algorithm targeting energy efficiency simply picks the lowest energy configuration that will meet the deadline. Our algorithm targeting thermal efficiency simply picks the highest performance configuration that will not exceed the thermal limit. We first discuss how we make the power, performance, and temperature predictions and then present the algorithms in turn. We also present an algorithm that combines predictive and reactive control and discuss how to apply energy and thermal control together.

4.1 Predictability

A common conjecture made about general-purpose processors is that their use of complex features (e.g., out-of-order issue) results in unpredictable execution times, making them unsuitable for real-time multimedia applications. Previously we tested this conjecture by examining execution time variability at the frame granularity for several multimedia applications [6]. We found that while there is often some variability, it is mostly caused by the application algorithm and input [6]. In contrast to conventional wisdom, aggressive architectural features induce little additional variability, and thus little unpredictability. The intuition behind this finding is that the nature of the work done for each frame is the same, even if the amount of work varies across frames (e.g., more work is done to encode video frames with a lot of motion).

Related to this finding, we found that some key aspects of the processor's behavior vary little at the frame granularity when running these applications (given a fixed configuration of the processor). In particular, three metrics are almost constant across all frames of the same type:[3] (1) the rate at which instructions are

[3]Two applications in our suite have multiple frame types (i.e., I, P, and B frames for MPEG-2 encoder and decoder). For these, we treat each type separately.

executed (IPC, or instructions executed per cycle), (2) the average power dissipation, and (3) the maximum temperature reached at the hottest part of the chip [6, 7, 12]. We can run a single frame with a given hardware configuration and measure the above quantities, then use the measurements as predictions for all other frames of that type for that configuration. Additionally, while the number of instructions executed per frame may vary significantly over the course of the application, it changes slowly; thus, for each frame we can predict it using the values from recent frames [6]. We derive adaptation control algorithms based on these prediction techniques.

4.2 Predictive Control for Energy Efficiency

At the beginning of each frame, our algorithm for energy efficiency predicts and chooses the hardware configuration that will minimize energy for that frame without missing the deadline. The algorithm can control both adaptive architecture resources and DVS. A *hardware configuration* is a combination of the configuration for all architecture resources and the voltage and frequency. Our algorithm chooses a single hardware configuration for each frame, thereby adapting all adaptive resources together.

The algorithm consists of two phases: a profiling phase at the start of the application and an adaptation phase. The profiling phase profiles one frame of each type for each architecture configuration A, at some base voltage and frequency. The algorithm collects the instructions per cycle (IPC_A) and average power (P_A) for each frame.

Since the average IPC and power for a configuration are roughly constant for all frames of a given type, the IPC_A and P_A values from the profiling phase are used to predict the IPC_A and P_A of all other frames of that type. Also, IPC is almost independent of frequency for the applications considered here (because they have little memory stall time), so IPC_A can be used to predict the IPC for architecture A at all frequencies. P_A and the voltage for each frequency can be used to predict the power for architecture A at all frequencies ($Power \propto V^2 f$).

For each hardware configuration, H, with architecture A, the algorithm computes the most instructions executable within the deadline (I_{max}) as $I_{max_H} = deadline \times IPC_A \times f_H$. It also computes the energy consumed per instruction (EPI) for each configuration as $EPI_H \propto \frac{P_A \times {V_H}^2}{IPC_A}$. It then constructs a table with an entry for each configuration containing its I_{max} and EPI, sorted in order of increasing EPI. Since the average IPC and power for a configuration are roughly constant, EPI also remains roughly constant across frames.

After profiling is complete, the algorithm enters the adaptation phase. For each frame, it predicts the number of instructions, using a simple history-based predictor (takes the maximum of the last five frames and adds some leeway). It then searches the table (starting at lowest EPI) for the first entry with $I_{max} \geq predicted\ instructions$. It predicts this to be the lowest energy configuration that will still meet the deadline, and so chooses it.

For processors that support a large number of frequency choices (e.g., Intel's XScale), the number of configurations may be too large to use this table-based approach for choosing the configuration. A variation of this algorithm for such processors is given in [7], but omitted here for lack of space.

We call this algorithm *predictive*.

4.2.1 Combining Predictive and Reactive for Energy Efficiency

We have also previously proposed a combined predictive+reactive algorithm, which was found to give better energy savings than either the predictive or reactive algorithm alone [10]. This algorithm can adapt during a frame, but inherits some of the predictive algorithm's prediction abilities and ability to control slow-acting adaptations. The reactive algorithms always run while predictive runs, including during both the profiling and adaptation phases. The predictive algorithm sets the maximum amount of each resource; the reactive algorithms cannot increase beyond these limits. Running the reactive algorithms during the profiling phase allows the predictive algorithm to account for the frame-level energy and performance impact of the reactive adaptations.

The predictive+reactive algorithm, however, suffers from the disadvantage of the reactive algorithms in that it requires significant tuning for the heuristics applied for the reactive part.

4.3 Predictive Control for Thermal Efficiency

Our predictive thermal algorithm attempts to determine the highest performing architectural configuration which is also thermally safe [12]. Unlike our predictive energy algorithm, which attempts to save energy by slowing down execution as much as possible within the application deadline, our predictive thermal algorithm seeks to execute applications as fast as possible without violating thermal limits.

For every architecture, A, the algorithm profiles the IPC, IPC_A, and the maximum temperature reached, ($T_{max|A,f_{prof}}$) by any structure on chip for an appropriate number of frames of each type. This profiling is performed at the lowest supported voltage, V_{prof}, and frequency, f_{prof}, in order to ensure that no thermal emergency occurs during profiling. The algorithm then determines the maximum frequency, $f_{max|A}$ at which each profiled architecture is still thermally safe using the expression

$$\frac{T_{limit}}{T_{max|A,f_{prof}}} = \frac{V^2_{max|A} f_{max|A}}{V^2_{prof} f_{prof}} \tag{1}$$

where $T_{max|A,f_{prof}}$ is the maximum temperature reached by any structure on chip during profiling, $V_{max|A}$ is the voltage required to support $f_{max|A}$, and T_{limit} is the chip's current thermal limit (which is the difference in temperature between the maximum allowed chip temperature and the current heat sink temperature[4]). The exact form of the above expression is determined by the thermal model used (for e.g., a different model will be used if leakage effects are accounted for).

[4]In the above formula, we actually use a temperature 0.2^{0}C lower than the actual thermal limit, to provide a small leeway for safety.

Base Processor Parameters	
Processor speed	1GHz
Fetch/retire rate	8 per cycle
Functional units	6 Int, 4 FP, 2 Add. gen.
Integer FU latencies	1/7/12 add/multiply/divide (pipelined)
FP FU latencies	4 default, 12 div. (all but div. pipelined)
Instruction window (reorder buffer) size	128 entries
Register file size	192 integer and 192 FP
Memory queue size	32 entries
Branch prediction	2KB bimodal agree, 32 entry RAS

Base Memory Hierarchy Parameters	
L1 (Data)	64KB, 2-way associative, 64B line, 2 ports, 12 MSHRs
L1 (Instr)	32KB, 2-way associative
L2 (Unified)	1MB, 4-way associative, 64B line, 1 port, 12 MSHRs
Main Memory	16B/cycle, 4-way interleaved
Base Contentionless Memory Latencies	
L1 (Data) hit time (on-chip)	2 cycles
L2 hit time (off-chip)	20 cycles
Main memory (off-chip)	102 cycles

Table 1: Base (default) system parameters.

If $f_{max|A}$ is greater than the largest supported frequency in the system, the largest supported frequency is used. If $f_{max|A}$ is lower than the lowest supported frequency, architecture A is discarded by the algorithm as it can not be run safely for the given thermal limit.

Since the performance of a hardware configuration is proportional to the product of its IPC and frequency, the maximum thermally safe performance of an architecture is proportional to $f_{max|A} \times IPC_A$. The algorithm therefore chooses the architecture, A, with the highest $f_{max|A} \times IPC_A$ product. This architecture running at $f_{max|A}$ is predicted to be the fastest thermally safe hardware configuration and is used in the rest of the run (for the corresponding frame type).

Our predictive thermal algorithm can exploit both architectural adaptation and DVS. Like all the other algorithms, the architectural adaptations available to the predictive thermal algorithm are instruction window resizing and functional unit adaptation. When the instruction window is resized, the predictive thermal algorithm also rescales the register file size. The reactive thermal algorithm is not allowed to do so because of the large time overhead involved with register file resizing.

The predictive thermal algorithm is described in detail in [12].

4.4 Combining Predictive Energy and Predictive Thermal Algorithms

Although we present results for predictive energy and predictive thermal algorithms separately, these two algorithms can be combined easily if we want to use them both on a system. In such a system, the profiling phase of the thermal algorithm should be run first to identify the thermally safe configurations. Then the energy algorithm can select the lowest energy configuration out of the available thermally safe configurations.

5 Experimental Methodology

The methodology used in this study is similar to that in our previous work [10, 12].

5.1 Systems Modeled

We use the RSIM simulator for performance evaluation and the Wattch tool integrated with RSIM for energy and power measurement. We derive temperature from power using the model described in [11]. The base, non-adaptive, processor studied is summarized in Table 1. We model a centralized instruction window with a unified reorder buffer and issue queue, and a separate physical register file.

The predictive, reactive and predictive+reactive algorithms for energy are explained in detail in [10] where we refer to them as *Global*, *Local*, and *Global+Local* respectively. The predictive and reactive algorithms for temperature control are explained in detail in [12]. The *R-IwFu* reactive algorithm in [12] is used as the reactive algorithm for this paper and the *P-ArchDVS* predictive algorithm is used as the predictive thermal algorithm in this paper. The following methodology applies to both energy and thermal algorithms. However, a predictive+reactive algorithm is not studied for temperature control (there does not exist one yet) and all references to predictive+reactive are only in the context of energy.

Predictive and predictive+reactive algorithms always use DVS and register file adaptation whereas reactive algorithms do not employ either due to high overhead. For the reactive algorithms, we change the size of the instruction window and the number of active functional units. For predictive and predictive+reactive algorithms, we profile all possible combinations of the following architecture configurations (54 total): instruction window size $\in \{128,96,64,48,32,16\}$, number of ALUs $\in \{6,4,2\}$, and number of FPUs $\in \{4,2,1\}$. Predictive+reactive, due to its incorporation of reactive, can choose any supported architecture configuration during a frame (i.e., it is not restricted to the 54 profiled ones). With both predictive and reactive algorithms, when we change the number of active functional units, we assume that the issue width is equal to the sum of all active functional units and hence changes with the number of active functional units. Consequently, when a functional unit is deactivated, the corresponding instruction selection logic is also deactivated. Similarly, the corresponding parts of the result bus, the wake-up ports of the instruction window, and ports of the register file (including decoding logic) are also deactivated.

For predictive algorithm, we ignore time and energy overheads for both architecture adaptation and DVS since they are very small compared to the time and energy for a frame. For reactive, we model a delay of 5 cycles to activate any deactivated resource. The results are not very sensitive to this parameter. We also model the energy impact of the extra bits required in each instruction window entry for reactive instruction window size adaptation (four bits, as in [10]). Other energy overheads for controlling temporally local adaptation are likely to be small, and so are ignored as explained in detail in [10].

Experiments with DVS assume a frequency range from 100MHz to 1GHz for energy and 100MHz to 2.2GHz for temperature with 1MHz steps and corresponding voltage levels derived from information available for Intel's XScale processor as further discussed in [7] and [12]. We assume clock gating for all processor resources. If a resource is not accessed in a given cycle, Wattch charges 10% of its maximum power to approximate the energy consumed by logic within the resource that cannot be gated (or cannot

App.	Type	Input Size		Base	Tight	Mean Slack on Base (%)	
		Time	Frames	IPC	Deadline	Tight	Loose
GSMdec	Speech codec	20s	1000	3.7	15μs	9.8	54.9
GSMenc		20s	1000	4.6	45μs	8.9	54.4
G728dec		0.63s	250	2.3	51μs	10.2	55.1
G728enc		0.63s	250	2.1	65μs	9.2	54.6
H263dec	Video codec	6s	150	3.4	450μs	15.6	57.8
H263enc		6s	150	2.3	18.78ms	25.1	62.5
MPGdec		8.33s	250	3.6	2.11ms	31.2	65.6
MPGenc		8.33s	250	3.0	55.3ms	38.7	69.4
MP3dec	Audio	26.1s	1000	3.0	495μs	22.3	61.1

Table 2: Workload, deadlines, and slack (% of deadline) on the base processor. Base IPC is the mean per frame IPC on the base processor. The loose deadlines are twice the tight ones.

always be gated when unused). We assume that resources that are deactivated by the adaptive algorithms do not consume any power. In our model, due to clock gating, deactivating an unused resource saves only 10% of the maximum power of the resource.

5.2 Workload and Experiments

Table 2 summarizes the nine applications and inputs used in this paper. These were also used in [6, 7, 10] and are described in more detail in [6] (for some applications, we use fewer frames, and for G728 codecs we use only one frame type – we combine one frame of each type from [6] for each frame here). We do not use multimedia instructions because most of our applications see little benefit from them and we lack a power model for multimedia enhanced functional units.

In our experiments, we assume 5% of deadline misses is acceptable. The energy savings, both absolute and relative, are sensitive to the slack, or leftover processing time, for the base processor.[5] Therefore, for our experiments we use two different sets of deadlines. The first set is the tightest deadlines for which the base processor still makes the deadline for all frames. We refer to this as the tight set. For the second, loose set, we double each of the tight deadlines. Table 2 gives the tight deadlines and mean slack on the base processor for both sets of deadlines.[6] The table shows that some applications have a lot of slack even with the tight deadlines. This is due to per frame execution time variability, and also, for MPG codecs, to multiple frame types, with some requiring less execution time.

6 Results

6.1 Achieving Energy/Temperature vs. Performance Tradeoffs

In this section, we demonstrate the ability of the predictive algorithms (for both energy and temperature) to predict, within a reasonable tolerance, the energy saved and the maximum temperature reached with

[5]The deadlines are assigned by the real-time scheduler which must consider the system load [13]. This interaction is beyond the scope of this study but is the subject of another paper submitted to the same special issue of Computer.

[6]Some of the deadlines are very short which might make context switch and DVS overhead too large. Although not evaluated here, one solution is to group (buffer) multiple frames, increasing the granularity of scheduling and temporally global adaptation.

App	Loose Deadlines		Tight Deadlines	
	Predictive	**Pred+React**	**Predictive**	**Pred+React**
GSMdec	0.5	0.5	1.3	1.2
GSMenc	-0.1	-0.7	-0.2	-1.6
G728dec	-0.3	-0.2	-0.7	-0.3
G728enc	0.3	0.0	0.4	0.0
H263dec	0.0	0.2	0.1	0.6
H263enc	0.0	-0.1	0.0	-0.2
MPGdec	0.1	0.3	0.2	0.6
MPGenc	2.0	2.1	4.9	5.4
MP3dec	0.0	-0.1	-0.1	-0.2

(a) % error in energy savings prediction relative to base energy

App	Loose Deadlines		Tight Deadlines	
	Predictive	**Pred+React**	**Predictive**	**Pred+React**
GSMdec	1.8	1.9	1.8	1.9
GSMenc	0.9	1.1	0.9	1.1
G728dec	0.9	1.0	0.9	0.9
G728enc	0.9	1.0	1.3	1.0
H263dec	1.0	1.1	1.5	1.5
H263enc	1.0	1.1	1.0	1.1
MPGdec	1.3	1.3	1.3	1.3
MPGenc	2.4	2.6	2.4	2.6
MP3dec	0.7	0.9	0.7	0.9

(b) % error in execution time slowdown prediction relative to actual slowdown.

Table 3: **For predictive and predictive+reactive algorithms for energy (a) error in energy savings prediction as a percentage of base energy and (b) error in execution time slowdown prediction as a percentage of the actual slowdown.**

the corresponding performance degradation. This ability to predict makes predictive algorithms capable of achieving the required energy-performance or temperature-performance tradeoffs.

6.1.1 Energy vs. Performance Tradeoffs

We measure the predictability of the energy adaptation algorithms by measuring their ability to predict energy savings and the execution time slowdown. This allows making informed energy vs. performance tradeoffs. We show results for the predictive and predictive+reactive algorithms, although note that the reactive part of the latter algorithm is previously tuned. (No results appear for the reactive algorithm because they inherently do not predict energy savings or slowdown.)

To predict energy savings, our algorithms use EPI information from the profiling phase. The energy savings predicted is the difference in the profiled EPI of the most aggressive configuration (base architecture at the highest frequency) and the profiled EPI of the configuration to be used. We measure the predictability for energy savings as the absolute difference between the predicted and the actual energy savings, as a fraction of the energy of the base (most aggressive) configuration (averaged over all frames), tabulated in Table 3(a). The lower this value, the higher the predictability. The corresponding actual energy, normalized to the base configuration, is given in Figure 1.

App	**Temperature**		**Slowdown**	
	Loose	**Tight**	**Loose**	**Tight**
GSMdec	1.9	0.5	0.6	1.5
GSMenc	0.7	1.5	1.6	1.2
G728dec	2.9	4.5	0.4	0.1
G728enc	2.0	2.3	0.6	0.9
H263dec	3.0	4.0	2.0	1.8
H263enc	0.7	7.5	3.0	1.4
MPGdec	1.5	2.8	0.1	0.9
MPGenc	1.3	4.5	1.1	0.2
MP3dec	2.3	1.8	4.4	3.3

Table 4: **% error in temperature prediction and % error in slowdown prediction for the thermal predictive algorithm.**

Similarly, the execution time slowdown predicted by the algorithm is the ratio of the predicted time per instruction (TPI) of the configuration to be used to the predicted TPI of the base configuration. Here, $TPI = \frac{1}{IPC \times frequency}$ and is calculated using the predicted (profiled) IPC and the frequency of the configuration (which is known). We measure the predictability of execution time slowdown as the absolute difference between the predicted and actual slowdown, as a fraction of the actual slowdown (averaged over all frames), tabulated in Table 3(b). Again, the lower this value, the higher the predictability.

Table 3 shows that the predictability for energy savings and performance slowdown is high. The actual error in energy savings predictions (relative to base energy) is always less than 4.9% for the predictive algorithms and less than 5.4% for the predictive+reactive algorithm, both with loose and tight deadlines. MPGenc exhibits a larger error than other applications since its IPC changes somewhat during profiling. Even this problem can be overcome by reprofiling once we detect a significant error between estimated and actual energy. The error in execution time slowdown (relative to actual slowdown) is prediction is always less than 2.6%, for both loose and tight deadlines.

Our algorithms can also predict the absolute energy savings and absolute execution time per frame. These estimates, however, depend on the ability to predict the instruction count for each frame (this ability is needed even for non-adaptive hardware for real-time scheduling). With our simple instruction count predictor, we find that the absolute execution time can be predicted with a maximum error of 15.3% and the absolute energy consumption can be predicted with a maximum error of 13.7% across all cases.

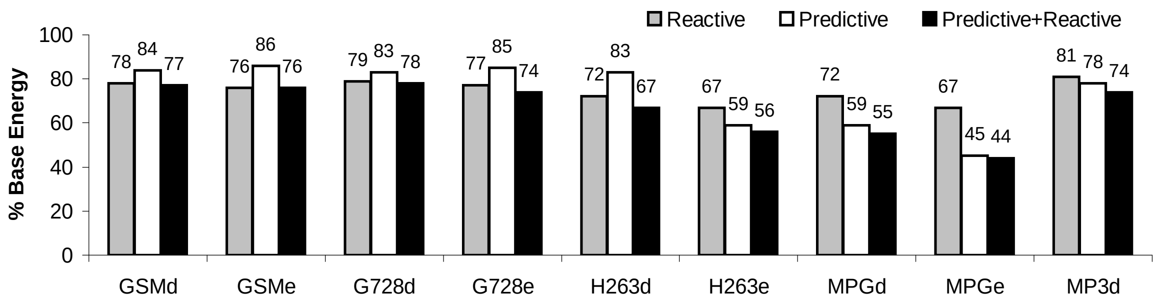
6.1.2 Temperature vs. Performance Tradeoffs

The second and third columns of Table 4 show the error in predicting the maximum temperature achieved for two temperature targets. The loose target assumes a temperature threshold of 8^0C above the temperature of the heat sink and the tight target assumes a temperature threshold of 4^0C above the temperature of the heat sink. Table 4 shows that the predictive algorithm can predict the maximum temperature quite accurately, with a less than 3% error for the loose target and $\leq$ 7.5% error for the tight target.[7]

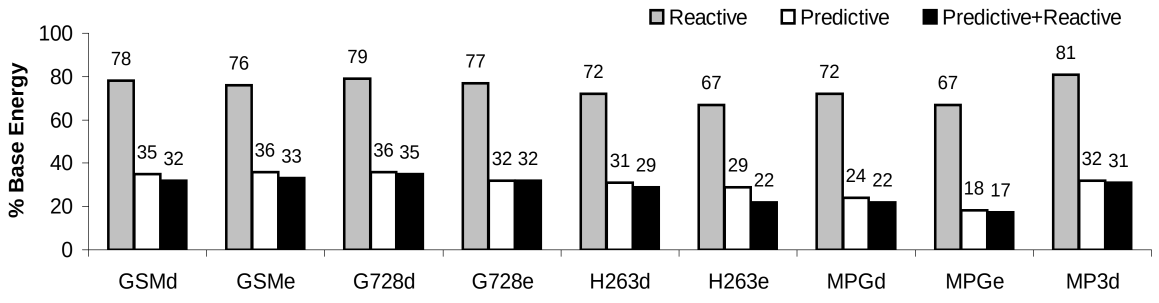
The last two columns of Table 4 show the percent error in the slowdown prediction relative to the actual slowdown for the predictive algorithm. The actual slowdown is given in Figure 2 for each application. Like the temperature prediction, the slowdown prediction by the predictive thermal algorithm is also quite accurate with $\leq$ 4.5% error for the loose target and $\leq$ 3.3% error for the tight target.

6.2 Comparing Energy Savings of Reactive and Predictive Control

6.2.1 Energy Savings



(a)



(b)

Figure 1: **Energy consumption (normalized to the base processor) with reactive, predictive and reactive+predictive algorithms with DVS and with (a) tight deadlines (b) loose deadlines**

Figure 1(a) and (b) compare the energy consumption of predictive, reactive, and predictive+reactive algorithms, with DVS. Except for GSM, G728, and H263dec with tight deadlines, the predictive algorithm consumes less energy than the reactive algorithm. For all applications, with loose and tight deadlines, predictive+reactive algorithm is the best.

Since GSM, G728 and H263dec do not have much execution time variability, it is possible to set very tight deadlines (less than 15.6% slack) with these applications. With very tight deadlines, the predictive algorithm cannot choose simpler architectures or lower frequencies since there is not much slack to exploit. However, reactive algorithms can detect resource under-utilization within a frame and power down resources

[7]Although 7.5% may sound high, this is a small error considering the target is a small 4^0C. Moreover, this error includes the 0.2^0C leeway set aside to avoid exceeding the maximum temperature.

to save energy. Even with tight deadlines, applications other than GSM, G728 and H263dec have some slack (25.1% to 38.7%) which the predictive algorithm can exploit. In particular, MPEG has three different frame types with a significant amount of execution time variability among the frame types. This leads to the predictive algorithm doing better than the reactive algorithm.

For the same reasons, with loose deadlines, the predictive algorithm does significantly better than with tight deadlines since it can trade off performance for energy by selecting less complex architectures and lower frequencies/voltages. However, the reactive algorithm cannot exploit the slack present with loose deadlines since exploiting slack entails predicting execution time with architecture adaptation and DVS. Therefore, the predictive algorithm achieves much better energy efficiency than the reactive algorithm with loose deadlines.

In all cases, predictive+reactive is best, beating the predictive algorithm by an average of 7% for the loose deadlines and an average of 9% for the tight deadlines.

6.2.2 Performance Degradation vs. Thermal Limit

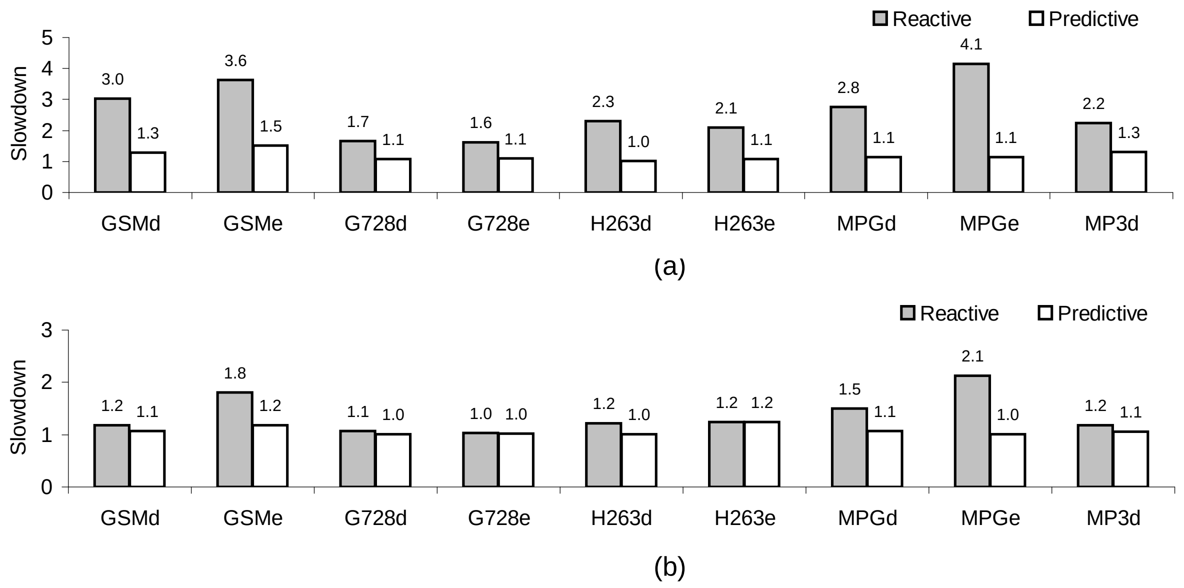


Figure 2: **Slowdown of each application with predictive and reactive thermal algorithms for a temperature target (a) 4^0C (b) 8^0C above the heatsink temperature.**

Figure 2(a) and (b) shows the slowdown of each application with predictive and reactive thermal algorithms for a temperature target of 4^0C (tight) and 8^0C (loose) above the heat sink temperature, respectively. Note that this is not percentage slowdown but the ratio of the execution time of the adaptive system to that of the base non-adaptive system. For both temperature targets, we see that the predictive algorithm achieves a lower slowdown than the reactive algorithm. The ability to predict the execution time with different architectures helps the predictive algorithm to select the highest performing architecture/frequency configuration that is thermally safe.

The predictive thermal algorithm outperforms the reactive algorithm primarily because the predictive algorithm can use high overhead adaptations like register file resizing and DVS. These adaptations result in significant power and temperature savings, with a lower penalty on performance. However, due to the large time latency involved in the invocation of these adaptations, the reactive thermal algorithm can not use them.

7 Conclusions

Multimedia applications are an increasingly important workload for general-purpose processors on a growing number of systems (e.g., future handheld, laptop, and desktop systems). These systems often operate under stringent energy and/or thermal constraints. There has been a lot of recent work on improving energy and thermal efficiency for general-purpose processors through processor hardware adaptation, often at the cost of performance. The control algorithms for processor adaptation can be broadly categorized into two classes: (1) reactive algorithms that react to the current condition of the system assuming that the same conditions will last for a short duration, and (2) predictive algorithms that predict the behavior over a considerable period and adapt accordingly. Predictive algorithms have the ability to apply high overhead adaptations like dynamic voltage and frequency scaling (DVS) and do not require the extensive tuning of reactive algorithms. Previous work has focused on the energy and thermal advantages of predictive algorithms due to these reasons.

Multimedia applications, however, additionally need high predictability due to their (soft) real-time constraints. Furthermore, they often have computational slack and are also able to operate at varying levels of quality, affording the opportunity to make intelligent performance vs. energy/temperature tradeoffs. Thus, for multimedia applications, not only should the adaptation algorithms be able to provide increased energy or thermal efficiency, but it should also be possible to predict their impact on performance, energy, and temperature.

In this paper, we show that our predictive adaptation algorithms are indeed capable of predicting their impact on performance degradation, energy savings, and temperature within a reasonable tolerance and hence capable of achieving predictable energy/temperature vs. performance tradeoffs. The algorithms are also able to predict the absolute energy and performance; however, this prediction is dependent on the accuracy of the instruction count predictor (which would also impact the predictability of the base non-adaptive system).

In terms of absolute energy savings, the predictive algorithm provides lower savings than the reactive algorithm in a few cases; a combined predictive+reactive algorithm always provides the best savings (average of 9% better than predictive for tight deadlines). We find that the predictive+reactive algorithm has good predictability; however, it requires a major effort in off-line tuning. An alternate algorithm that uses a predictive approach to get the sub-frame level savings of the reactive algorithm would provide the best of all worlds.

For thermal management, the predictive algorithm showed better performance than the reactive algorithm for all thermal limits and applications evaluated. Thus, in this case, the predictive algorithm is superior in both absolute performance and predictability.

References

[1] Iris Bahar and Srilatha Manne. Power and Energy Reduction Via Pipeline Balancing. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[2] Alper Buyuktosunoglu et al. An Adaptive Issue Queue for Reduced Power at High Performance. In *Proc. of the Workshop on Power-Aware Computer Systems*, 2000.

[3] Daniele Folegnani and Antonio González. Energy-Efficient Issue Logic. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[4] Tom R. Halfhill. Transmeta Breaks x86 Low-Power Barrier. *Microprocessor Report*, February 2000.

[5] Chirstopher J. Hughes and Sarita V. Adve. Spreading Slack for Optimal Energy-Performance Tradeoffs for Multimedia Applications. In *UIUC Technical Report No. UIUCDCS-R-2003-2358 (Submitted for Publication)*, 2003.

[6] Christopher J. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[7] Christopher J. Hughes, Jayanth Srinivasan, and Sarita V. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In *Proc. of the 34th Annual Intl. Symp. on Microarchitecture*, 2001.

[8] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. In *Proc. of the 25th Annual Intl. Symp. on Comp. Architecture*, 1998.

[9] Roberto Maro, Yu Bai, and Iris Bahar. Dynamically Reconfiguring Processor Resources to Reduce Power Consumption in High-Performance Processors. In *Proc. of the Workshop on Power-Aware Computer Systems*, 2000.

[10] Ruchira Sasanka, Christopher J. Hughes, and Sarita V. Adve. Joint Local and Global Hardware Adaptations for Energy. In *Proc. of the 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002.

[11] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *Proc. of the 8th Intl. Symp. on High Performance Comp. Architecture*, 2002.

[12] Jayanth Srinivasan and Sarita V. Adve. Predictive Dynamic Thermal Management for Multimedia Applications. In *Proc. of the 2003 Intl Conf. on Supercomputing*, 2003.

[13] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. Design and Evaluation of A Cross-Layer Adaptation Framework for Mobile Multimedia Systems. In *Proceedings of the SPIE/ACM Multimedia Computing and Networking Conference*, 2003.