Predictive Dynamic Thermal Management for Multimedia Applications*

Jayanth Srinivasan and Sarita V. Adve
Department of Computer Science
University of Illinois at Urbana-Champaign
{srinivsn,sadve}@cs.uiuc.edu

Abstract

Dynamic Thermal Management (DTM) techniques have been proposed to save on thermal packaging and cooling costs for general-purpose processors. However, when invoked, these techniques result in a significant performance degradation. This paper concerns performance-effective DTM for multimedia applications. We make two contributions: (1) Current DTM algorithms are **reactive** *in nature. We propose a* **predictive** *DTM algorithm targeted at multimedia applications, which allows the efficient use of response mechanisms that have high invocation overhead. We find that for our applications, our predictive algorithm performs significantly better than existing reactive DTM algorithms. (2) We evaluate the effectiveness of different DTM response mechanisms. Specifically, we demonstrate the importance of tailoring DTM response mechanisms to the thermal "hot-spots" on the chip and the current thermal limit, and show that a predictive combination of architecture adaptation and dynamic voltage scaling (DVS) performs the best across a broad range of applications and thermal limits.*

Categories and Subject Descriptors

C.1.1 [**Processor Architectures**]: Single Data Stream Architectures

General Terms

Algorithms, Management, Performance

Keywords

Thermal management, Adaptive architectures, Low power

1. Introduction

It is anticipated that in the future, peak power dissipation and consequent thermal considerations will often be the dominant limit to processor performance and a significant component of cost. Expensive packaging, heat sinks, and other cooling solutions are required to maintain acceptable processor temperatures. Gunther et al. estimate that above 60W, thermal packaging increases the total cost per chip by more than $1/W [7]. At the same time, the SIA road map estimates peak power consumption of 150W and higher in future processors [1], implying prohibitively expensive thermal solutions. Current thermal solutions are designed for peak processor power to ensure safe operation at all times. However, the peak processor power and resulting peak temperature are rarely observed, and tend to be much higher than the typical power and temperature. This disparity is likely to increase as further techniques are applied to extract instruction-level parallelism (ILP). Researchers have therefore proposed the use of *dynamic thermal management* (DTM) [3, 9, 15]. DTM allows the thermal solution to be designed for a temperature less than the peak. In the (hopefully) rare case when the chip approaches the thermal limit, DTM invokes a hardware response to bring down the temperature, typically by reducing system performance. Possible response mechanisms include a variety of architectural adaptations (e.g., fetch toggling or throttling) and dynamic voltage scaling (DVS).

This work concerns dynamic thermal management for processors, employing both architectural adaptation and DVS. Unlike previous work which focused on SPEC benchmarks, we consider the domain of multimedia applications. These applications are expected to form a large part of the workload on a growing number of systems employing general-purpose processors [14]. This paper exploits certain properties of these applications to propose a new DTM control algorithm that provides much better performance than previous algorithms.

All of the previous work in DTM for processors is *reactive* – the processor typically runs at full performance; only when the temperature gets too close to the thermal limit, a response to curtail performance (and thereby reduce temperature) is initiated. Such schemes suffer from at least two problems. First, they have limited time to respond to a thermal emergency; therefore, they must use mechanisms that have a low invocation time overhead or risk unnecessary performance loss. This precludes the efficient use of mechanisms such as DVS and register file resizing, which potentially have large thermal benefits but high invocation time overheads. Second, engaging the appropriate reactive response at the appropriate time requires significant prior tuning of the system. To overcome the second problem, Skadron et al. [15] proposed the use of control-theoretic techniques. This approach appears promising; however, it is as yet unclear how to tune controllers for use with the complex, potentially multiple, microarchitectural responses that are possible in the systems we consider (see Sec-

*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, EIA-0103645, CCR-0209198, and the University of Illinois. Sarita V. Adve is also supported by an Alfred P. Sloan Research Fellowship.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICS'03, June 23–26, 2003, San Francisco, California, USA.
Copyright 2003 ACM 1-58113-733-8/03/0006 ...$5.00.

tion 3). Current state-of-the-art control-theoretic DTM schemes do not use high overhead response mechanisms.

This paper proposes a new DTM control algorithm for multimedia applications that takes a *predictive* approach. Multimedia applications process discrete units of data called frames. Previous work has shown that frames of the same type involve more or less the same type (but not the same amount) of work [10]. We extrapolate this result to observe that each frame (or a predetermined number of consecutive frames) reaches the same peak temperature, which can be determined by profiling. This profile information can be used to predict the highest performance, thermally safe hardware configuration for the rest of the frames of that type. Since adaptation is invoked at most once per frame, high overhead adaptation mechanisms are possible. Additionally, our predictive algorithm is simpler in that it does not require any application-specific and response-specific tuning of reactive controllers.

We evaluate our algorithm on nine multimedia benchmarks, using the recent thermal model developed by Skadron et al. [15], for a range of thermal limits. For response mechanisms, we evaluate DVS and three architecture adaptations. To represent the state-of-the-art, we study proportional fetch-toggling (instruction fetch is disabled every N cycles). A drawback of fetch-toggling is that it affects the entire chip – while this reduces temperature across the chip, it also significantly affects performance. It is, however, well-known that power density across the chip is not uniform, resulting in localized high temperatures or thermal hot spots. We therefore also studied two adaptations that have a more localized impact – instruction window resizing and switching off active functional units (resulting in limiting the issue width and deactivating appropriate register file ports). These adaptations have not been considered before for thermal management.

Our findings are as follows. First, for our system and applications, the new predictive DTM algorithms perform significantly better than the reactive algorithms. The main reason is their ability to efficiently use thermally effective adaptations that have high invocation overhead like DVS and register file resizing.

Our second set of findings concern the effectiveness of different response mechanisms. Specifically, our results highlight the importance of selecting response mechanisms based on the thermal hot spots on chip and the thermal limit. For architectural responses, we found that for both the reactive and predictive algorithms, fetch-toggling was inferior to instruction window resizing and functional unit adaptation. Although all architectural responses ensured thermal safety, fetch-toggling showed a significantly higher performance degradation. This is because for our system and applications, the register file is always the hottest structure on chip. Compared to the other two mechanisms, fetch-toggling is overkill for controlling register file power. Similarly, comparing architectural adaptation to DVS (applicable only to the predictive algorithm), we found that in most cases, architecture adaptation provided the bulk of the benefit because it most directly addressed the thermal hot spot. Nevertheless, there are some cases where DVS is more effective and we identify these cases. Overall, a combination of DVS and architectural adaptation with a predictive algorithm proved the most effective DTM scheme.

Our predictive thermal algorithm uses observations similar to our previous work on a predictive energy management algorithm. Section 6 summarizes the differences between energy and thermal management and the relationship between the two algorithms.

2. Reactive DTM algorithms

As mentioned, all the previous work in dynamic thermal management for processors is *reactive*. While the system is running, if the temperature on chip gets too close to the thermal limit, a response mechanism is initiated. This response leads to a reduction in temperature, typically accompanied by a degradation in performance. Once the processor resumes safe thermal operation, the reaction mechanisms are shut off. There are two key features in the design of reactive DTM algorithms: when to invoke reactive response mechanisms and what response mechanism to use.

2.1 Previous reactive schemes and limitations

Huang et al. [9] proposed DEETM, a framework capable of dynamically choosing multiple responses for energy and temperature management. For thermal control, DEETM checks the temperature every few milliseconds; if the temperature is close to an emergency level, hardware adaptations are invoked in a predetrmined order. The specific responses studied are DVS, entering light sleep mode, and some mechanisms targeted at the memory hierarchy.

Brooks et al. [3] propose invoking a response when the processor power consumption crosses a predetermined threshold. They studied a constant threshold of 24W for all applications and five response mechanisms – clock frequency scaling, voltage and frequency scaling (DVS), fetch-toggling, throttling (the number of instructions fetched every cycle is reduced), and speculation control (instruction fetch is disabled when the number of unresolved branches crosses a threshold). They found clock frequency scaling and DVS to be inefficient because of their invocation time overhead. Among the other responses, fetch-toggling was found to perform the best for most benchmarks and speculation control for others.

Skadron et al. [15] use formal control theory to control DTM, using fetch-toggling as their response mechanism. The previous approaches used fixed trigger temperatures (or power) and fixed strength responses for all applications and situations, resulting in conservative decisions and unnecessarily poor performance for some applications. In contrast, the use of formal control theory allows the trigger temperature and the fetch-toggling rate to vary based on the current and prior thermal stress level.

All the above DTM schemes suffer from the following three problems. First, they have a limited time to respond to a thermal emergency, and so cannot use responses with a high invocation time overhead efficiently (e.g., DVS and register file resizing). Second, engaging the appropriate response at the appropriate time requires significant response-specific and application-specific tuning of the DTM algorithm. Although Skadron et al. [15] use control-theoretic techniques to overcome this problem, it is as yet unclear how systems with complex, potentially multiple response mechanisms can be efficiently tuned. Third, the specific response mechanisms proposed for the processor core restrict the performance of the entire processor even though localized heating occurs at a faster rate than chip-wide heating. Invoking adaptations that more directly affect the thermal hot-spot can potentially provide the same thermal benefit with a less severe performance penalty. Huang et al. [9] have proposed techniques local to the memory hierarchy; however, in our simulations, the caches were not hot spots. As a result, we chose to focus on the processor core.

2.2 Reactive algorithms studied here

We compare our proposed predictive DTM algorithm with two reactive algorithms. Both algorithms track the system temperature at fixed intervals and invoke a response when the temperature of any on-chip structure comes within a certain bound of the thermal limit. This temperature at which DTM is invoked is referred to as the *trigger temperature*. To give the best showing to the reactive algorithms, we determine the trigger temperature separately for

each application and algorithm through a manual tuning process, described at the end of this section. Once the temperature falls below the trigger temperature, normal operation resumes. The two algorithms differ in the responses invoked, as described below.

R-Toggle: We use R-Toggle to represent the state-of-the-art. This is the best manually tuned (non-control theoretic) reactive algorithm studied by Skadron et al. [15]. It uses fetch-toggling as its response mechanism. Once the trigger temperature is crossed, the toggling rate is varied linearly from no toggling (normal fetch) to full toggling (no fetch) depending on the proximity to the thermal limit.

We do not model the control-theoretic algorithms studied by Skadron et al. because (1) their performance impact was very small [15], (2) we do not yet understand how to develop control-theoretic algorithms tuned for multiple responses for multiple applications as in the R-IwFu scheme described below, and wanted to use similar tuning techniques for the two algorithms to ensure a fair comparison, and (3) for most of the applications studied here, we see that temperature stays fairly constant; therefore, manually tuned algorithms are likely to perform similar to control-theoretic algorithms.

R-IwFu: As mentioned earlier, we found that the register file was the hottest structure for all our applications. Although fetch-toggling reduces the number of instructions in the processor pipeline resulting in a lower rate of register file accesses, it also significantly affects other parts of the processor resulting in performance losses. Instead, we consider response mechanisms that exploit the knowledge that the register file is the hottest structure. Specifically, we consider deactivating functional units, which results in a reduced number of active register file ports and reduces register file power regardless of the activity. We also consider resizing the instruction window. In the reactive scheme, this adaptation has an indirect effect on the register file just like toggling - however, the impact on IPC is less severe than toggling. The reason we chose this response was to contrast the reactive algorithms with our proposed predictive algorithms where instruction window adaptation has a direct impact on the register file power (see Section 3.2).

The R-IwFu algorithm reported here combines both the above responses. When the processor temperature exceeds the trigger temperature, the instruction window size and the number of functional units are reduced by a fixed amount at each temperature sample. Once the temperature goes below the trigger, the instruction window and functional units are fully activated again.

Manual tuning: Different response mechanisms used on different applications require different trigger temperatures to ensure safe operation. In choosing this temperature, we must ensure that it is neither too low (which could unnecessarily hurt performance) nor too high (which could be unsafe). For the fairest comparison, the trigger temperature used for each combination of the reactive algorithms and applications was individually determined to ensure the best possible performance that was also thermally safe.

3. Predictive DTM algorithms

3.1 Properties of multimedia applications

Before discussing the predictive algorithm, we discuss the application characteristics exploited by the algorithm. Recall that multimedia applications typically process discrete units of data called a frame. Further, some of the applications studied statically distinguish multiple frame types, e.g., I, P, and B frames for MPEG. Our algorithm uses two prior results for such applications [10, 12]: (1) For a given application, architecture, and frequency, average IPC and average power dissipation of a frame are almost constant among all frames of the same type. This is because while the amount of work per frame may vary, the nature of the work is roughly the same for all frames of a given type.(2)IPC of a frame is almost independent of clock frequency, since little time is spent in memory stalls.

For each frame type,

For each architecture, A:

(1) Measure IPC_A and $T_{max|A,fprof}$ (at a common safe profiling voltage/frequency, V_{prof} / f_{prof})

(2) For the thermal limit, T_{limit}, find the maximum safe frequency, $f_{max|A}$, using:

$$\frac{T_{limit}}{T_{max|A,fprof}} = \frac{V_{max|A}^2 f_{max|A}}{V_{prof}^2 f_{prof}}$$

(3)If $f_{max|A}$ < min. supported freq., then discard architecture A.

(4) If $f_{max|A}$ > max. supported freq, then $f_{max|A}$ = max. supported freq.

(5) Performance$_A$ α IPC$_A$ X $f_{max|A}$

Choose architecture, A, with highest performance, running at $f_{max|A}$.

Figure 1. The predictive thermal algorithm.

From the above observations, since the IPC and power dissipation of a frame are constant, and the nature of the dominant computation is constant, we extrapolate that the temperature profiles and the maximum temperature attained during different frames of the same type will be similar. We later validate this assumption (Table 4).

3.2 Predictive control algorithm

The goal of our predictive thermal algorithm is to determine the highest performing, thermally safe architectural configuration. This is a significant point of difference from our previous predictive energy management algorithm [12] (Section 6 describes all the differences in detail). The energy algorithm exploits the characteristic that there is a fixed deadline for the execution of a multimedia application frame. It therefore seeks to save energy by *slowing* the frame execution as much as possible, without violating the deadline. In a multiprogrammed environment, however, it is advisable to finish an application frame before its deadline, to allow other applications to be scheduled in a timely way. Hence, our predictive thermal algorithm seeks to execute applications as *fast* as possible while ensuring thermal limits are not violated.

The algorithm, summarized in Figure 1, starts with the application being profiled at a frequency, f_{prof}. For each architectural configuration, A, the algorithm measures IPC_A (the IPC of A), and maximum temperature reached ($T_{max|A,f_{prof}}$) by any structure on chip for an appropriate number of frames of each type. To ensure that proper thermal behavior is observed, it may be necessary to profile multiple frames of an application for each architecture and frame type. The number of frames that need to be profiled will depend on the thermal RC time constant of the structures on chip and the size of each application frame. Section 4 gives the number of profile frames for the applications and system we study. To ensure there is no thermal emergency, profiling is performed at the lowest supported frequency and voltage (if the system approaches thermal emergency during profiling, then the architecture is discarded, and is no longer considered a potential candidate by the algorithm). Note that the algorithm tracks the temperature of each thermally important structure individually; we use the granularity of logical functional blocks for this purpose (Section 4.3.2).

Next, the algorithm determines the maximum frequency, $f_{max|A}$ at which each profiled architecture is still thermally safe (determined separately for each frame type). For the thermal model we

use (Section 4.3.2), $f_{max|A}$ can be calculated using ,

$$\frac{T_{limit}}{T_{max|A,f_{prof}}} = \frac{V_{max|A}^2 f_{max|A}}{V_{prof}^2 f_{prof}} \quad (1)$$

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 $f_{max|A}$, V_{prof} is the profiling voltage supporting the profiling frequency, f_{prof}, 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). Section 4.3.2 shows how the above relationship can be derived for the thermal model we use. The exact form of the relationship could be different for other thermal models (e.g., when leakage effects are accounted for).

If $f_{max|A}$ is not supported in the system, the closest supported frequency lower than $f_{max|A}$ is used. If the system only supports frequencies higher than $f_{max|A}$ for an architecture A, then the architecture A cannot be used safely for the given thermal limit and is no longer considered as a candidate by the algorithm.

Now, the performance of a given hardware configuration is proportional to the product of its frequency and IPC. Since our multimedia applications spend little time in memory stalls, IPC remains almost constant across frequencies [10] obviating the need to scale IPC with frequency. Hence, 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).

Although we expect our predictions to be accurate, there is always the possibility that the hardware configuration chosen is inappropriate, either because the frame profiled was not representative or because the application behavior changed. Too conservative a choice would imply a performance loss, which could be detected in some cases and profiling could be triggered again. Too aggressive a choice would lead to a thermal crisis. Mechanisms to handle a thermal crisis are required in any real system with DTM. Depending on when the crisis is triggered, appropriate reactive "crisis responses" can be invoked. This response could range from a relatively gentle response to a drastic processor shut-down to avoid imminent chip-burn. After the thermal crisis has been handled, the predictive algorithm can re-profile the application and select a new hardware configuration.

In a multiprogrammed system, the hardware configuration selected by the above algorithm could potentially be influenced by the application scheduled in the previous time slice. We discuss this issue in Section 6.

3.3 Predictive algorithm response mechanisms

The predictive algorithm can exploit both architectural adaptation and DVS. To isolate the benefits of each, we evaluate three versions: **P-Arch** only performs architectural adaptation and does not support DVS; i.e., only one fixed frequency is available to the predictive algorithm. **P-DVS** adapts the voltage and frequency, but not the architecture; i.e., only one fixed architectural configuration is available to the algorithm. **P-ArchDVS** is the most flexible and can perform both architectural adaptation and DVS.

Like the reactive algorithm, the architectural adaptations available to the predictive algorithms (P-Arch and P-ArchDVS) are instruction window resizing and functional unit adaptation. We also studied toggling, but it was very inefficient compared to the other mechanisms and so we do not report those results here.

One important difference in the effect of instruction window adaptation as invoked in the predictive algorithm compared to the reactive algorithm is that the former is also able to change the number of active physical registers with instruction window size. A smaller instruction window requires fewer physical registers for renaming. Therefore, with the predictive algorithm, the number of active physical registers, of each type (floating point and integer), is equal to the number of logical registers plus the number of entries in the instruction window. The reactive algorithm does not change the register file size because it is not clear how to do so with our processor model (reducing the size requires "garbage collecting" register contents during the course of the execution of a frame). Dropsho et al. [6] suggest the need for a software handler to control register file resizing. Even if it could be done entirely in hardware, the latency involved will be at least of the order of a few 100 cycles, if not more, and could be too large for reactive algorithms. These are not problems with the predictive algorithm because the adaptations are invoked before the start of a frame; i.e., before any state is accumulated in the registers. This distinction is of particular importance in our experiments because the register file was found to be the hottest structure on chip.

3.4 Overheads and software support

We briefly discuss the software and hardware costs associated with the predictive algorithm. The major software overhead lies in profiling. We expect it to be acceptable for several reasons. First, many real-time multimedia scheduling algorithms already do profiling for admission control to determine CPU time (e.g., [4]). This is typically done for hundreds of frames, and can subsume the profiling for our algorithm. Second, the total number of frames that need to be profiled by the predictive algorithm is the product of the number of architectural configurations available, the number of frame types, and the number of frames that need to be profiled at a time to guarantee thermal stability (given in Table 2). This is insignificant relative to the total frames executed during the application run. Third, the profiling is on-line, so potentially the user could still get an output for certain low computation workloads like speech codecs.

The rest of the software overhead comes from using the profiling information to determine the best architecture and frequency to run. This is relatively simple and incurs negligible overhead.

Additionally, the predictive algorithm also requires information on the beginning and the end of frames as well as the frame type (for applications that support multiple types). Again, real-time applications typically already provide the former information to the operating system for scheduling purposes. This could be extended to include information on the frame type as well.

The only runtime hardware overheads involved are for invoking architectural adaptation and DVS. Functional unit adaptation and instruction window adaptation can be performed in the order of a few cycles. For DVS, current commercial processors capable of voltage scaling incur overheads ranging from 10s to 100s of μseconds [8, 13]. These overheads are also negligible since they are invoked at most once per scheduling slice.

Finally, the algorithm assumes hardware sensors for temperature measurement, but this is an issue for all DTM algorithms.

4. Methodology

4.1 Architectures

The base non-adaptive processor studied is similar to the MIPS R10000 and is summarized in Table 1. We assume a centralized instruction window that integrates the issue queue and reorder buffer (ROB) but has a separate physical register file. We also

Base Processor Parameters	
Processor speed	2.2GHz
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 non-adaptive processor.

study a version of the base processor with support for dynamic voltage/frequency scaling (DVS). The voltages used for each frequency were extrapolated from the information available for Intel's XScale (StrongArm-2) processor [13]. We allowed the frequency to range from 100MHz (at 0.7V) to 2.2GHz (at 1.75V) but our system only utilized frequencies between 1GHz and 2.2 GHz. We do not model time overheads for invoking DVS since as discussed in Section 3.4, these are expected to be negligible.

We study processors capable of fetch toggling, and adapting their instruction window size and/or the number of active functional units and issue width. The instruction window is broken into segments of 8 entries each, and at least two segments must always be active. As mentioned earlier, for the predictive algorithm, we resize the register file based on the instruction window size. For functional units, we require that at least one integer ALU must always be active. The issue width of the processor is equal to the sum of all active functional units and hence changes when we change the number of active functional units. Since we adapt the issue width of the processor with functional unit adaptation, we power down the selection logic corresponding to the functional units that are powered down. Also, when a functional unit is powered down, the corresponding part of the result bus, the wake-up ports to the instruction window, and write ports to the register file are also powered down. We model a delay of 5 cycles to power up an inactive functional unit or instruction window segment. When a functional unit or instruction window segment is to be powered down, the system must wait for the units to complete their current tasks before shutting them down. As a result, we do not charge an extra delay for powering down the functional units and instruction window. As with DVS, we do not model the overhead for register file resizing in the predictive algorithm since it is invoked infrequently.

For predictive adaptation with DVS, we profiled all possible combinations of the following configurations (54 total): instruction window size $\in$ {16,32,48,64,96,128}, number of ALUs $\in$ {6,4,2}, and number of FPUs $\in$ {4,2,1}. There was a considerable variety in the configurations chosen by the predictive algorithms for different applications. This usage data is provided in Section 7.

The reactive algorithms sample the system temperature every 1μsec (Section 4). As a result, reactive adaptations are also invoked at this granularity. For R-Toggle, we allow the toggling rate to vary over a range of 20 levels, ranging from no toggle (fetch every cycle) to full toggle (no fetch). In R-IwFu, the instruction window size is varied in steps of 16 entries. The number of ALUs and FPUs is varied in steps of a single functional unit.

App.	Type	No. Frames Profiled	Cycles per Frame	No. Frames Executed
GSMenc	Speech codec	10	8.065+E4	100
GSMdec		10	2.002+E4	100
G728enc		10	9.498+E3	100
G728dec		10	7.415+E3	100
H263enc	Video codec	1	1.544+E7	1
H263dec		1	3.431+E5	5
MPGenc		1	3.405+E7	1
MPGdec		1	1.313+E6	1
MP3dec	Audio	1	6.421+E5	10

Table 2. Workload description. For MPG, only B frames are evaluated.

Finally, our base architecture is intentionally aggressive in order to expose a wide adaptation space for our algorithms. As will be seen in Section 5.4, the entire adaptation space is exploited by our applications. Section 5.5 addresses some implications of using alternate base architectures.

4.2 Workload description

Table 2 summarizes the nine applications and inputs used in this paper. For each application, it gives: (1) the average number of execution cycles per frame, (2) the number of frames profiled for each architectural configuration and frametype (the total time profiled should be much larger than the thermal RC constant for thermal stability), (3) number of frames executed after profiling for evaluating the performance degradation of the DTM schemes. We use many more frames for our analysis in Table 4 which reports statistics on the standard deviation of IPC and maximum temperature on the base non-adaptive architecture. Specifically, we use 100 frames for the video codecs and more than 1000 frames for all the other applications.

4.3 Performance and temperature evaluation

4.3.1 Simulator

We use the RSIM simulator [11] for performance evaluation. We use the Wattch tool [2] integrated with RSIM for power measurement. We derive temperature from power using the thermal model discussed in the next subsection. The Wattch options used assume clock gating for all the components of the processor with 10% of its maximum power charged to a component when it is not accessed in a given cycle. We assume that the resources that are powered down by our adaptive algorithms do not consume any power. The power model used here does not model leakage power (leakage power was found to be small for the technology parameters assumed here, but will grow in the future).

4.3.2 Temperature model

We use the thermal model developed by Skadron et al. [15]. This model tracks the temperature of individual structures on chip using exponential rate equations, based on each structure's thermal resistance and capacitance. This allows modeling the local temperature of different parts of the chip. Temperature is modeled at the granularity of a logical functional block, treating each block as a uniform heat source. To obtain a relationship for the change in temperature ΔT, consider a block with thermal resistance R_{th} and thermal capacitance C_{th}, changing in temperature from T_{old} to T_{new} over a time interval Δt, while dissipating an average power P in the interval. Both T_{old} and T_{new} are defined relative to the heat sink temperature. This same convention of defining temperatures relative to the heat sink is used in the rest of the paper.

Now, based on the exponential rise equation,

$$T_{new} = PR_{th} - (PR_{th} - T_{old})e^{\frac{-\Delta t}{R_{th}C_{th}}} \quad (2)$$

For $\Delta t << R_{th}C_{th}$ and approximating to the first order, the exponential becomes $(1 - \frac{\Delta t}{R_{th}C_{th}})$. Based on this, the change in temperature $\Delta T = T_{new} - T_{old}$, is given by

$$\Delta T = \frac{P\Delta t}{C_{th}} - \frac{T_{old}\Delta t}{R_{th}C_{th}} \quad (3)$$

Our algorithm also requires modeling the effect of voltage and frequency scaling on temperature. We substitute the well known proportionality , $P \alpha V^2 f$, in the above equation. Simplifying, we get:

$$T_{f_1} \alpha V_1^2 f_1 \quad (4)$$

where T_{f_1} and V_1 are the system temperature and voltage at frequency f_1. Thus, the relationship between the temperatures, T_{f_1} and T_{f_2}, at two different frequencies, f_1 and f_2, is given by

$$\frac{T_{f_1}}{T_{f_2}} = \frac{V_1^2 f_1}{V_2^2 f_2} \quad (5)$$

where V_1 and V_2 are the system voltages at f_1 and f_2. The above relationship is used in our predictive algorithm in Section 3.

The thermal resistances and capacitances used in this paper are listed in Table 3. These are based on the data used by Skadron et al. in [15]. Due to a lack of publicly available information, they based the capacitances and resistances on estimates obtained from the MIPS R10000 die photo. By using different thermal resistance values, we can tweak the temperature ranges encountered by structures on chip to match current processor values. However, for consistency and a baseline for comparison, we chose to use the values listed in [15].

We adapted Wattch to track per-structure temperatures based on the equations derived above. The power consumption of the structures is sampled from Wattch every μsecond which is a small Δt, compared to the RC time constant of $100 \mu sec$ (Table 3).

As can be seen in Table 3, the time constant associated with structures on chip is significantly lower than the typical heat sink time constant (10s of seconds [15]). As a result, the temperature simulator assumes that the dynamic aspects of the heat sink for short time intervals can be ignored, resulting in a constant heat sink temperature.

Our thermal model ignores heat diffusion effects between different blocks by assuming that the tangential resistances between different functional blocks is much larger than each block's individual normal thermal resistance – in other words, in our model, the temperature of a block will not be directly influenced by any other block. Recently, Skadron et al. [16] developed a thermal model to account for thermal diffusion effects. However, this model was used only to model the effects of diffusion on steady state temperatures, and not for dynamic temperature simulation. The thermal model also neglects errors in sensor placement and readings, and the effect of leakage power on temperature. It should be noted that these issues arise in all DTM research. Section 6 discusses the impact of these approximations on our results.

4.4 Thermal limits

We define the thermal limit at any time as the difference between the maximum allowed temperature of the chip and the temperature of the heat sink at that time. The thermal control algorithms discussed in this paper all strive to maintain the processor temperature within this thermal limit. If the processor crosses the thermal limit, it enters thermal crisis. Note that the thermal limit depends on the quality of the heat sink and the ambient temperature. Since it is desirable to design processors to work with a variety of heat sinks and ambient temperatures, we model multiple thermal limits ranging from 4^oC to 14^oC above the heat sink temperature. This particular range was chosen based on the maximum temperatures reached during our simulations by the baseline non-adaptive processor. If we had used different thermal resistance values, our range would be different.

Structure	Area (m^2)	R (K/W)	C (J/K)
LSQ	5.0e-7	2	5.0e-5
Inst. Window	9.0e-7	1.11	9.0e-5
Regfile	2.5e-7	4	2.5e-5
Bpred	3.5e-7	2.86	3.5e-5
D-Cache	1.0e-6	1	1.0e-4
ALU	1.0e-6	1	1.0e-4

Table 3. Per-structure data. RC time constant of all structures is $100 \mu sec$.

4.5 Metrics

As in previous DTM work [3, 15, 9], we use performance as the main metric of comparison. Our algorithms seek to minimize the performance impact while maintaining safe thermal levels.

Additionally, the thermal control algorithm should strive to avoid or at least minimize the cycles spent in crisis (in the latter case, other aggressive techniques might be required to ensure chip safety). As a result, a crucial metric when evaluating thermal control algorithms is cycles spent in crisis. In order to limit the design space evaluated, *we manually tuned the trigger temperature of different mechanisms to ensure that no algorithm spent any cycles in crisis* (see Section 2.2).

5. Results

5.1 Application temperature profiles

For each application, Figure 2 shows the maximum temperature on chip and the IPC over time, collected at the granularity of 1 μsec for the base non-adaptive architecture without any thermal control mechanisms. The horizontal axis is the time in μsecs. The vertical axis is the maximum temperature above the heat sink temperature for the upper curve and IPC for the lower curve. These profiles cover the parts of the execution that showed the most variability in temperature. Table 4 lists the range, mean, and standard deviation in temperature of the two hottest structures on chip for each application. In our simulations, the hottest structure on chip was always the register file, and the second hottest structure was always the ALU. Table 4 also includes the corresponding IPC statistics.

As can be seen, with the exception of the video encoders, the overall variation in temperature with time is very low for all the applications. Even in the case of the video encoders, distinct temperature phases sustained over long periods of time are visible. The work performed by the video encoders takes more time and tends to be more varied than the work done by the other applications. This leads to phase behavior seen in their temperature profiles.

As is evident from the profiles, local variations seen in the IPC are smoothed out in the temperature curve. This is because of the relatively large thermal time constant for the structures on chip (100 μsec). However, as can be seen, a significant change in IPC over a sustained period of time results in a corresponding change in temperature. Table 4 highlights this observation. The only applications with a standard deviation in temperature greater than 1.5% are the video encoders. However, all the applications experience significant standard deviation in IPC (from 18% to 27%).

5.2 Reactive algorithms

Table 5 compares the performance of the two reactive schemes discussed in Section 2. For each application, the ratio of execution times of R-Toggle to R-IwFu is shown for different thermal limits. The higher the ratio, the better R-IwFu performs compared to

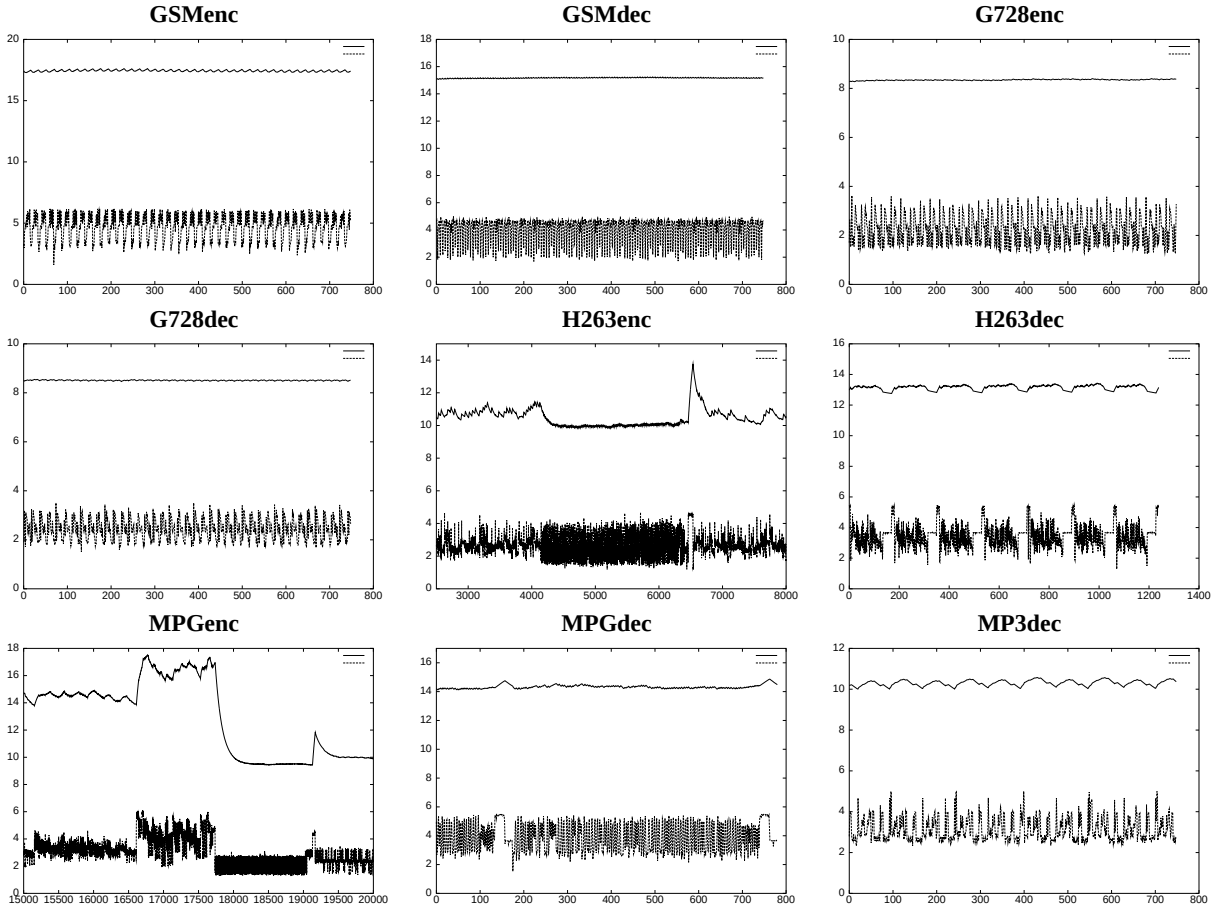

Figure 2. Maximum temperature (upper curve) and IPC profiles (lower curve) at a $1\mu sec$ granularity for the base non-adaptive processor. The y-axis is both maximum temperature above the heat sink temperature and IPC. The x-axis shows the time in $\mu seconds$.

App.	Hottest structure (^oC over heatsink)			2nd hottest structure (^oC over heatsink)			IPC		
	Range	Mean	Std. Dev. (%)	Range	Mean	Std. Dev. (%)	Range	Mean	Std. Dev. (%)
GSMenc	17.29-17.62	17.44	0.40	04.93-05.02	04.97	0.40	1.61-6.16	4.72	21.74
GSMdec	15.06-15.24	15.17	0.21	14.29-14.35	14.32	0.22	1.69-5	3.92	24.82
G728enc	08.27-08.40	08.35	0.27	03.05-03.10	03.08	0.27	1.24-3.63	2.20	27.01
G728dec	08.47-08.55	08.51	0.18	03.00-03.03	03.02	0.17	1.52-3.51	2.42	18.18
H263enc	09.79-13.77	10.52	4.41	03.32-03.99	03.44	1.91	1.18-4.68	2.63	20.97
H263dec	12.76-13.42	13.14	1.24	03.58-03.79	03.70	1.28	1.29-5.46	3.46	21.97
MPGenc	09.44-19.13	14.53	14.22	03.27-05.40	04.37	10.47	1.26-6.12	3.40	26.75
MPGdec	14.13-14.86	14.33	0.91	04.07-04.28	04.19	0.75	1.53-5.49	3.92	25.25
MP3dec	10.01-10.57	10.39	1.31	03.50-03.62	03.57	0.68	2.32-5.01	3.13	18.73

Table 4. Statistics for temperature and IPC variability at a 1 μsec granularity for the base non-adaptive processor.

App.	Thermal limit. (^oC above the heat sink temperature)										
	4	5	6	7	8	9	10	11	12	13	14
GSMenc	4.34	4.06	2.76	2.21	1.78	1.57	1.34	1.27	1.19	1.13	1.08
GSMdec	3.25	3.30	2.04	1.60	1.26	1.04	1.03	1.03	1.03	1.02	1.01
G728enc	4.61	2.83	1.73	1.27	1.07	1.00	1.00	1.00	1.00	1.00	1.00
G728dec	4.75	2.85	1.79	1.29	1.07	1.00	1.00	1.00	1.00	1.00	1.00
H263enc	6.00	3.93	2.29	1.72	1.41	1.23	1.05	1.01	1.01	1.01	1.00
H263dec	5.41	3.75	2.39	1.82	1.45	1.23	1.04	1.01	1.01	1.00	1.00
MPGenc	2.01	2.05	1.79	1.71	1.50	1.59	1.36	1.34	1.17	1.12	1.10
MPGdec	5.50	3.90	2.67	2.10	1.70	1.48	1.27	1.18	1.11	1.06	1.02
MP3dec	4.69	2.91	1.92	1.45	1.17	1.04	1.03	1.00	1.00	1.00	1.00
Average	4.51	3.29	2.15	1.69	1.38	1.24	1.12	1.09	1.06	1.04	1.02

Table 5. Reactive schemes - Ratio of execution time of R-Toggle to R-IwFu.

R-Toggle. As can be seen, R-IwFu performs similar to or better than R-Toggle for all the points in the table. For a thermal limit of $\leq 6^oC$ above the heat sink temperature, R-IwFu performs at least 1.7 times better than R-Toggle, and performs 6 times as well in one case. The performance difference decreases as we move towards less stringent thermal limits since the reactive responses are invoked less frequently.

As mentioned, the hottest structure on chip is the register file. Functional unit adaptation directly affects register file power by reducing the number of active ports in the register file. On the other hand, although toggling reduces the number of instructions in the pipeline at any given time, it does not target the register file thermal power directly (except by reducing the average activity factor). Hence, to achieve the same thermal benefit, R-Toggle creates a larger IPC drop in the application than functional unit adaptation. Although instruction window adaptation is a global mechanism, it provides more register file thermal benefit than toggling. Finally, since toggling disables fetch, it will always result in a performance degradation when invoked. On the other hand, if the processor resources are not being completely utilized, functional unit or instruction window adaptation can possibly result in thermal benefit with minimal performance loss. This is discussed further in Section 5.5.

The performance improvement of R-IwFu over R-Toggle motivates hot spot aware adaptations for temperature control. Rather than using globally restrictive schemes which are not motivated by specific hot spots (like R-Toggle), an adaptation targeted at the structure with the largest thermal stress would likely result in better performance. It should be noted that the register file might not always be the hottest structure on chip. In such cases, local adaptations specific to the new hottest structure would have to be invoked.

5.3 Predictive algorithms

Figure 3 shows the performance of the three predictive schemes discussed in Section 3, P-DVS, P-Arch, and P-ArchDVS, for different thermal limits. It also shows the same data for the best performing reactive scheme, R-IwFu. The horizontal axis in the graphs corresponds to the thermal limit, and the vertical axis represents the performance in terms of the slowdown over the base non-adaptive architecture. Note that this is not % slowdown, but the ratio of execution times of the DTM algorithms over the base non-adaptive architecture. As the thermal constraints become stricter (from right to left on the horizontal axis), the slowdown increases (from bottom to top on the vertical axis). Table 6 provides a pairwise comparison of R-IwFu vs. P-ArchDVS, P-DVS vs. P-Arch, and P-Arch vs. P-ArchDVS for a strict, moderate, and relaxed thermal limit.

As can be seen, the predictive schemes perform significantly better than the reactive schemes for all the applications including the video encoders which showed some temperature variability. On average, for a strict thermal limit of 4^oC above the heat sink temperature, the best performing predictive scheme, P-ArchDVS is more than twice as fast as the best performing reactive scheme, R-IwFu. This is primarily because the predictive schemes can use high time overhead adaptations like register file resizing and DVS. These adaptations result in significant power and temperature savings, with a relatively lower penalty on performance. Also, since very little variability is seen in the temperature profiles of most applications, the hardware configuration chosen by the predictive algorithm tends to be close to optimal for the entire execution.

Among the predictive schemes, P-ArchDVS always performs the best. This is rather intuitive - all the hardware configurations

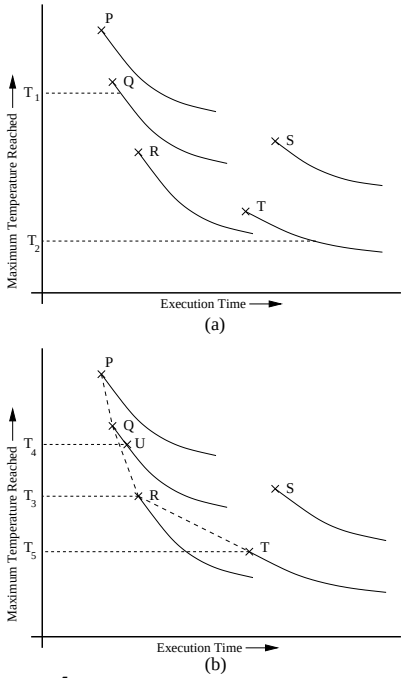

Figure 4. Sample configuration space for algorithm.

available to P-Arch and P-DVS are subsets of the configurations available to P-ArchDVS. Comparing architectural adaptation and DVS, the bulk of the benefit comes from architectural adaptation. Specifically, the addition of DVS to P-Arch only gives an 8% benefit at 4^oC, and 1% at 8^oC and 14^oC. Similarly, comparing P-Arch and P-DVS, we see that P-DVS is 23% slower than P-Arch at 4^oC. The performance difference between the different schemes reduces as the thermal limit is relaxed (from left to right on the horizontal axis in Figure 3). Section 5.5 qualitatively analyzes the reasons for the above trends.

5.4 Configurations chosen

To help us understand the extent to which the adaptive hardware features provided to the predictive algorithms are exploited, we look at the hardware configurations chosen by the different predictive algorithms for each application and thermal limit. Table 7 lists the architectural configurations chosen by P-Arch, and Table 8 lists the frequencies chosen by P-DVS. To conserve space, Table 7 merges the 54 configurations available to P-Arch into groups. In general, lower numbered configurations are more aggressive than higher numbered ones. Similarly, Table 8 lists frequency ranges (the lowest frequeny chosen was 1168MHz).

As can be seen in Table 7, no single architectural configuration is best for all applications and thermal limits. A given processor may be designed for multiple applications and thermal limits (the thermal limit can vary depending on the heat sink used or the ambient temperature). Table 7 indicates that such a processor would utilize a considerable range of architectural configurations, clearly motivating architectural adaptation for thermal control. Similarly, Table 8 indicates the large frequency range utilized, motivating DVS for thermal control.

For lack of space, we do not show the configurations chosen by P-ArchDVS. Overall, P-ArchDVS chose more aggressive architectures than P-Arch with higher frequencies than P-DVS.

5.5 Qualitative analysis

This section presents a qualitative analysis of the above results (in a style similar to that for the energy algorithm in [12]).

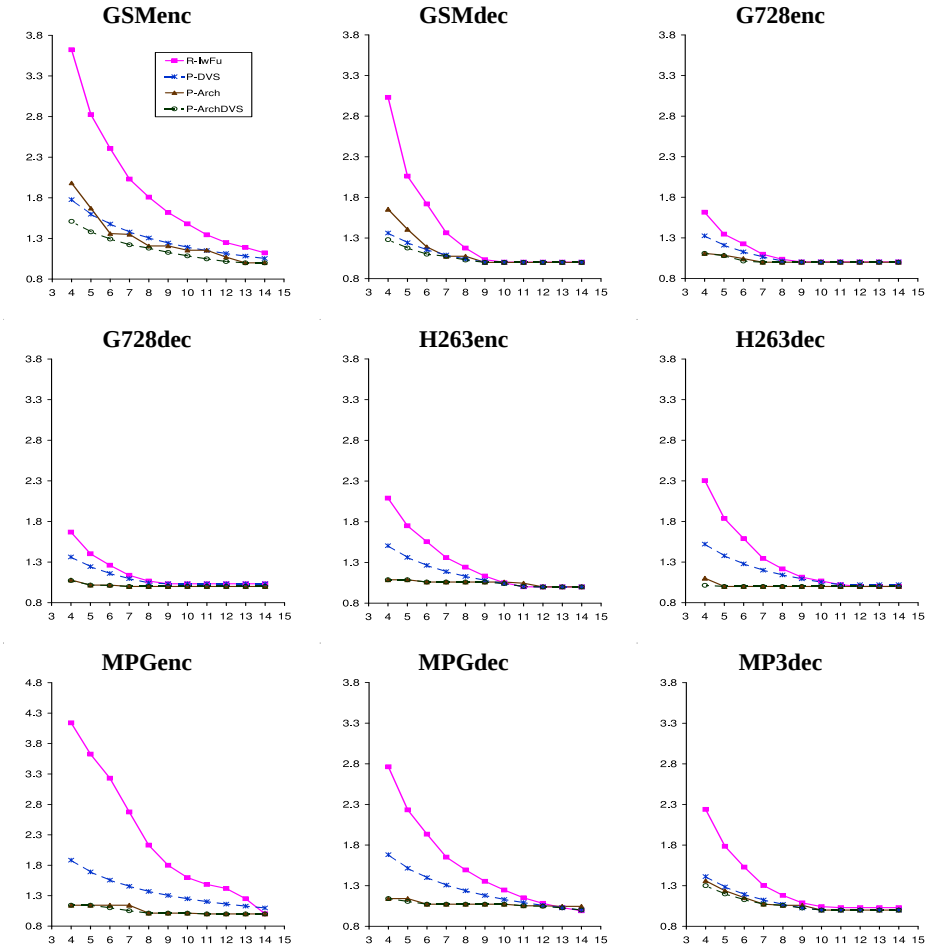

Figure 3. Predictive algorithms for all applications. The y-axis is the slowdown caused by the DTM algorithm over the base non-adaptive architecture. The x-axis shows the thermal limit in $^{\circ}C$ above the heat sink temperature. MPGenc is on a different scale.

App.	R-IwFu vs. P-ArchDVS			P-DVS vs. P-Arch			P-Arch vs. P-ArchDVS		
	4 $^{\circ}C$	8 $^{\circ}C$	14 $^{\circ}C$	4 $^{\circ}C$	8 $^{\circ}C$	14 $^{\circ}C$	4 $^{\circ}C$	8 $^{\circ}C$	14 $^{\circ}C$
GSMenc	2.40	1.53	1.12	0.89	1.08	1.05	1.31	1.02	1.00
GSMdec	2.37	1.14	1.00	0.82	0.97	1.00	1.29	1.05	1.00
G728enc	1.46	1.04	1.00	1.19	1.02	1.00	1.00	1.00	1.00
G728dec	1.55	1.07	1.03	1.26	1.04	1.03	1.00	1.00	1.00
H263enc	1.92	1.17	1.00	1.38	1.06	1.00	1.00	1.00	1.00
H263dec	2.27	1.22	1.00	1.38	1.14	1.02	1.09	1.00	1.00
MPGenc	3.62	2.10	1.00	1.65	1.35	1.10	1.00	1.00	1.00
MPGdec	2.42	1.39	1.00	1.47	1.15	0.96	1.00	1.00	1.05
MP3dec	1.72	1.12	1.03	1.04	1.01	1.00	1.05	1.00	1.00
Average	2.19	1.31	1.02	1.23	1.09	1.02	1.08	1.01	1.01

Table 6. Comparison of different schemes- Execution time ratios for of R-IwFu—P-ArchDVS, P-DVS—P-Arch, and P-Arch—P-ArchDVS for a strict, moderate and relaxed thermal limit.

Thermal limit ($^{\circ}C$)	P-Arch configurations chosen					
	1-9	10-19	20-29	30-39	40-49	50-54
14	GSMe,GSMd,G728e,G728d,MP3d,H263e,MPGe,MPGd	H263d				
10	GSMd,G782e,G728d,MP3d,MPGd	GSMe,H263e,H263d,MPGe				
6	G782e,MPGd	GSMd,G728d,MP3d,H263e,H263d	GSMe			MPGe
4	GSMe,G782e,G728d	GSMd	MP3d,MPGd	H263e,H263d	H263e	MPGe

Table 7. Hardware configurations chosen by P-Arch. Lower numbered configs. are generally more aggressive than higher ones.

Thermal limit ($^{\circ}C$)	P-DVS frequencies chosen (MHz)				
	2200	1900-2200	1600-1900	1300-1600	0-1300
14	G728e,G728d,MP3d,H263e,H263d,MPGd	GSMe,GSMd,MPGe			
10	G728e,G728d	MP3d,H263e,H263d,MPGd	GSMe,GSMd,MPGe		
6		G728e,G728d	MP3d,H263d,MPGd	GSMe,GSMd,H263e,MPGe	
4			G728e,G728d	MP3d,H263e,H263d,MPGe	GSMe,GSMd,MPGe

Table 8. Frequencies chosen by P-DVS. No frequency below 1000MHz was chosen.

Representing the configuration space. To aid our analysis, we consider the configuration space available to the different schemes, and reason about which configuration would be chosen. Figure 4(a) represents the configuration space, showing the tradeoff between the maximum temperature (y-axis) and execution time (x-axis) for each configuration, for a hypothetical application.[1] Each curve in the figure represents an available architecture; different points on the curve represent the architecture running at different frequencies (the reason for the shape of the curves is given below). All points in the figure are available to P-ArchDVS (assuming voltage/frequency scaling over a continuous range). P-Arch, however, operates with a fixed frequency and so can only choose configurations with different architectures at this fixed frequency, marked by P, Q, R, S, and T in the figure. P-DVS, on the other hand, is restricted to the one curve that corresponds to the chosen base architecture, P, in our case. In the figure, configurations that are closer to the origin are generally more desirable since they reach low temperatures and have high performance. For a given thermal limit (maximum temperature), we would like to choose the left-most configuration available that is below this temperature.

We start by focusing on P-ArchDVS since it has the most configurations available to it. We first explain why P-ArchDVS sees the large benefits from architecture adaptation and then explain why it generally sees only little benefit from DVS in our experiments. We then discuss why P-Arch does better than P-DVS and why our predictive algorithms work better than our reactive algorithms for our experiments.

Benefits from architecture adaptation for P-ArchDVS: To understand P-ArchDVS, we first derive a mathematical expression for the shape of the curves in Figure 4(a). For an architecture A at frequency f, execution time is proportional to $\frac{1}{IPC_A \times f}$. From equation 1 and since voltage is approximately linearly related to frequency for a large range, the maximum temperature for architecture A at frequency f is $T_{max|A,f} = T_{max|A,f_{prof}} \times \frac{f^3}{f_{prof}^3}$. Since the profiling frequency, f_{prof}, is a constant, $T_{max|A,f} \propto \frac{T_{max|A,f_{prof}}}{IPC_A{}^3} \times \frac{1}{Execution\ time^3}$, defining the shape of each curve.

The value $\frac{T_{max|A,f_{prof}}}{IPC_A{}^3}$ is a constant for architecture A and we refer to it as the *thermal constant* (higher thermal constants imply higher temperatures for a given execution time). Thus, in the range where voltage $\propto$ frequency, it follows that each curve in Figure 4 follows a cubic relationship and will not intersect with any other curve. Further, for any thermal limit, T_{limit}, the maximum performance will be obtained by the architecture with the lowest thermal constant, say A_{lowest}, running at the frequency $f_{prof} \times \frac{T_{limit}}{T_{max|A,f_{prof}}}^{\frac{1}{3}}$. Denote this frequency as $f_{A_{lowest}}$.

We can now deduce three cases where architecture adaptation will provide significant benefits to P-ArchDVS. First, if the architecture with the lowest thermal constant (A_{lowest}) is different for different applications, then architecture adaptation will be beneficial across applications.

Second, since the frequency range supported by a typical system is limited, it is possible that for architecture A_{lowest}, the corresponding $f_{A_{lowest}}$ for the required thermal limit is higher than the highest frequency supported by the system. Thus, this system cannot see the full performance potential of architecture A_{lowest} at that thermal limit, and there could be another safe architecture with higher performance at a supported frequency. In Figure 4(a), architecture R is A_{lowest}, but architecture Q has the best performance for thermal limit T_1 within the supported frequency range, again motivating architecture adaptation.

Finally, it is possible that $f_{A_{lowest}}$ for the required thermal limit is lower than the lowest frequency supported by the system. In this case, A_{lowest} cannot be used safely for the specified temperature limit and an alternate architecture must be used, again motivating architecture adaptation. Figure 4(a) illustrates this situation at temperature T_2 – architecture R does not reach T_2 at any supported frequency, but architecture T (which has a higher thermal constant) does.

In our experiments, we only see the first and second source of benefits from architectural adaptation with P-ArchDVS. The third situation was not seen, but could occur with even lower thermal limits or with systems with a more limited range of supported frequencies.

Benefits from DVS for P-ArchDVS: Without DVS P-ArchDVS defaults to P-Arch and is forced to pick configurations at a fixed frequency. In Figure 4(a), points P, Q, R, S, and T are the choices available. Note, however, that architecture S will never be chosen since R and T have both higher performance and lower temperature than S. Below, we refer to a curve joining the points that are considered by P-Arch (in our example, a curve joining P, Q, R, and T) as the P-Arch curve. Part (b) of Figure 4 illustrates this curve (dashed line) for a hypothetical application, along with the (solid) DVS curves similar to part (a). P-Arch will always choose the left-most architecture on its curve that is below the specified thermal limit. DVS will provide a benefit to P-ArchDVS for thermal limits where a DVS curve has a point to the left of the architecture chosen on the P-Arch curve. There are three key cases.

First, at the thermal limits where the slope of the P-Arch curve is less steep than the DVS curves (which are cubic), DVS will provide configurations with better performance. In figure 4(b), this occurs at thermal limits of T_3 and lower. At thermal limits above T_3, the P-Arch curve (joining points P, Q, and R) is steeper than all the DVS curves. We find that for most of our applications and thermal limits, the P-Arch curve falls at a steeper rate than the DVS curves. Due to the aggressiveness of our base non-adaptive architecture and the effectiveness of our adaptation mechanisms, architectural adaptation results in a large thermal benefit for a small performance loss, resulting in better benefits than DVS. However, at stricter thermal limits, the effectiveness of architectural adaptation reduces because the available system resources become more critical. Hence, the rate of fall of P-Arch configurations can become less than cubic and DVS becomes more effective than architecture adaptation (thermal limit lower than T_3). For our applications, this effect is significant only for GSMenc and GSMdec at the strictest thermal limits.

A second case where DVS may do better is because of the discrete nature of the P-Arch curve. Even though the curve joining P, Q, and R is to the left of the DVS curves in Figure 4(b), not all points on the curve are available configurations. Thus, for a thermal limit of T_4, DVS would enable us to pick configuration U while P-Arch would have to pick configuration R. However, again, because of the steeper slope of the P-Arch curve and because we have a large number of architecture configurations available, generally, the performance difference between U and R can be expected to be small.

Finally, DVS provides benefits for all temperatures below the lowest maximum temperature supported by all of the architecture configurations available to P-Arch. For example, in Figure 4(b),

[1] The shapes and relationships of the different curves are similar to the real applications; however, some effects are exaggerated for easier understanding.

below temperature T_5, P-Arch has no safe choice. However, we did not see this in our experiments because we only considered temperature ranges where all algorithms had some safe choices.

In summary, DVS can potentially provide large benefits to P-ArchDVS at thermal limits where further reducing the architectural resources for the safe architectures will severely impact IPC, or if there is no safe architecture configuration supported for those limits. These situations generally occur at relatively low thermal limits. For most of the applications we tested, this limit was not reached. We do see the trend, however, that as the thermal limit is reduced, the addition of DVS provides increasing benefits for P-ArchDVS.

P-Arch vs. P-DVS. The above discussion also clarifies why P-Arch performs significantly better than P-DVS in our experiments. The configuration space available to P-DVS is limited to a single curve in the graph, and depends on the base architecture. We evaluated P-DVS on the most aggressive architecture available. In figure 4(b), this corresponds to the DVS curve passing through point P. The above reasoning based on the relative steepness of the DVS and P-Arch curves is relevant here as well, and shows again why P-Arch is better, especially at high thermal limits.

Note that if we had chosen an architecture with a lower thermal constant as our base (i.e., a curve lower than the current base), then P-DVS would show better performance relative to P-Arch (since the slope of the P-Arch curve becomes less steep at the architectures with lower thermal constants). However, since the ordering of architectures based on the thermal constant is different for different applications, it is unclear which architecture to choose based on this reasoning. Furthermore, our current base architecture affords performance points not seen in other architectures, and is the architecture of choice for many situations (see Table 7), justifying its use as the base.

Reactive vs. predictive algorithms. The behavior of the reactive vs. predictive algorithms can also be explained using Figure 4(b). In the figure, the curve passing through different configurations available to R-IwFu would pass through point P and stay above the P-Arch and P-DVS curves. Similarly, the reactive toggling curve would be higher than the R-IwFu curve.

6. Discussion and future work

Predictive+reactive algorithm: Although our predictive algorithm performs well, there is still room for improvement. Based on the peak temperature reached during profiling, the predictive algorithm chooses a hardware configuration that would be safe for the entire application run. For applications that see some variability in temperature (e.g., the video encoders), this configuration may be sub-optimal when the temperature is significantly lower than the peak. As seen in our applications, even with high variability, the temperature stays constant for a relatively long period. This is because of the relatively large thermal time constants of the structures on chip – significant changes in IPC over a sustained period in time are required to create noticeable changes in temperature. This motivates a coarse-grained reactive algorithm piggy-backed on our predictive algorithm. We leave this to future work.

Non-multimedia workloads: Although our evaluation has focused on multimedia applications, we feel that this work could apply to other general-purpose workloads (e.g., SPEC) as well. For reasons discussed above, we can expect that even with general workloads, the temperature will stay roughly constant for relatively long phases. Although these workloads do not have the periodic, frame-based behavior of multimedia applications, much recent work on runtime optimization is motivated by repetitive (though not periodic) execution phases in these applications. It is likely that constant-temperature phases will be correlated with such phases, and prior work can be leveraged to detect them. In that case, again, a predictive+reactive thermal management algorithm such as described above may be possible. This is also a fertile ground for future work.

Multiprogrammed workloads: In a multiprogrammed environment, our predictive algorithm will work on a per-application basis. If multiple multimedia applications are running, each application will be profiled individually and controlled separately based on its specific IPC and temperature profiles. Since operating system time slices are of the order of milliseconds which is much greater than the thermal time constant, we expect that an application will not be affected by the thermal properties of other applications running concurrently on the system. Due to the large time slice, the application will reach a stable temperature which depends only on its properties and not on the starting temperature. However, it is possible that other applications running on the processor could gradually cause the heat sink temperature to change. This will change the thermal margin allowed to the predictive algorithm. In such a situation, the predictive algorithm should select another configuration for the new thermal limit at the beginning of the time slice – this does not require any reprofiling, but only determining a new architecture/frequency combination for the new thermal limit. Finally, our predictive algorithms do not require all the concurrent applications to be multimedia applications. The predictive algorithm will control the multimedia applications while other applications will be controlled by the processor's default thermal control mechanisms.

Thermal vs. energy control: As mentioned earlier, although there are similarities between processor energy reduction and processor thermal control, there exist distinct differences in the two goals. We discuss these differences below. Since our previous predictive energy algorithm [12] is closely related to the predictive thermal algorithm, we focus on a comparison between the two.

We first briefly describe the previous energy algorithm [12]. The algorithm adapts at the frame granularity; adaptations can be invoked at frame boundaries. For a given application and frame type, the algorithm profiles the average power, P, and IPC of all the candidate architectures. Using an instruction count predictor to predict the instruction count of future frames, and ranking the architectures in increasing order of energy-per-instruction (which can determined from P and IPC), the algorithm selects the most energy efficient architecture that will also meet the frame deadline.

We next discuss the differences between energy and thermal control in general, and between the specific predictive algorithms in particular. First, energy and thermal control operate at different temporal and spatial granularities. Energy control attempts to lower the sum total of energy consumed over the entire application run, while thermal control must ensure that at no time is the thermal limit exceeded. Therefore, our previous energy algorithm is sensitive to the amount of work done in a particular frame and potentially applies a different adaptation for different frames. Our thermal algorithm on the other hand picks the same configuration for all frames, since it only cares about the peak temperature which does not change. Similarly, the spatial granularity at which the two algorithms operate differs. The energy algorithm tracks the power consumption of the entire chip as a whole, while the temperature algorithm concentrates on the power consumption of specific localized structures on chip. A direct result of this is that architectural adaptation is better than DVS for thermal control, but DVS was found to be better for the predictive energy algorithm.

The second set of differences occur because the energy algo-

rithm seeks to minimize energy while meeting a performance deadline. This deadline is determined by the OS based on system load. Thus, there is a minimum performance target. On the other hand, with thermal control, there is no minimal target – the only constraint on the system is that it should not burn. In a system where energy is not a constraint, the goal of the thermal algorithm is to avoid chip-burn with the maximum possible performance, so it can allow as many applications to run as possible. This is the scenario we evaluate in this paper. In contrast, the energy algorithm has the option of slowing down as long as the deadline is met.

Thermal model errors: As mentioned in Section 4.3.2, our thermal model ignores effects due to heat diffusion, errors in sensor placement and readings, leakage power, and non-ideal heat sinks. However, other than sensor error (which is an issue for all DTM schemes), these approximations do not affect the premise or the operation of our predictive algorithm, as long as a relationship between temperature and frequency can be determined (i.e., a relationship to replace equation 1). The approximations will likely affect the absolute values of our quantitative evaluations, but are unlikely to affect the qualitative insights we have derived.

For example, consider diffusion effects. If structures with the highest power densities are surrounded by cooler structures, then it is possible that our evaluation will misidentify the hottest structures. However, this is a function of the chip floorplan on which designers do not always have full control. Additionally, we have confirmed that the register file and functional units are thermal hot spots in at least some commercial processors [5, 7]. Thus, our evaluations have identified reasonable hot spots, even if the absolute temperatures may have some error, and the qualitative results likely hold. It is of course possible that for some applications and floor plans, the register file and functional units are not the hottest structures on chip. In that case, different localized response mechanisms targeted at different potential chip hot spots need to be explored. We also note that the performance advantage of the predictive (vs. reactive) algorithms (as shown in Figure 3) is quite significant, largely due to the availability of additional high-overhead response mechanisms. It is unlikely that this large benefit is completely misrepresented by our temperature model approximations. Nevertheless, more accurate thermal modeling is a crucial area for further research.

7. Conclusions

To save on packaging, heat sink, and cooling solution costs for general-purpose processors, dynamic thermal management techniques (DTM) have been proposed. However, these techniques result in a performance degradation when invoked. This paper proposes new DTM algorithms for processors, targeted towards the increasingly important workload of multimedia applications. In contrast to current DTM schemes which are all reactive in nature, we propose *predictive* DTM algorithms that exploit certain properties of multimedia applications. We find that our predictive algorithms perform significantly better than existing reactive DTM algorithms, performing at least twice as well as reactive algorithms for the strictest thermal limit and up to 3.6 times as well in some cases. This is because the predictive algorithms can efficiently use high time overhead adaptations like register file resizing and DVS which result in significant thermal benefit for a limited loss in performance. Also, our analysis found that there is very little variation in the temperature profiles of several of our multimedia applications. This implies that the hardware configurations chosen by our predictive algorithms will generally remain close to optimal for the entire application execution run.

We also evaluate the effectiveness of different DTM response mechanisms. For architectural adaptation, for both reactive and predictive algorithms, we found that instruction window and functional unit adaptation performed better than fetch-toggling. The register file was the hot spot in our system. Local adaptations like functional unit adaptation and predictive instruction window resizing which directly target register file power resulted in much smaller performance degradation than chip-wide techniques like fetch-toggling (which reduce chip activity). For similar reasons, architecture adaptation performed better than DVS for many cases although the combination of the two gave the best results. Overall, our results show the importance of tailoring the DTM responses to the thermal hot spots and thermal limits of the system.

DTM is still a nascent field and there are many promising avenues of future work. Section 6 details some of this work.

8. REFERENCES

[1] In *International Technology Roadmap for Semiconductors, http://public.itrs.net/*, 2002.

[2] D. Brooks et al. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *ISCA*, 2000.

[3] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *HPCA*, 2001.

[4] H.-H. Chu and K. Nahrstedt. CPU Service Classes for Multimedia Applications. In *Proceedings of IEEE Multimedia Computing and Systems*, 1999.

[5] J. Deeney. Thermal modeling and measurement of large high power silicon devices with asymmertic power distribution. In *International Symposium on Microelectronics*, 2002.

[6] S. Dropsho et al. Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power. In *PACT*, 2002.

[7] S. H. Gunther et al. Managing the impact of increasing microprocessor power consumption. In *Intel Technology Journal, 1st Quarter*, 2001.

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

[9] M. Huang et al. A Framework for Dynamic Energy Efficiency and Temperature Management. In *MICRO*, 2000.

[10] C. J. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *ISCA*, 2001.

[11] C. J. Hughes et al. RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors. *IEEE Computer*, Feb 2002.

[12] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In *MICRO*, 2001.

[13] Intel XScale Microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[14] C. E. Kozyrakis and D. Patterson. A New Direction for Computer Architecture Research. *IEEE Computer*, November 1998.

[15] K. Skadron et al. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *HPCA*, 2002.

[16] K. Skadron et al. HotSpot: Techniques for Modeling Thermal Effects at the Processor-Architecture Level. In *THERMINICS*, 2002.