Voltage and Frequency Control With Adaptive Reaction Time in Multiple-Clock-Domain Processors

Qiang Wu Philo Juang Margaret Martonosi Douglas W. Clark
Depts. of Computer Science and Electrical Engineering
Princeton University
{jqwu, pjuang, mrm, doug} @princeton.edu

ABSTRACT

Dynamic voltage and frequency scaling (DVFS) is a widely-used method for energy-efficient computing. In this paper, we present a new intra-task online DVFS scheme for multiple clock domain (MCD) processors.

Most existing online DVFS schemes for MCD processors use a fixed time interval between possible voltage /frequency changes. The downside to this approach is that the interval boundaries are predetermined and independent of workload changes. Thus, they can be late in responding to large, severe activity swings. In this work, we propose an alternative online DVFS scheme in which the reaction time is self-tuned and adaptive to application and workload changes. In addition to designing such a scheme, we model the proposed DVFS control and use the derived model in a formal stability analysis. The obtained analytical insight is then used to guide and improve the design in terms of stability margin and control effectiveness.

We evaluate our DVFS scheme through cycle-accurate simulation over a wide set of MediaBench and SPEC2000 benchmarks. Compared to the best-known prior fixed-interval DVFS schemes for MCD processors, the proposed DVFS scheme has a simpler decision process, which leads to smaller and cheaper hardware. Our scheme has achieved significant energy savings over all studied benchmarks (19% energy savings with 3% performance degradation on average, which is close to the best results from existing fixed-interval DVFS schemes). For a group of applications with fast workload variations, our scheme outperforms existing fixed-interval DVFS schemes significantly due to its adaptive nature.

Overall, we feel the proposed adaptive online DVFS scheme is an effective and promising alternative to existing fixed-interval DVFS schemes. Designers may choose the new scheme for processors with limited hardware budget, or if the anticipated workload behavior is variable. In addition, the modeling and analysis techniques in this work serve as examples of using stability analysis in other aspects of high-performance CPU design and control.

1. INTRODUCTION

Various Dynamic Voltage and Frequency Scaling (DVFS) schemes have been proposed in the literature to improve the energy efficiency of high-performance processors, for example [11, 14, 19, 24]. Among all DVFS schemes, intra-program DVFS can take advantage of program phase changes during execution and achieve higher energy efficiency as compared to OS-level inter-program DVFS schemes. Among intra-program DVFS methods, hardware-based online DVFS schemes are driven by dynamic workloads, and thus are more applicable than static profile-based offline ones. In this paper, we design, model, analyze, and evaluate a new intra-program online DVFS scheme with adaptive reaction time in the context of multiple clock domain (MCD) processors [15, 20].

Most existing online DVFS schemes use a fixed time interval to frame possible voltage/frequency changes. Specifically, during the time interval, certain system metrics or statistics are monitored, such as IPC [8] or average issue queue occupancy [19, 23]. At the end of the interval, the statistics from the current and past intervals are used to compute a new voltage/frequency setting for future intervals. One limitation of the above schemes is that the interval boundaries are predetermined and independent of workload changes. Thus, no matter what severe workload change occurs, the fixed-interval approaches wait and attempt to adjust voltage/frequency at the end of the interval. In addition, they might miss opportunities to respond to large activity swings inside the interval. One simple scenario for this case is that the workload increases dramatically in the first half-interval and decreases in the second half. Average statistics (for example the average queue occupancy) over the entire interval may not be able to capture this workload change.

Compared to the above fixed-interval schemes, in this work we propose an online DVFS scheme in which the reaction time for DVFS is not predetermined and is instead determined by the actual workload variation. In other words, the instant to react is *adaptive* to large, severe workload changes. So it could be more responsive. On the other hand, given no or only minor workload changes, an *adaptive* scheme will stay inactive for an arbitrarily long time. So it could also be cost-effective.

We design our adaptive online DVFS scheme in the context of a Multiple Clock Domain (MCD) architecture, which has been shown to be a promising alternative to today's synchronous architecture and a better platform for DVFS control [15]. (A brief review of MCD is provided in Section 2.) The triggering condition is based on recent instant queue occupancy for the issue queues. More specifically, two queue signals are monitored at each sampling period. The first one is the relative queue occupancy value with respect to a reference value, while the second one is the difference of queue occupancies between two sampling points. Based on these queue signals, appropriate DVFS decisions are made through a simple process, which also uses deviation window and resettable time delay to handle the noise problem and avoid unnecessary DVFS actions. After an initial design is done, we wish to obtain some analytical insights on whether or how the designed DVFS system will work and how to improve it. So we derive an aggregate continuous model for the designed DVFS controller and use it in a formal stability analysis. The analytic insight obtained is then used to guide and improve the design in terms of stability margin and control effectiveness. Finally, we evaluate our DVFS scheme through a cycle-accurate MCD simulator over a wide set of MediaBench and SPEC benchmarks.

Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture (HPCA-11 2005)
1530-0897/05 $20.00 © 2005 **IEEE**

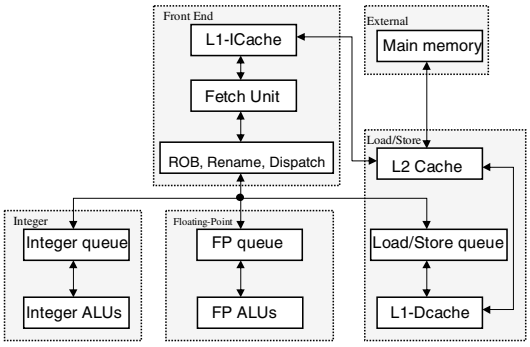

Figure 1: The clock domain partitions in an MCD processor by Semeraro *et al* [20].

Overall, we feel the main contribution of this work is threefold. First, rigorous modeling and stability analysis techniques have been applied to the design to provide insight and guidance, and make the design more efficient and more resilient. Second, the decision process in this design is very simple. This leads to smaller and cheaper hardware. So this scheme is useful for processors with limited hardware budgets. Third, for a group of applications with fast workload variations, the current scheme outperforms existing online DVFS schemes for MCD significantly due to its self-tuning reactive nature (on average, 18% better than [23], and nearly 3 fold better than [19]). So this scheme is also useful and suitable to processors where the type of application behavior (i.e. rapid workload variations) is known in advance.

The rest of the paper is structured as follows. In Section 2, we give a brief review of the MCD processor design and implementation. Section 3 describes the detailed design of our adaptive DVFS scheme. In Section 4, we model the designed DVFS control and gain insights through a formal stability analysis. This is followed in Section 5 by experimental results. In Section 6, we highlight important related work. Finally, Section 7 offers our conclusions.

2. BACKGROUND: MULTIPLE CLOCK DOMAIN MICROPROCESSORS

Some computer architects and researchers have predicted that in order to overcome the increasingly severe problems of clock distribution and power consumption, future high-performance microprocessors may need to have multiple clock domains (MCD) or use some form of asynchrony [1]. MCD processors use the Globally Asynchronous Locally Synchronous (GALS) clocking style [15]. Each function block or domain operates with an independently generated clock, and synchronization circuits ensure reliable inter-domain communication.

Advantages of MCD processors include less clock distribution and skew burden, less power consumption due to the absence of a global clock tree, design modularity, and extra flexibility in DVFS control [15]. The primary disadvantage of MCD processors is the inter-domain communication and synchronization overhead. An interface circuit is needed if data passes between two domains.

One key design issue for an MCD processor is the choice of where to partition the clock domains. It is still an open research question as to how to partition in order to maximize the power performance benefit. Most existing MCD implementations use architectural functional blocks as natural boundaries for clock domains. For example, Figure 1 shows a 4-domain partition used by the MCD implementation by Semeraro *et al.* [20], which consists of the front end, integer processing core (INT), floating point processing core (FP), and load store unit (LS). The main memory is considered as an external separate clock domain not controlled by the processor (for more details see [20]). Another popular MCD implementation by Iyer and Marculescu [10] uses a 5-domain partition, which is similar to that in Figure 1 but with the front-end split into two clock domains.

Another key design issue for an MCD processor is the synchronization interface design. A good interface design needs to have low latency, high throughput, and virtually no synchronization failure (i.e., no metastability). Nearly all of the existing MCD interface designs use some kind of queue structures for efficiency. One group of designs [6] uses token-ring based FIFOs, which have a very low latency and low synchronization overhead (there is no synchronization cost if the token-ring FIFO is neither full nor empty). Another group of designs uses arbitration-based queue structures (often with a stoppable clock) [21, 25]. The designs in this group are typically failure-free, but may need to check synchronization for each data transfer. For example, the design by Sjogren and Myers [21] includes arbitration and synchronization circuits which can detect whether the source and destination clock edges are far enough apart (i.e., greater than the so-called synchronization window size in [19]), in order for the source generated signal to be successfully accessed at the destination. This design has been used by the MCD implementation in [20]. Note that for a situation like that in Figure 1, where issue queues already exist between some domains, the interface queue structure can be integrated with the existing issue queue to form a combined issue/interface queue structure.

The MCD implementations in [10] and [20] also provide the capability of independently configuring the frequency and voltage in each clock domain. An aggressive XScale-style DVFS model is assumed, in which a clock domain can execute through the DVFS transition and there is no or very little idle time for the domain waiting for the PLL [7]. In addition, a relatively fast DVFS transition speed is assumed (around $1\mu s/20mv$) in [19] based on some reported industrial numbers [7]. Also the XScale-style model may allow any frequency to be used within the allowable range.

The quantitative benefit/overhead studies in [10, 20] have shown that an MCD processor has the potential to achieve significant power and performance efficiency. However, many design and control issues remain open and need more investigation in order to fully take advantage of the power/performance benefits brought by the MCD processors [15]. Online DVFS is one of these issues, as we will show next.

3. DESIGN OF ADAPTIVE DVFS SCHEME

One key design issue for adaptive DVFS is how sensitive it should be in responding to workload changes. Another design question is how big the frequency/voltage adjustment should be for a triggered action. So, before proceeding to an actual design, we first give some rationale and discussion of these key design issues.

As mentioned in the introduction, a variable-interval DVFS scheme responds immediately to large, severe workload changes. On the other hand, it will stay inactive for an arbitrarily long time given no or only minor workload change. Therefore, in general, the number of voltage/frequency adjustments for an adaptive DVFS scheme will ultimately depend on the pattern of workload change in a program. (Note this is different from that for the fixed-interval DVFS schemes where only one adjustment is possible for a fixed time interval.) However, for a given program, the number of adjustments will also depend on the setting of triggering conditions: what size of workload change should be treated as *severe* enough to trigger an action?

The main reason for which we want to respond only to severe workload changes is the DVFS switching cost, which includes both

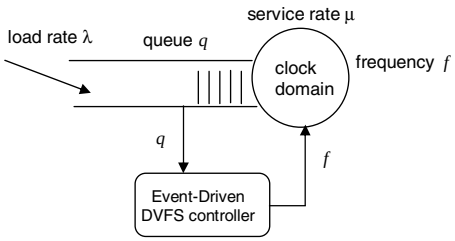

Figure 2: A local control model for a clock domain with an input queue.

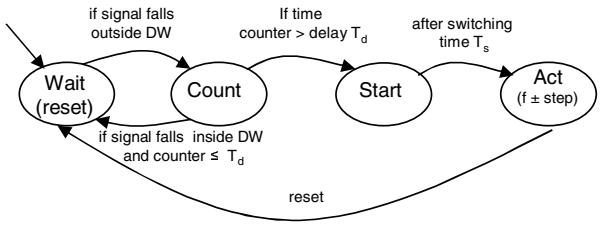

Figure 3: The state transition graph for our DVFS control, where the *signal* refers to the queue signal ($q_i - q_{ref}$) or ($q_i - q_{i-1}$); DW is the deviation window.

time and energy cost. (The switching cost is typically proportional to the magnitude of the switching.) The energy cost part for the DVFS switching comes from the voltage regulator (VR) [4]. Since the capacitor in VR is relatively small (most VRs are dual-phased so there are two output capacitors), the switching energy cost is small and is ignored in most DVFS studies [11, 14, 19, 23]. The time cost is the main concern in DVFS switching. This is basically the transition/switching time and may include some amount of idle time for the processor waiting for PLL relocking [7].

Because of this DVFS switching cost, the adaptive DVFS action should be triggered only for large workload changes (in terms of magnitude and duration) such that the benefit brought by the DVFS is greater than the switching cost (i.e., there is a net gain in terms of energy-delay product improvement). Therefore, for adaptive DVFS control, the choices of the triggering condition and the amount of adjustment at each action should be based on the DVFS switching cost. For a DVFS implementation with relatively fast transition time and no (or very little) processor idle time (denote this group as XScale-style DVFS [7]), the triggering condition and adjustment step can be chosen as relatively low or small in order to have more frequent and fine-grained DVFS control. On the other hand, for a DVFS implementation with relatively slow transition time and long processor idle time (denote this group as Transmeta-style DVFS [19]), the triggering condition and adjustment step should be chosen as relatively high or big in order to reduce the switching overhead; and we will have less frequent and more coarse-grained DVFS control for this case.

3.1 A Design for MCD Processors

Conceptually, the online DVFS problem for an MCD processor is to adapt the frequency (and execution speed) to program phases and workload changes in each clock domain. In this work, we utilize interface queues to guide the DVFS control. Recall from Section 2 that there are interface queues between clock domains for synchronization. Intuitively, these queues give clues about the speed balance between the sender domain and the receiver domain. For example, an emptying queue may imply that the receiver is operating too quickly relative to the sender. Conversely, a filling queue may suggest that it is too slow. A stationary queue might indicate a perfect match between sender and receiver execution speeds. Therefore, an online DVFS controller can use this queue information, such as the fullness of queues and the rate of queue changes, to detect workload changes and respond to them by increasing or decreasing voltage and frequency in a clock domain.

In this work, we will only use local queue/domain information to direct DVFS control (a so-called decentralized control scheme). In other words, we assume the interactions between different queues and domains are weak and can be ignored. (This assumption is typically valid for an MCD implementation with relatively simple structure such as that in Figure 1.) A centralized DVFS scheme which utilizes all queue/domain information may work better, but is much harder to design, as it is still an open research problem.

Based on the above rationale, we will use the local queue information as trigger signals for the DVFS actions. Figure 2 shows such a local control model, where the circle represents a clock domain being controlled (such as a floating point function unit). The clock domain is connected to other domains (such as a decode/issue domain) through an interface queue (input or output) which has a finite size. The queue occupancy is sampled and used as a possible triggering signal by the DVFS controller, which controls the frequency (and hence the voltage) of the clock domain.

More specifically, our DVFS controller monitors two queue signals: the relative queue occupancy (q_i - q_{ref}) and the relative queue difference (q_i - q_{i-1}); where q_i represents the queue occupancy at the i_{th} sampling point, and q_{ref} is the reference queue occupancy, i.e. the target or nominal queue occupancy. A possible DVFS action will be triggered based on these two queue signals. To simplify our design, we use a single step of incrementing or decrementing the clock frequency (and voltage accordingly) as the triggered DVFS action. As mentioned earlier, in general, the choice of the step size (and the time delay, which will be defined shortly) depends on the DVFS switching cost. For the MCD processor with an XScale-style DVFS model described in Section 2, we will choose relative small step sizes in order to have more fine-grained frequency adjustments. For processors with Transmeta-style DVFS model, the design framework in this section can still be used, but larger values should be chosen for the step size.

To handle the noise or random short-time variation in the queue occupancy and avoid unnecessary DVFS actions, we use a combination of deviation windows and time-delay relay. The deviation window (DW) is a small interval around the origin (denoted as $[-DW, +DW]$), while the time-delay relay is essentially a resettable time counter. A signal will activate the time counter if the signal falls outside the deviation window. If a pre-set time-delay has passed, a possible DVFS action will be triggered. Note, for the time-delay, there are a number of design options. For example, it can be set as a simple constant (T_{d0}) , or a constant with a scaling factor depending on some system statistics. In our design, we design it as a constant with a scaling factor depending on signal values and current frequency setting.

Figure 3 shows the state transition graph for our DVFS design. In the figure, the *signal* refers to a queue signal (either $q_i - q_{ref}$ or $q_i - q_{i-1}$). Initially, the system is in a *Wait* state. A transition into *Count* will occur if the queue signal falls outside the deviation window. The system will stay in the *Count* state until either the preset time delay (T_d) has passed and a transition is made to the *Start* state, or the queue signal falls inside the deviation window before the time delay has passed and a transition is made to the initial *Wait* state, which will reset the time counter. The *Start* state represents that a frequency (voltage) increment/decrement has been triggered or scheduled. Since it takes a certain amount of time to physically

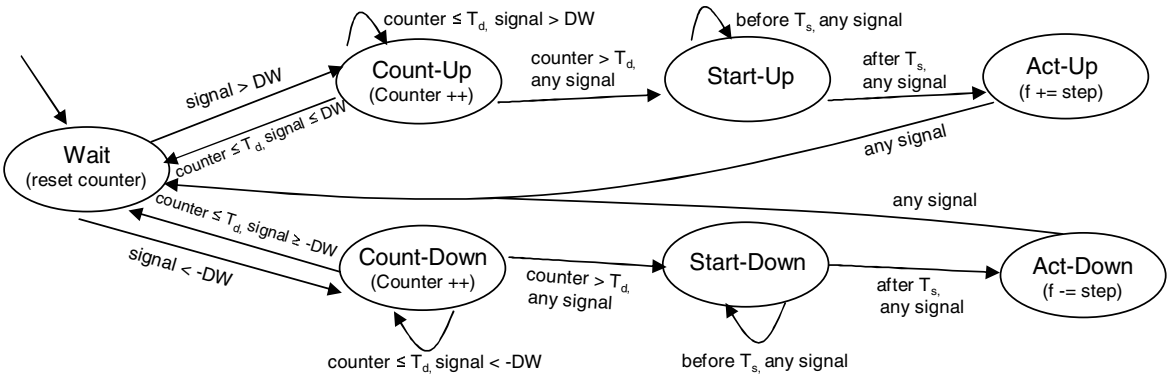

Figure 4: The detailed finite state machine (FSM) graph for our DVFS control, where the *signal* refers to the queue signal ($q_i - q_{ref}$) or ($q_i - q_{i-1}$); DW is the deviation window; T_d is the time delay; T_s is the switching time spent for one DVFS action.

switch frequency/voltage, the increment/decrement action will be accomplished (in the *Act* state) after a switching time T_s. After that, the system transitions back to the *Wait* state, resets the time counter, and is ready for a new round of operations. The detailed finite state machine (FSM) graph for our DVFS controller is shown in Figure 4, where we shown increment and decrement actions separately.

In the above descriptions, for the sake of clarity, we assume that the two finite state machines (FSM), one for the trigger signal $(q_i - q_{ref})$ and one for the signal $(q_i - q_{i-1})$, operate independently. However, in practice, methods are required to reconcile the *Act* operations triggered by different queue signals in different FSMs. So we add a new state called *Schedule* into the above operation state graphs. If, at any time, only one queue signal is triggering the *Act* operation, then the *Scheduler* will initiate and *start* the action in the same way as we described before. On the other hand, if two queue signals are triggering DVFS actions at the same time, the system will schedule the two actions depending on the actions being triggered. Specifically, if two identical actions are being triggered (both are *Up* or both are *Down*), the system will schedule the two actions in sequence (or combine them into one action with a step size twice as big). On the other hand, if two opposite actions are being triggered (one is *Up* while the other one is *Down*), the system will cancel them all and reset both signals to the *Wait* state.

In the above design the reference queue value q_{ref} can be any value which is neither full nor empty. However, similar to the observation in [23], we notice that the position of q_{ref} specifies the actual tradeoff between performance degradation and energy saving. We can increase q_{ref} to make the DVFS controller more aggressive in saving energy, or decrease q_{ref} to preserve performance more. The main reason is that the choice of the nominal operating point q_{ref} and its distances to the two queue end-points reflect the relative margin for the queue to tolerate control errors and input noise before the queue becomes full (will lose performance) or empty (will waste energy) – see [23] for details.

Next, we look at the issue of hardware implementation. From the description of the above design, we see the adaptive DVFS decision logic requires very little hardware in addition to the existing MCD hardware, since the interface queues and the voltage/frequency switching mechanism already exist in current MCD implementations [20]. The main additions are some book-keeping hardware. Figure 5 shows the block diagram for a possible hardware implementation of the basic DVFS decision logic. Specifically, an adder is used to compute the trigger signal (either $q_i - q_{ref}$ or $q_i - q_{i-1}$). Since a queue size is around 20 ($\ll 2^6$), a 6-bit adder is sufficient. Then, a 7-bit comparator is used to compare the trigger signal with the deviation window (DW). The rest is a 5-state finite state machine (FSM) and a time-delay counter. The FSM corresponds to the left five states in Figure 4. For the counter, if we assume the time delay in Figure 4 is ≤ 256, an 8-bit time-delay counter is sufficient. The two output signals of the FSM, *Start-up* and *Start-down*, will be used by the voltage (and frequency) switching mechanism in the voltage regulator. Note the hardware requirement for this scheme is roughly in the same order as the book-keeping hardware (like counters) required by the fixed-interval DVFS schemes in [19, 23]. However, in [19, 23], some extra hardware is required to compute appropriate voltage and frequency settings on a per-interval basis. This extra hardware is more complex than the book-keeping hardware discussed above (for example, multipliers/dividers or lookup tables are required to implement the PID controller in [23]). So, overall, the hardware requirement for the present scheme is much smaller and cheaper than those in [19, 23].

4. MODELING AND STABILITY ANALYSIS OF ADAPTIVE DVFS SYSTEM

In this section, we will look at the adaptive DVFS design from a control system point of view. The modeling and stability analysis are intended to gain insights and answer questions such as

- Will the above design work? Under extreme cases, can it lead to unbounded or unstable results?
- If it works, how well does it work and how can it be improved?
- More specifically, in order to improve the stability margin and control effectiveness, how should we choose the design options/parameters like the basic time delays? (There are two queue signals, so there are two time delays to pick. Should we use the same amount of delay for both? If not,which one should be bigger or smaller?)

4.1 Overview of Modeling and Stability Analysis

This subsection gives an overview of modeling and analysis of the adaptive DVFS system without going through the derivation and stability analysis details, which will follow.

In order to analytically evaluate the DVFS design in the previous section, we need to model the DVFS control operation and the involved queue and clock domain dynamics. As mentioned earlier, we use the local control model in Figure 2. The clock domain has frequency f and execution capability (or service rate) μ, which is a function of f. The workload (or arrival rate) is denoted as λ. Note

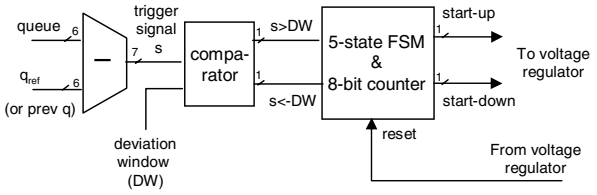

Figure 5: Block diagram for a possible hardware implementation of the basic DVFS decision logic.

both μ and λ are time varying and we will denote them as $\mu_{(t)}$ and $\lambda_{(t)}$. Similarly, we denote the queue occupancy at time t as $q_{(t)}$. With these notations, we are ready to derive a model.

Recall from Section 3 that our DVFS controller for MCD processors has fine-grained *step* control. We can conveniently approximate the DVFS control by a continuous-time model[1]. This model is expressed as

$$\dot{f}_{(t)} = \frac{m}{h_{(f)}} \frac{step}{T_{m0}} (q_{(t)} - q_{ref}) + \frac{l}{h_{(f)}} \frac{step}{T_{l0}} \dot{q}_{(t)} \qquad (1)$$

Intuitively, the above equation models the aggregate effects of the frequency control operation in Figures 3 and 4. The first part of the right hand side of the equation corresponds to frequency control operation for the queue signal $(q_i - q_{ref})$, while the second part corresponds to the operation for the queue signal $(q_i - q_{i-1})$. Specifically, in the equation, $\dot{f}$ is the time derivative of normalized frequency f (i.e. $\partial f/\partial t$); $q_{(t)}$ and q_{ref} are the queue and reference queue occupancy as in Section 3; $\dot{q}$ is the time derivative of queue occupancy; *step* is the step size of the frequency change triggered by the queue signals; T_{m0} and T_{l0} are the basic time delays for the queue signals $(q_i - q_{ref})$ and $(q_i - q_{i-1})$ respectively; m and l are constants which are mainly from the conversion due to the different units used for queue occupancies and frequencies; $h_{(f)}$ is a function of f which is used to take account of possible affects of f on the effective time delay.

We then derive a model for the the queue and clock domain dynamics. At an aggregate level, they are modeled as

$$\begin{aligned} \dot{q}_{(t)} &= \gamma\, \lambda_{(t)} - \gamma\, \mu_{(t)} \\ \mu_{(t)} &= \frac{1}{t_1 + \frac{c_2}{f_{(t)}}} \end{aligned} \qquad (2)$$

Intuitively, the first equation in (2) means the queue occupancy change in a sampling time unit is equal to the number of arrived elements minus the number of departed/executed elements in that unit time (this is essentially a continuous-time version of the well-known Lindley equation [13]); γ is a constant which is proportional to the size of sampling period. The second equation models the execution/service rate μ in terms of clock frequency f, which is essentially a continuous-time version of the model used in [23, 24]. The t_1 and c_2 are constants whose meaning will be explained in detail in Section 4.3.

Putting together (1) and (2), we have a complete model for the involved DVFS controller, queue, and clock domain dynamics. This model is inherently nonlinear. To simplify the stability analysis, we linearize the system model through a standard nonlinear transformation technique [2] (which is essentially done by choosing an appropriate $h_{(f)}$ in (1) to compensate for the nonlinear function in (2) – details in Section 4.3). Then we proceed with some classic stability analysis for the linearized DVFS control system. Through the stability and control performance analysis, we have obtained the following:

- Remark 1: Given any non-zero values of *step* and basic time delays, the DVFS control system in Section 3 is stable. So, for any workload inputs, the DVFS controller would not lead to unbounded or unstable results.

- Remark 2: The control effectiveness of our design, in general, is mostly dependent on the values of time delays. A smaller time delay tends to improve the control response and settling time, and thus increase the control effectiveness. On the other hand, a smaller time delay will weaken the system's noise rejection ability, which may lead to more unnecessary/incorrect DVFS actions and thus reduce the overall DVFS efficiency. So there is a tradeoff between the control effectiveness and the system's noise rejection ability.

- Remark 3: In order to have small percent transient overshoot in system response, the values of the time delay for those two queue signals should be constrained by an inequality constraint (details in Section 4.3). With a typical system setting, this constraint implies that the time delay for the signal $(q_i - q_{ref})$ should be relatively larger than that for $(q_i - q_{i-1})$, and a setting of 2-8 time larger would typically lead to fairly good results.

In the rest of this section, we will give the derivation and analysis details. People who are already familiar with these subjects or who are not interested in these details may wish to skip them and go directly to the experimental evaluation in Section 5.

4.2 Modeling the Adaptive DVFS controller

The DVFS controller in Section 3 is essentially a discrete device with some inherent deviation windows and adjustable time delays. We will derive a model to capture the aggregate effect of its adaptive operations. We start by doing it for the queue signal $(q_i - q_{ref})$ only. Later, we will give a complete model for DVFS operations with both queue signals.

In aggregate, the frequency control operation with the queue signal $(q - q_{ref})$ in Figure 3 and 4 can be modeled as

$$f_{i+1} = f_i + step \cdot I(q_i - q_{ref}) \qquad (3)$$

where

$$I(q_i - q_{ref}) = \begin{cases} 1 & \text{if } (q_i - q_{ref}) > DW \text{ for } (T_d + T_s) \\ -1 & \text{if } (q_i - q_{ref}) < -DW \text{ for } (T_d + T_s) \\ 0 & \text{Otherwise} \end{cases}$$

Intuitively, the above equation means a frequency increment or decrement will occur if the queue signal $(q_i - q_{ref})$ falls outside the deviation window for a consecutive $(T_d + T_s)$ amount of time (in sampling period units). Otherwise, the frequency stays unchanged. In the equation, DW stands for Deviation Window; T_d is the time delay; T_s is the switching time; and others are defined the same as before.

In the DVFS design in Section 3, the discrete *step* was chosen to be relatively small in order to have more fine-grained frequency adjustments. With a small *step*, the above discrete model can be approximated by a continuous-time model as follows (we use a combined time delay $T_m = T_d + T_s$).

[1]For processors with Transmeta-style DVFS, which require more coarse-grained *step* control, the modeling and analysis in this section can still be applied but will be less accurate. A similar but more complicated discrete-time model can be derived to get a better and more accurate analysis result. We leave this as possible future work.

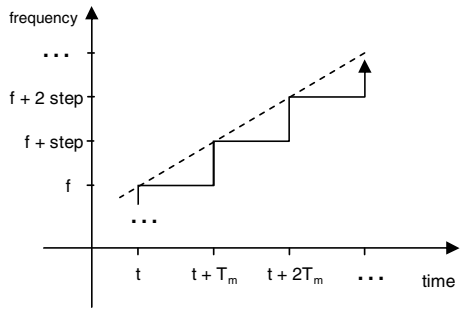

Figure 6: The approximation of the Step-up action modeled in (3) by a continuous-time linear increment action as modeled in (4).

$$\dot{f}_{(t)} = \begin{cases} \frac{step}{T_m} & \text{if } (q_{(t)} - q_{ref}) > DW \\ \frac{-step}{T_m} & \text{if } (q_{(t)} - q_{ref}) < -DW \\ 0 & \text{if } |q_{(t)} - q_{ref}| \leq DW \end{cases} \quad (4)$$

The above model captures the aggregate effects of the frequency control operation by approximating the discrete-time step up or down action in (3) with a continuous-time linear increment or decrement action as illustrated in Figure 6 (the slope of the line is $\frac{step}{T_m}$). In the equation, $\dot{f}_{(t)}$ is the derivative of the frequency f at time t.

As mentioned in the last section, we choose the time delay T_m as a constant with a scaling factor which is negatively dependent on the absolute value of the queue signal. That is, T_m is expressed as

$$T_m \ = \ T_{m0} \cdot \frac{1}{|q_{(t)} - q_{ref}| \cdot m} \quad (5)$$

The above equation means that when the value of the queue signal is larger, the time delay will be smaller and the DVFS controller will respond more quickly. Specifically, T_{m0} is the basic (constant) time delay; m is a constant which is mainly used to convert the value of queue occupancy to the scaling factor for time delay.

In addition, we augment the right side of equation (5) by a function $h_{(f)}$ to model possible impacts of frequency f on the time delay. We set $h_{(f)}$ to 1 if the level of frequency has no effect on the time delay.

Substituting the delay T_m in (5) into the continuous-time model in (4), assuming $DW \approx 0$, and rearranging algebraically, we have a new model in an ordinary differential equation (ODE) format as

$$\dot{f}_{(t)} \ = \ \frac{m}{h_{(f)}} \cdot \frac{step}{T_{m0}} \cdot (q_{(t)} - q_{ref}) \quad (6)$$

Like (4), the above equation models the aggregate effects of frequency control operation in Figures 3 and 4. However, compared to (4), this model is much simpler and more amenable to analysis.

We follow a similar procedure to derive another similar model for DVFS operations with the other queue signal $(q_i - q_{i-1})$. Since our DVFS controller is driven by both queue signals, we combine these two models to get a complete model for our DVFS controller.

$$\dot{f}_{(t)} \ = \ \frac{m}{h_{(f)}} \frac{step}{T_{m0}} (q_{(t)} - q_{ref}) + \frac{l}{h_{(f)}} \frac{step}{T_{l0}} \dot{q}_{(t)} \quad (7)$$

The above model captures the control effects of the DVFS operation with both trigger signals, $(q_i - q_{ref})$ and $(q_i - q_{i-1})$. (Note that this equation is the same as (1) in the overview sub-section and is repeated here for convenience). In the equation, $\dot{q}_{(t)}$ is derivative of queue value at time t; l and T_{l0} have similar interpretations as the m and T_{m0} in (6).

4.3 Stability Analysis of the Adaptive DVFS System

As described in the overview subsection, the modeling for the queue and clock domain dynamics is expressed by the following equations.

$$\dot{q}_{(t)} \ = \ \gamma \, \lambda_{(t)} - \gamma \, \mu_{(t)} \quad (8)$$

$$\mu_{(t)} \ = \ \frac{1}{t_1 + \frac{c_2}{f_{(t)}}} \quad (9)$$

Intuitively, the first equation means the queue occupancy change in a sampling unit time is equal to the number of arrived elements minus the number of departed/executed elements in that unit time. γ in the first equation is a constant proportional to the size of the sampling period. The second equation models the execution/service rate μ in terms of clock frequency f. This $\mu \sim f$ model is essentially a generalization of the model used in [23, 24], which is discrete-time and models the average execution speed of a clock domain as a function of the average frequency in a time interval. The model in [23, 24] is based on the observation that, in most clock domains, execution time can be separated into two parts, one that is independent of clock frequency and one that is dependent. For example, in a load/store domain, the time spent for accessing asynchronous memory due to a cache miss is independent of domain frequency, while the time for querying and accessing a cache is dependent on frequency. Accordingly, in this model, t_1 is the average amount of unit time per instruction that is independent of frequency, and c_2 is the average number of frequency-dependent cycles per instruction (the value of t_1 and c_2 can be estimated online or offline using methods similar to those in [11, 24]). In our model in (9), we generalize the model in [23, 24] into a continuous-time model by assuming that, at every sampling time unit, the $\mu_{(t)}$ and $f_{(t)}$ satisfies the same relationship as in the discrete $\mu \sim f$ model in [23, 24].

Putting together (7),(8) and (9), we have a complete model for the involved DVFS controller, queue, and clock domain dynamics. This model is inherently nonlinear. To simplify the stability analysis, we transform the equation (7) in terms of a new state variable of $\dot{\mu}_{(t)}$ through the equation

$$\begin{aligned} \dot{\mu}_{(t)} \ &= \ \frac{\partial \mu}{\partial f} \cdot \frac{\partial f}{\partial t} \\ &= \ \frac{c_2}{(t_1 f + c_2)^2} \cdot \dot{f}_{(t)} \end{aligned} \quad (10)$$

which is obtained by taking derivatives on both sides of (9). After this transformation, equation (7) becomes

$$\dot{\mu}_{(t)} \ = \ \frac{c_2}{(t_1 f + c_2)^2} \cdot \frac{1}{h_{(f)}} \left[\frac{m \cdot step}{T_{m0}} (q_{(t)} - q_{ref}) + \frac{l \cdot step}{T_{l0}} \dot{q}_{(t)} \right] \quad (11)$$

Like (7), this equation also models the frequency control in Figure 3 and 4, but is expressed in a different state variable $\mu_{(t)}$. We see that, in this equation, if we choose the function $h_{(f)}$ proportional to $\frac{c_2}{(t_1 f + c_2)^2}$, the non-linear part in it will be compensated for and the above equation becomes linear. Since the function $\frac{c_2}{(t_1 f + c_2)^2}$ is relatively complex to implement in practice, we will approximate it by a simpler quadratic function $\frac{k}{f^2}$ around the operating point. (Here, k is a constant factor dependent on the $\mu \sim f$ relationship; it can be estimated using t_1 and c_2 values.) Then, we can simply choose $h_{(f)} = \frac{1}{f^2}$ to linearize the equation (11).

Therefore, after the above linearization, we have a new system as

$$\begin{aligned} \dot{q}_{(t)} \ &= \ \gamma \, \lambda_{(t)} - \gamma \, \mu_{(t)} \\ \dot{\mu}_{(t)} \ &= \ \frac{m \cdot k \cdot step}{T_{m0}} (q_{(t)} - q_{ref}) + \frac{l \cdot k \cdot step}{T_{l0}} \dot{q}_{(t)} \end{aligned} \quad (12)$$

The above linearized system model is equivalent to the original system model in (7) - (9). Therefore, in order to understand the stability and transient behavior of our DVFS control system, we can conveniently analyze this linearized system model. By classic stability theory, the stability behavior of a linear system is decided by its characteristic roots [12]. For the system in (12), we can solve the characteristic roots as

$$s_{1,2} = \frac{-K_l}{2} \pm \frac{\sqrt{K_l^2 - 4K_m}}{2} \quad (13)$$

where $K_m = \frac{m \cdot \gamma \cdot k \cdot step}{T_{m0}}$ and $K_l = \frac{l \cdot \gamma \cdot k \cdot step}{T_{l0}}$.

Based on the above characteristic roots, we have the following observations and remarks.

Remark 1: With a typical system setting (non-zero parameters), the DVFS control system in Section 3 is stable. So, for any kind of workload inputs, our DVFS controller would not lead to unbounded or unstable results.

The above remark comes from the following. A linear system is stable if all its characteristic roots are negative (i.e. on the left side of the s-plane) [12]. With a typical system setting, all system parameters are non-zero and positive. So K_m, K_l will be positive. Then, from (13), we see both roots will be on the left side of the s-plane. Thus the DVFS control system is stable.

Remark 2: The control effectiveness of our design, in general, is mostly dependent on the value of time delays. A smaller time delay tends to improve the control response and settling time, and thus increase the control effectiveness. On the other hand, a smaller time delay will weaken the actual system's noise rejection ability (not modeled by the analytical model), which may lead to more unnecessary/incorrect DVFS actions and thus reduce the overall DVFS efficiency. So there is a tradeoff between the control effectiveness and the system's noise rejection ability.

The above remark is based on the following. Though any positive K_m and K_l values would make the system stable, the control effectiveness is dependent on the actual values of K_m and K_l. Among all the parameters in the K_m and K_l definitions, the basic time delays T_{m0} and T_{l0} are the most adjustable parameters in the design space. So, in general, the control effectiveness is mostly dependent on the values of the time delays. More specifically, the control effectiveness is typically characterized by the *settling time* (t_s) and the *rising time* (t_r) of the system unit-step response [12]. For this system, we have $t_s = \frac{8}{K_l}$ and $t_r = \frac{0.8}{\sqrt{K_m}} + \frac{1.25K_l}{K_m}$ using the formulas in [12]. So, smaller time delays T_{m0} and T_{l0} (thus larger K_m and K_l) will improve the rising and settling time, and increase the control effectiveness. On the other hand, in our DVFS system, the time delays are used to handle the noise (i.e. the random short time input variation) and avoid unnecessary DVFS actions. So, smaller time delays will weaken the system's noise rejection ability, and may lead to more unnecessary/incorrect DVFS actions. Therefore, the time delays should be chosen to balance the control effectiveness and the noise rejection.

Remark 3: In order to for the system to have relatively small percent transient overshoot in the system response, the values of the time delay for the two queue signals should be constrained by an inequality. With a typical system setting, this constraint implies that the time delay for the signal $(q_i - q_{ref})$ should be relatively larger than that for $(q_i - q_{i-1})$, and a setting of 2-8 time larger would typically lead to fairly good results.

The above remark is based on the following. In this system, the maximum percent transient overshoot is decided by a parameter called damping ratio $\xi = \frac{K_l}{2\sqrt{K_m}}$ [12]. To have a small percent overshoot (say $\leq 15\%$) and a good rising time also, we have an inequality constraint of $0.5 \leq \xi \leq 1$ for this system. Substituting the ξ in the above inequality with K_l and K_m, we have $\frac{K_l^2}{4} \leq K_m \leq K_l^2$. With a typical system setting, we have $K_l < 1$. Combining the above two inequalities, we have $K_m < K_l$. Assuming all other parameters (such as *step*) are the same for the two queue signals, the above inequality implies $T_{m0} > T_{l0}$. In other words, the time delay for the signal $(q_i - q_{ref})$ should be relatively larger than that for $(q_i - q_{i-1})$. For example, if $K_l = \frac{1}{2}$, we have $\frac{1}{16} \leq K_m \leq \frac{1}{4}$. So the time delay T_{m0} is 2-8 times larger than T_{l0}, which would typically lead to fairly good transient control response. Inside this range, we can take a value close to the upper end (8 times slower) if overshoot is the major concern, or take a value close to the low end (2 times slower) if rising time is the major concern.

So far, we have analytically evaluated the adaptive DVFS design in terms of stability and control effectiveness. In the next section, we will experimentally evaluate the DVFS design and check how well it works in practice.

Table 1: Summary of All Simulation Parameters

Simulation Parameters	Value
Domain frequency range	250MHz – 1.0GHz
Domain voltage range	0.65V - 1.20V
Frequency/voltage change speed	73.3 ns/MHz, 171 ns/2.86mV
Signal sampling rate	250MHz
Time delays (sampling)	$T_{l0} = 10$, $T_{m0} = 50$
Step size (f/v)	2.3MHz/2.86mv
Reference queue point	7 INT, 4 FP, 4 LS
Deviation window (DW)	± 1 or 0
Domain clock jitter	$\pm$110ps, normally distributed
Inter-domain synchro window	300ps
Branch predictor:	
2-level	L1 1024, hist 10, L2 1024
Bimodal	size 1024, BTB 4096 sets, 2-way
Combined	size 4096
Decode/Issue/Retire width	4/6/11
L1 data cache	64KB, 2-way
L1 instr cache	64KB, 2-way
L2 unified cache	1MB, direct mapped
Cache access time	2 cycle L1, 12 cycles L2
Memory access latency	80 first chunk, 2 inters
Integer ALUs	4 + 1 mult/div unit
Floating-point ALUs	2 + 1 mult/div/sqrt unit
Issue queue size	20 INT, 16 FP, 16 LS
Reorder buffer size	80
LS retire buffer size	64
Physical Register file size	72 INT, 72 FP

5. EXPERIMENTAL RESULTS

In this section, we will show experimental results for evaluation. We will compare our results to those from the conventional synchronous voltage/frequency scaling and to those from two existing fixed-interval DVFS schemes for MCD processors [19] [23]. At the end of this section, we will also compare our results to [23] with different and shorter interval lengths.

5.1 Simulation Methodology and Setup

Our simulation environment is based on that in [23], which is in turn based on the SimpleScalar toolset [5] with the Wattch [3] power estimation extension and the MCD processor extension [14, 20]. The MCD extension by Semeraro *et al.* in [14, 20] has 4 clock domains as shown in Figure 1.

We implemented our DVFS controller for local queues and domains, following the design and analysis in Sections 3 and 4 . Also, as in previous work on DVFS in MCD [14, 19, 23], we made the front end domain run at a fixed maximum speed, and allowed the INT, FP, and LS domains to be controlled by the DVFS controller.

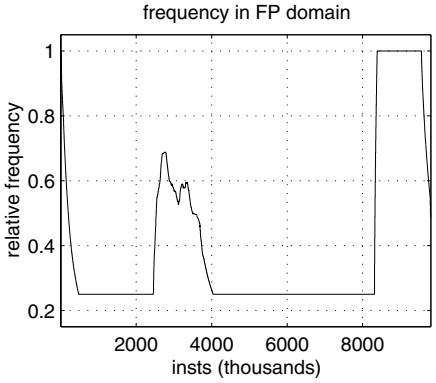

Figure 7: Frequency settings obtained from our adaptive DVFS in the FP clock domain for the benchmark of Epic-decode

.

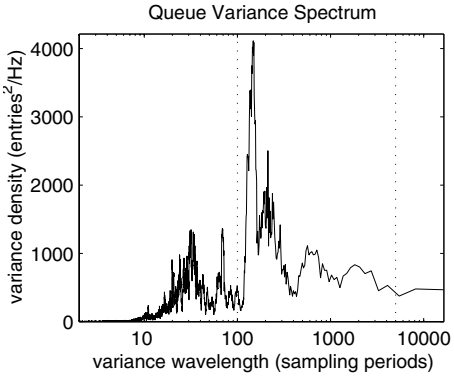

Figure 8: Variance spectrum for queue entries in the INT domain for the benchmark Epic-decode. The dotted line marks the interesting frequency/wavelength range used to identify *fast* workload variations.

Since we are assuming an aggressive XScale-like DVFS model as described in Section 2, we choose a fine-grained step size for the frequency increment or decrement (2.4MHz/step, so it takes 320 steps to traverse the total frequency/voltage range). We choose a sampling rate of 250MHz for the queue signals, which corresponds to the lower bound of the frequency range $250MHz - 1GHz$. Based on the remarks 2 and 3 in Section 4, we choose the basic time delays for the queue signals $(q_i - q_{i-1})$ and $(q_i - q_{ref})$ as $T_{l0} = 8$ and $T_{m0} = 50$ respectively (in units of sampling period). We emulate the signal-dependent time delay by having larger time-counter increments in Figures 3 and 4 for larger signal values. The time delay for *counting-down* is also scaled by a factor of $(\frac{1}{f^2})$, where f is the relative frequency using $f_{max} = 1.0GHz$ as the base. So, for *counting-down* in a clock domain, the time delay would be larger with a lower frequency level and the system would be more cautious in further scaling down the clock frequency. We choose a q_{ref} of 6 for the INT clock domain which is roughly $\frac{1}{3}$ of its total queue size; and a q_{ref} of 4 for the FP and LS clock domains which are $\frac{1}{4}$ of their total queue sizes. These numbers are chosen to make the overall performance degradation around 5%. For the deviation window DW, we choose $DW = \pm 1$ for the queue signal $q_i - q_{ref}$, and $DW = 0$ for the signal $q_i - q_{i-1}$. Also, we assume an aggressive clock gating that is applied whenever the unit is not used. All other architecture parameters are chosen to have the same values as those in [14, 19, 23]. A summary of all simulation parameters is shown in Table 1.

As an illustrative example, Figure 7 shows the frequency settings in the FP domain obtained from our DVFS controller for the MediaBench benchmark Epic-decode. We chose this benchmark because its FP issue queue has a very simple workload pattern, in which the queue is emptying except for two distinct phases, as observed in [19, 23]. From this figure, we see, in the beginning stage, the adaptive DVFS controller detected the queue emptiness and quickly reduced the FP frequency to $f_{min} = 0.25GHz$. The first non-empty workload phase occurred around the moment at 2500k instructions when a modest workload increase was detected and the frequency recovered from $f_{min} = 0.25$ to a value around 0.6. After this recovery, the frequency gradually dropped to $f_{min} = 0.25$ again as the queue decreased and became complete empty again around the moment at 4000k instructions. The queue stayed empty until a dramatic increase occurred around the moment at 8200k instructions. The adaptive DVFS controller detected this dramatic increase of queue entries and quickly adjusted the clock frequency to $f_{max} = 1.0GHz$.

5.2 Benchmark Classifications

We want to evaluate our adaptive DVFS scheme with a broad set of applications. To show variety, we will present results for 6 MediaBench, 6 SPEC2000int, and 5 SPEC2000fp applications as shown in Table 2. We chose roughly the same subset of SPECint and SPECfp as those used in [14, 19, 23]. For MediaBench benchmarks, we use the official data input set in the MediaBench web site and the whole program as the simulation window; for SPEC2000 benchmarks, we use the reference input set and choose the simulation window using the published Early SimPoint numbers [17].

Before we start to show the energy/performance results from our DVFS scheme and compare them to those from prior fixed-interval DVFS schemes, we want to first look at some benchmark characteristics which affect the performance of the current and prior online DVFS schemes. As mentioned in Section 1, the current scheme has potentially better responsiveness due to its adaptive nature and is more suitable for applications with fast workload changes. Therefore, one benchmark characteristic we might wish to look at is the workload variability. In the rest of this subsection, we will study this characteristic and identify applications with relatively fast workload variations.

We use the queue occupancies to characterize the program workload. A metric which we can use to justify the application workload as varied is the queue occupancy variance. However, there is a problem with this metric: it only reflects the total workload variations, and not necessarily the *fast* variation. To overcome this problem, we make use of spectrum or spectral analysis [16].

The spectrum of a time series is the distribution of variance as a function of variance frequency [16] (denote the workload variance frequency as ω to distinguish it from the clock frequency f). Note the basic components for spectrum are the sinusoidal waves of different frequency or wavelengths. Spectral analysis estimates and computes the variance associated with each frequency component (The method we use is the Multi-taper method [16] which utilizes the famous fast Fourier transform during the estimation process).

With the spectrum or spectral density function (spectral density is in terms of variance per unit frequency), we can get the variance associated with any range of frequencies by integrating the spectral density over the increment of frequency ω. Therefore, in order to identify fast workload variations, we can compute and inspect the queue variance associated with high frequencies or short wavelengths only. The question is how to quantify *fast*, *high* or *short*.

Since we are studying adaptive DVFS as compared to fixed-interval DVFS (which is assumed to have a fixed interval of length

Table 2: Classification of benchmarks based on workload variation characteristics (numbers are in units of queue entry square)

Benchmarks	Queue variance			Classification
	INT	FP	LS	
epic-decode	17.2	0.1	7.3	
jpeg-decode	12.4	0.0	8.0	
jpeg-encode	14.4	0.0	10.1	*Group1*
172.mgrid	1.7	14.5	5.7	(fast
173.applu	0.8	16.8	8.3	workload
176.gcc	19.1	0.0	7.1	variation)
183.equake	16.7	0.0	5.1	
186.crafty	14.4	0.0	7.0	
197.parser	13.6	0.0	6.6	
adpcm-decode	2.9	0.0	0.5	
adpcm-encode	7.6	0.0	0.8	
epic-encode	4.1	3.1	1.6	*Group2*
164.gzip	9.4	0.0	4.5	(slow or
171.swim	1.0	1.8	1.3	negligible
179.art	4.2	4.6	2.9	variation)
181.mcf	5.1	0.0	4.1	
256.bzip2	13.3	0.0	2.1	

N, $N = 5 \sim 10k$ sampling periods normally), we define any queue variance component with a wavelength of N or less as *fast* or *short*. This is because any queue swings inside the control interval (i.e. with a wavelength of N or less) will not be captured by the fixed-interval schemes (as the swings offset by the averaging in the fixed-interval schemes). In addition, queue variance components with extremely short wavelengths are considered noise and are ignored (i.e. they are too fast to be captured by either approach). We choose 100 sampling periods as the noise drop-off line because the basic time-delay for our DVFS scheme is about 50.

Therefore, to identify fast workload variations, we compute and inspect queue variance with wavelengths in the range of $[100, N]$, where N is the interval length for the fixed-interval schemes. As an illustrative example, Figure 8 shows the variance spectrum of the queue entries in the INT domain for the benchmark Epic-decode (For convenience, we show the spectrum as a function of the wavelength, rather than the frequency ω). The queue variance in the interesting frequency range (marked by the dotted line in Figure 8) can then be computed.

In Table 2, we have computed the queue variance associated with the defined interesting frequency/wavelength range for all the benchmarks. Based on these variance numbers (larger number means more workload variation), we classify the benchmark set into two groups with roughly same number of applications: *group1* with relatively large workload variations and *group2* with relatively small or negligible variations in the defined frequency range. (Specifically, in table 2, INT and LS numbers are used to classify integer benchmarks; while FP and LS numbers are used to classify floating-pointing benchmarks.)

We believe that the above benchmark characterization and classification will let us have a better and deeper understanding of the experimental results in the next subsection.

5.3 Energy and Performance Results

We will present energy and performance results from our new adaptive-reaction DVFS scheme (denoted as *adaptive*), and compare them to existing fixed-interval DVFS schemes for MCD processors [19, 23]. The work in [19] is one of the best-known DVFS schemes for MCD, and is based on a heuristic called *AttackDecay* (denoted as *fixed-heuristic*). The parameters for *fixed-heuristic* here are taken from [19], with a 5% performance degradation target because this target value will lead to an actual performance degradation similar to *adaptive*. The other work in [23] is a more recently proposed DVFS scheme for MCD which uses a Proportional Integral Derivative (PID) based DVFS controller (denoted as *fixed-PID*). Similarly, the parameters for *fixed-PID* are taken from [23]. For the sake of completeness, we also compare our results to conventional (fully) synchronous voltage/frequency scaling (denoted as *Synchro*) which scales the frequency/voltage for the whole processor[2].

Table 3: Average results over *group1* benchmarks

Schemes	Performance degradation	Energy savings	Energy-Delay product improvement
Synchro	4.7%	7.0%	2.6%
fixed-heuristic	5.9%	9.6%	4.3%
fixed-PID	3.2%	16.2%	13.5%
adaptive	3.3%	18.7%	16.0%

Table 4: Average results over *group2* benchmarks

Schemes	Performance degradation	Energy savings	Energy-Delay product improvement
Synchro	3.9%	8.4%	4.8%
fixed-heuristic	5.6%	16.3%	11.6%
fixed-PID	3.7%	24.5%	21.7%
adaptive	3.2%	19.7%	17.1%

The performance degradation, energy savings, and energy-delay product (EDP) improvement for each benchmark are shown in Figure 9. (Note that the benchmarks are arranged with *group1* on the left, and *group2* on the right.) The results are relative to the conventional (fully) synchronous processor without voltage scaling. So all results for MCD include about 1.5% percent inherited performance and energy overhead from the baseline MCD processor, as discussed in [19]. The average energy savings and performance loss for *adaptive* over all 17 benchmarks are 19.2% and 3.3% respectively. This gives an EDP improvment of 16.5% for *adaptive*, as compared to 3.9% for *Synchro*, 7.6% for *fixed-heuristic*[3], and 17.3% for *fixed-PID*. So we see, despite its simpler design and hardware requirement, the *adaptive* results on average are quite close to the best from existing approaches (16.5% vs. 17.3%)

Furthermore, if we look at the energy/performance results for individual benchmarks, we have more interesting observations. As we expected, nearly all benchmarks in *group1* favor *adaptive* over other schemes. Table 3 show the average results over the benchmarks in *group1* for different approaches. From Table 3, we observe that, for *group1* benchmarks, *adaptive* has achieved significantly better energy efficiency than fixed-interval DVFS schemes. The EDP result from *adaptive* is about 18% better than that from *fixed-PID* (16.0% vs. 13.5%), and nearly 3-fold better than that from *fixed-heuristic* (16.0% vs 4.3%). This shows the efficiency of our DVFS design and its self-tuning reactive advantages.

Next, we look at energy/performance results for the *group2* benchmarks, which have relatively slow or negligible workload variations. Table 4 show the average results over the *group2* benchmarks for different approaches. We observe that the *adaptive* results are still significantly better than those from the *Synchro* and *fixed-heuristic* (we attribute this to the MCD advantage and/or the effective design in Section 3 which is guided by formal stability analysis). However, compared to the *fixed-PID* result, the *adap-*

[2]We were not able to implement synchronous dynamic voltage scaling, so the *Synchro* results are for static scaling which may under-represent the benefit of synchronous voltage scaling slightly.

[3]We noticed that the *fixed-heuristic* results from our experiments are close to the published results in [23], but lower than that in [19]. See discussion in [23] for possible reasons.

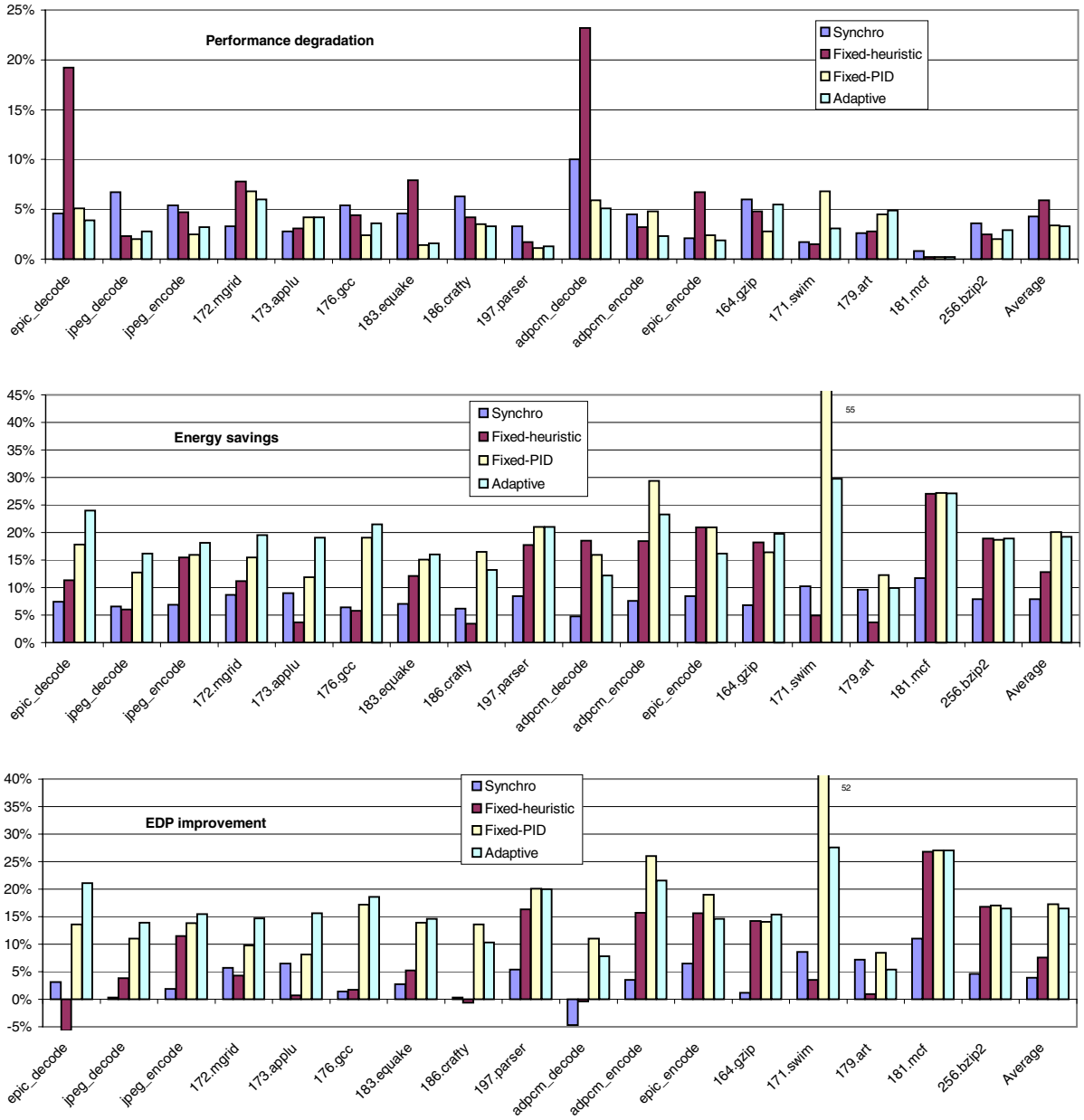

Figure 9: Performance degradation, energy savings, and energy-delay product (EDP) improvement for each benchmark; different schemes are *Synchronous* voltage scaling, two fixed-interval DVFS schemes (*fixed-heuristic* and *fixed-PID*), and the DVFS scheme with *adaptive* reaction time.

tive result lags by about 21% (17.1% vs. 21.7%). To understand the reason, we see that, for the *group2* benchmarks, the *adaptive* scheme gets little or no useful advantages over *fixed-PID* from responsiveness. On the other hand, *fixed-PID* still has the advantage over *adaptive* in terms of more accurate/intelligent DVFS decisions. Recall that, for *adaptive*, to keep the design simple and reduce the overhead of the decision process, we used simple time delays and single frequency increment or decrement for choosing possible DVFS actions. In general, DVFS actions picked by this mechanism are not as accurate/intelligent as those chosen by the *fixed-PID*, which may utilize system information from the current and past time-intervals and the PID controller to figure out more accurate voltage and frequency settings (of course, this requires more hardware). Therefore, for the *group2* benchmarks, the advantages associated with *fixed-PID* prevail and lead to better results for these than *adaptive*.

5.4 Comparison to *Fixed-PID* with Shorter Intervals

In previous sections, all *fixed-PID* results were obtained using the default interval-length in [23] ($10k$ instructions, which is also the interval length for *fixed-heuristic* in [19]). The question is what the results would be like for *fixed-PID* if a very short interval were chosen? Also, how do the *adaptive* results compare to those from *fixed-PID* with shorter intervals?

To answer the above questions, we first look at the pros and cons of reducing the interval-length for *fixed-PID*. On one hand, smaller interval length tends to make DVFS control in *fixed-PID* more fine-

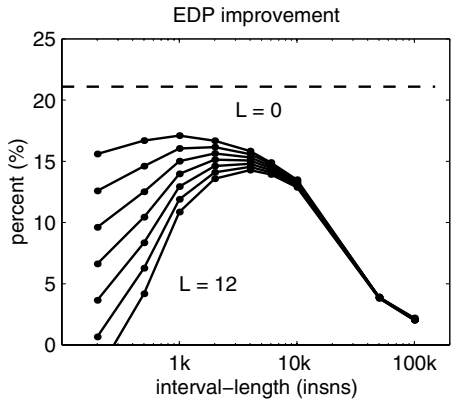

Figure 10: The energy delay product improvement (EDPI) for the benchmark Epic-decode obtained from *fixed-PID* (solid lines) as a function of interval lengths, and the increasing hardware complexity level L. We also show the EDPI from *adaptive* (the dashed line) as a comparison

grained, which may lead to more energy savings. On the other hand, the energy overhead associated with the DVFS decision process and trigger logic in *fixed-PID* also increases significantly with shorter intervals. As mentioned in Section 3, the decision logic in the *fixed-PID* can be separated into two parts. The first part is the logic being executed on a per-cycle basis (such as the counters and other book-keeping logic). The second part is the logic being executed on a per-interval basis, which is the part to implement the PID controller and is doing more complicated computations such as multiplication and division. Therefore, if the interval is relatively long, the amortized decision logic overhead will be small and negligible. However, if the interval becomes shorter and shorter, the amortized overhead will become larger and more significant.

Furthermore, even if we do not count the above decision overhead, the control efficiency for *fixed-PID* may still diminish with extremely short intervals. This is because the noise rejection ability of *fixed-PID*, which works like a low-pass filter using the average queue occupancy over the entire interval, begins to deteriorate with very short intervals.

Therefore, because of the above pros and cons, reducing the interval length may not lead to increased energy efficiency for *fixed-PID*. In general, there exist a range of medium interval-length values for which the *fixed-PID* has the best overall control performance. (The above discussion is also true for *fixed-heuristic*. The author of [18] showed that their DVFS algorithm works best for a range of interval lengths around 10k instructions).

Figure 10 shows the EDP improvement from the *fixed-PID* controller as a function of different interval lengths, and different decision logic overhead, for the benchmark Epic-decode. L in the figure is the scaling factor accounting for different level of hardware complexity of the *fixed-PID* decision logic[4]. ($L = 0$ is for the ideal and unrealistic case where the PID controller has zero energy cost). For comparison, we also show the EDP improvement from the *adaptive* for Epic-decode (the dashed line).

From Figure 10, we see the best control performance did not occur at very short intervals (like 1000 instructions or shorter). Rather, it occurred with an interval length somewhere in the medium range depending on the actual hardware complexity level.

We have also measured the efficiency of the *fixed-PID* with shorter intervals for other benchmarks, and compared them to results from *adaptive*. Overall, the observations and findings above hold for them as well.

[4]Specifically, L is used to scale the unit energy cost for *fixed-PID* decision logic in each time interval. The unit energy cost is estimated as $3 \cdot \alpha \cdot L \cdot C_0 \cdot V^2 \cdot k$, where *3* accounts for 3 clock domains; $\alpha = 0.5$ is the active factor, $L \cdot C_0$ is the total capacitance for the PID controller with C_0 chosen as 1% of the total capacitance of the integer ALU; V is the voltage in the front end; k is the total number of cycles the PID controller takes to finish the computation.

6. RELATED WORK

In this section, we highlight important related work which we have not discussed in previous sections.

As mentioned earlier, most existing hardware-based online intra-task DVFS work uses a fixed window or time-interval. There is little prior work in the direction of adaptive DVFS design. One pioneering work in that direction is the *Mode-switching* algorithm proposed by Iyer and Marculescu [9]. Their algorithm detects the *High* or *Low* queue occupancies (note they only look at the static queue occupancies, and do not use information on queue occupancy changes). If a *High* or *Low* queue condition has been detected for some consecutive cycles, a possible switch from *Fast-mode* to *Slow-mode* (or vice versa) will be made. We see, though the reaction time in their work is essentially adaptive, the simple two-mode switching algorithm has not been able to fully utilize the power/benefits brought by this paper's approach. For the purpose of comparison, we have re-implemented their *Mode-switching* DVFS algorithm and incorporated it into our simulation infrastructure. Our experimental results show that, in general, an energy-delay product improvement of $6 - 7\%$ has been achieved. In addition, without more detailed analysis, it is not analytically clear how to further improve [9].

Another two DVFS algorithms for MCD architectures which we have not discussed are the *Shaker* algorithm in [20] and the profile-based algorithm in [14]. They are both offline DVFS algorithms. As mentioned earlier, the focus of this paper is on hardware-based online DVFS schemes.

Recently, there have been increasing research efforts in applying control theory or other system theories in CPU design and control [22, 23]. One example is the applying of control theory in thermal control [22]. Another example is our own previous work on applying control theory to DVFS control for MCD processors [23]. The theoretical analysis in [23] is close to this work since the same architecture and the same system dynamics are being considered. However, the focus of the theoretical analysis in [23] is on the design, that is, how to model the controlled system and use control theory as guidances in designing a standard PID-based controller. On the other hand, the theoretical part of this work is focused on modeling/stability-analysis of an existing DVFS design. So in this work we derive a mathematical model for an existing design to be used in the analysis. Then, the obtained analytic insight is used to tune and improve the original design.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a new dynamic voltage and frequency scaling (DVFS) scheme for multiple-clock-domain (MCD) processors. The fundamental difference between this scheme and prior online DVFS schemes for MCD is in terms of when to react or make a DVFS decision. In the current scheme, the reaction time for DVFS is not predetermined, and is instead self-tuned and adaptive to application and workload changes.

As part of the scheme, we have designed a DVFS controller which has a simple decision process and reacts to recent instant queue occupancies. In addition, we have derived an analytic model

for the designed controller and use it in a formal stability analysis. The obtained analytical insight is then used to guide and improve the DVFS control.

Compared to the best-known prior fixed-interval DVFS schemes for MCD [19, 23], the decision process for the new scheme is much simpler, and this leads to smaller and cheaper hardware. Yet, our scheme has achieved a significant amount of energy savings over all studied benchmarks (about 19% energy savings with 3% performance degradation on average, which is close to the best results from prior online DVFS schemes). In addition, our scheme is potentially more responsive. For a group of applications with fast workload variations, our scheme outperforms existing fixed-interval DVFS schemes significantly: on average, 18% better than *fixed-PID* [23]; about 3 fold better than *fixed-heuristic* [19].

There are several avenues for future explorations. One possible effort would be to improve the DVFS control in this work by considering interactions among different queues. Another would be to combine the proposed DVFS control with thermal control to have a unified energy-thermal control scheme.

Overall, we feel the proposed adaptive online DVFS scheme is a promising alternative to the existing fixed-interval DVFS schemes. Designers may choose the new scheme for processors with limited hardware budget, or if the type of application behavior is known in advance to have high workload variability. In addition, the modeling and analysis techniques in this work serve as examples of using stability analysis in other aspects of high-performance CPU design and control.

Acknowledgments

We would like to thank David Albonesi and his research group for their MCD simulator, which we used as a base to develop our simulation infrastructure. We also thank Diana Marculescu, Anoop Iyer, Greg Semeraro, and YongKang Zhu for their helpful discussions during the development of this work; and the anonymous reviewers for their useful comments and suggestions on the paper. This work has been supported by NSF grants CCR-0086031 (ITR) and CNS-0410937. In addition, Martonosi's work is supported in part by Intel, IBM, and SRC.

8. REFERENCES

[1] Carl Anderson. Tuning and optimization of a 170m transistor microprocessor. In *Proceedings of the IEEE/ACM International Workshop on Timing Issue in the Specification and Synthesis of Digital System (TAU2000)*, Dec 2000.

[2] K.J. Astrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 1995.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimization. In *Proc. of the ISCA-27*, June 2000.

[4] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of ISLPED*, August 2000.

[5] D. Burger and T. M. Austin. The SimpleScalar tool set version 2.0. Technical Report 97-1342, Department of Computer Science, University of Wisconsin-Madison, June 1997.

[6] T. Chelcea and S. M. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proc. of DAC-2001*, pages 21–26, 2001.

[7] L.T. Clark. Circuit design of XScale microprocessors. In *Proceedings of the 2001 Symposium on VLSI Circuits*, June 2001.

[8] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *In Workshop on Complexity Effective Design, Vancouver, Canada, June 2000.*, June 2000.

[9] A. Iyer and D. Marculescu. Power efficiency of multiple clock multiple voltage cores. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2002.

[10] A. Iyer and D. Marculescu. Power-performance evaluation of globally asynchronous, locally synchronous processors. In *Proc. of the 26th ISCA*, May 2002.

[11] K.Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proceedings of DATE*, Feb 2004.

[12] B.C. Kuo. *Automatic Control Systems. , 7th edition*. Prentice Hall, 1995.

[13] D.V. Lindley. The theory of queues with a single server. In *Proceedings of the Cambridge Philosophical Society*, pages 277–289, 1952.

[14] G. Magklis, M.L. Scott, G. Semeraro, D.H. Albonesi, and S.Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *Proc. of the 30th ISCA*, June 2003.

[15] D. Marculescu, D.H. Albonesi, A. Buyuktosunoglu, and P. Bose. Partially asynchronous microprocessors (PAMs). In *ISCA 2003 Tutorial*, June 2003.

[16] D.B. Percival and A.T. Walden. *Spectral Analysis for Physical Applications*. Cambridge University Press, 1993.

[17] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *Proc. of the PACT-2003*, September 2003.

[18] G. Semeraro. Multiple clock domain microarchitecture design and analysis. In *Ph.D Thesis, University of Rochester*, August 2003.

[19] G. Semeraro, D.H. Albonesi, S.G. Dropsho, G. Magklis, S. Dwarkadas, and M.L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *Proc. of the 35th Micro*, pages 356–367, November 2002.

[20] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proc. of the 8th HPCA*, pages 29–40, February 2002.

[21] A.E. Sjogren and C.J. Myers. Interfacing synchronous and asynchronous modules within a high-speeed pipeline. In *Proceedings of the 17th International Conference on Advanced Research in VLSI*, pages 47–61, Sept 1997.

[22] K. Skadron, T. Abdelzaher, and M. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proc. of the 8th HPCA*, February 2002.

[23] Q. Wu, P. Juang, M. Martonosi, and D.W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *Proceedings of the 11th ASPLOS*, October 2004.

[24] Fen Xie, Margaret Martonosi, and Sharad Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proc. of 2003 PLDI*, June 2003.

[25] K.Y. Yun and A. E. Dooply. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, December 1999.