Execution Time Prediction for Energy-Efficient Hardware Accelerators

Tao Chen, Alexander Rucker, and G. Edward Suh
Cornell University
Ithaca, NY 14850, USA
{tc466, acr98, gs272}@cornell.edu

ABSTRACT

Many mobile applications utilize hardware accelerators for computation-intensive tasks. Often these tasks involve real-time user interactions and must finish within a certain amount of time for smooth user experience. In this paper, we propose a DVFS framework for hardware accelerators involving real-time user interactions. The framework automatically generates a predictor for each accelerator that predicts its execution time, and sets a DVFS level to just meet the response time requirement. Our evaluation results show, compared to running each accelerator at a constant frequency, our DVFS framework achieves 36.7% energy savings on average across a set of accelerators, while only missing 0.4% of the deadlines. The energy savings are only 3.8% less than an optimal DVFS scheme. We show with the introduction of a boost level, the deadline misses can be completely eliminated while still achieving 36.4% energy savings.

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems

Keywords

DVFS, energy efficiency, hardware accelerator

1. INTRODUCTION

Modern mobile SoCs often contain a large number of hardware accelerators/IP blocks, such as audio and video codecs, camera/image signal processor, gestures/motion processor, etc [1]. Analysis of die photos from recent generations of Apple SoCs indicate that more than half of the die area is used by blocks other than the CPU and GPU [2, 3]. The majority of these blocks are hardware accelerators. Mobile applications use hardware accelerators to enable features that are computationally intensive and thus not energy-efficient or infeasible to run on traditional CPUs, such as high-definition video playback/recording, image signal processing, etc.

MICRO-48 December 05-09, 2015, Waikiki, HI, USA
Copyright 2015 ACM. ISBN 978-1-4503-4034-2/15/12 ...$15.00.
DOI: http://dx.doi.org/10.1145/2830772.2830798

Although more energy-efficient than CPUs, the power consumption of hardware accelerators is becoming a concern as consumer markets demand ever more features and performance, such as 4K video and multi-megapixel cameras. For example, a recently published video codec [4] has an area of 2.16mm^2 in 28nm process and consumes 126.73mW. In comparison, the ARM Cortex-A7 has an area of 0.45mm^2 and consumes less than 100mW of total power in typical conditions [5]. Accelerators are also heavily used in next-generation wearable devices such as smart eyeglasses for augmented reality applications [6, 7]. Due to the compute-intensive nature of augmented reality algorithms, these heavy-weight accelerators consume considerable power, often greater than 400mW, resulting in less than 6 hours operation time with a 2.1Wh battery [6]. In addition to ASIC accelerators, FPGA-based accelerators are being deployed to accelerate various applications. For example, FPGAs used to accelerate the Bing web search engine consumes up to 22.7W per chip [8]. It is desirable to reduce the energy consumption of hardware accelerators, which could extend the battery life of mobile devices, or reduce the cooling and power distribution costs of datacenters.

Applications using hardware accelerators often have response time requirements, usually because they are interactive or frame-based. Interactive applications are required to respond to user inputs by a certain deadline for responsiveness. Frame-based applications need to render each frame before the screen refresh deadline, otherwise the frame will be dropped, and the application will feel sluggish. In both cases, meeting response time requirements is essential for good user experience. On the other hand, finishing tasks earlier than the response time requirements does not improve user experience due to the limits of human perception.

Today's software/hardware abstractions do not offer applications an easy way to express their timing requirements. As a result, the hardware accelerators/IP blocks are usually agnostic to the timing requirement of the applications, and operate in best-effort mode. When there is a high variation in the workload, the hardware accelerators need to operate conservatively to make sure they meet deadlines even in the worst-case. This means they often run at unnecessarily high performance levels compared to what is needed to meet deadlines in the average case, which leads to wasted energy. In other words, there often exists slack in interactive or real-time applications, which can be exploited to improve energy-efficiency by lowering the performance level of the system, using techniques such as dynamic voltage and frequency scaling (DVFS).

Various techniques exist for exploiting such slack in the software part of applications, using either reactive approaches [9–13] or predictive approaches [14–17]. However, exploiting slack to inform fine-grained DVFS has not been studied much for hardware accelerators. A previous study considered performing DVFS for smartphone applications that use hardware accelerators [18], but it assumed accelerators had constant execution time independent of inputs. However, many accelerators show significant input-dependent variations in execution time.

In this paper, we present a predictive approach to control the DVFS levels of hardware accelerators at fine granularity, exploiting input-dependent variations. We observe that input-dependent control decisions are the major source of execution time variations. A good estimate of an accelerator's execution time can be obtained if its control decisions for a certain input are known. In order to do this, we propose an automatic flow to generate a minimal version of a hardware accelerator from its RTL description, which computes the control decision features given an input. A model based on convex optimization is developed and trained to map these features to the accelerator's execution time. From that, we estimate the lowest DVFS level that meets real-time requirements, set a DVFS level, and run the accelerator.

Our approach is general and applicable to a wide range of hardware accelerators. We implemented and tested this predictive DVFS framework on a number of accelerators including video decoding, image processing, encryption, physics computation, etc. Our results show the proposed DVFS scheme achieves 36.7% energy savings compared to running each accelerator at a constant frequency. Comparing with a PID-based DVFS controller, predictive DVFS reduces deadline misses from 10.5% to 0.4% while being 4.3% more energy-efficient. The contributions of this paper are:

1. A look-ahead approach to control DVFS that uses metrics from upcoming workloads instead of looking at workload history.

2. An automated flow to extract features that affect execution time from RTL description of hardware accelerators, and a model based on convex optimization to map features to execution time and consequently the best DVFS level to use.

3. An automated flow to generate low-cost hardware-based predictors using hardware slicing techniques.

The rest of this paper is organized as follows. Section 2 discusses execution time variations in hardware accelerators and existing DVFS controllers. Section 3 describes our predictive DVFS framework, including the features used, prediction model, method to generate the predictor, and DVFS model. We also include a case study on using the framework for an example accelerator. Section 4 discusses our evaluation methodology, experimental setup, and evaluation results. Section 5 discusses related work, and Section 6 concludes the paper.

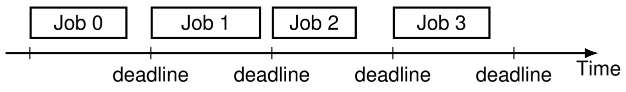

Figure 1: A sequence of jobs for a task.

2. FINE-GRAINED DVFS FOR HARDWARE ACCELERATORS

2.1 System Setup

The system we consider in this paper consists of processor cores, caches, main memory, and hardware accelerators. The cores and accelerators are loosely coupled. The accelerators access memory through DMA instead of going through the processor's cache. Each accelerator contains computation logic, and often internal scratchpad memories to store the working set. We assume each accelerator's DVFS level can be controlled individually.

2.2 Tasks and Jobs

Here we define some terminologies used in this paper. A *task* is a piece of work that has an associated deadline. For example, for a video player, decoding and rendering a frame is a task. In this case, the deadline associated with a task is determined by the frame rate of the video. A *job* is a dynamic instance of a task. For example, decoding a video at 60fps executes 60 jobs every second. Figure 1 shows a sequence of jobs for a task.

2.3 Execution Time Variation

The execution time for each job can vary depending on the job's input. For example, Figure 2 shows the execution time for a hardware H.264 decoder when decoding three video clips of the same resolution. We can see that even though all frames have the same resolution, there is a large variation in job execution time for frames in different videos, or even between frames in the same video. The reason for such large execution time variations is that for each frame, depending on the content in it, the H.264 algorithm chooses different modes to encode each macroblock in a frame, which leads to different computation complexity for decoding, and thus different execution time. Note that if we take into account videos of different resolutions, the execution time variation will be even larger. If we can lower the frequency for frames with shorter execution time, significant energy savings can be achieved.

However, setting an appropriate DVFS level for each job is not easy. The large and irregular variations in workload make it difficult to predict the next job's ex-

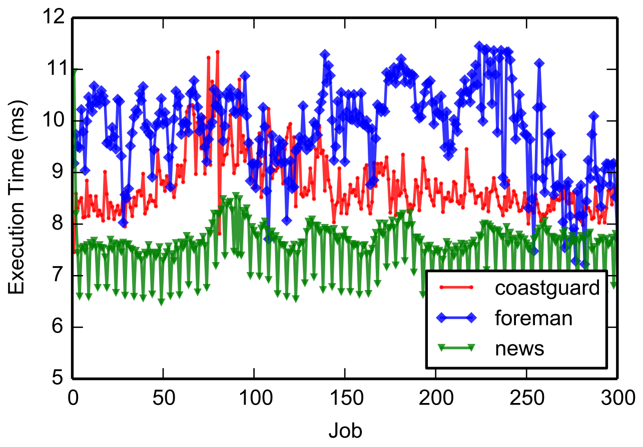

Figure 2: Execution time of H.264 decoder for three video clips at 60fps. Each point is one job (frame).

ecution time. Without accurate prediction, the DVFS controller has to be conservative and use higher DVFS levels to avoid deadline misses, missing opportunities for energy reduction. Otherwise, the DVFS controller risks missing deadlines when there is a sudden increase in job execution time.

2.4 Current Approaches to DVFS

DVFS is widely used for reducing the energy of computation. The key idea of DVFS is to reduce voltage and frequency to provide "just enough" performance to the application. An important part of a DVFS controller is the prediction of future workload, which allows the voltage and frequency to be lowered to the minimum required level while satisfying QoS requirements.

For applications without response time requirements, simple interval-based scheduling algorithms [19] can be used. These algorithms usually divide time into fixed-length intervals and measure the utilization of the previous interval and set DVFS level for the next interval. Since response time is not a requirement, some level of performance degradation caused by workload misprediction can be tolerated. These algorithms are widely used in operating systems. For example, the Linux power governors [9] are interval-based.

For applications with response time requirements, misprediction has to be minimized since it degrades quality-of-service. There are many approaches in literature and industry practice to perform DVFS under response time requirements. The following summarizes approaches that can be applied to hardware accelerators.

Table-based: Some hardware accelerators, including the Multi-Format Codec (MFC) in Samsung Exynos Series SoCs, use a lookup table to determine the DVFS level [20]. The table is addressed by a coarse-grained parameter, such as the resolution of a video. Before decoding a video, the driver will look into the table and set a DVFS level for the entire video sequence. People have also studied using the type of frames as inputs to the table [21]. However, these approaches do not take into account fine-grained job-to-job execution time variations. Essentially, these approaches set DVFS levels to the worst case for that coarse-grained parameter used to index the table. As can be seen in Figure 2, though all jobs have the same coarse-grained parameter (resolution in this case), most jobs have much shorter execution time than the worst-case. Thus the coarse-grained approach misses opportunities for energy reduction.

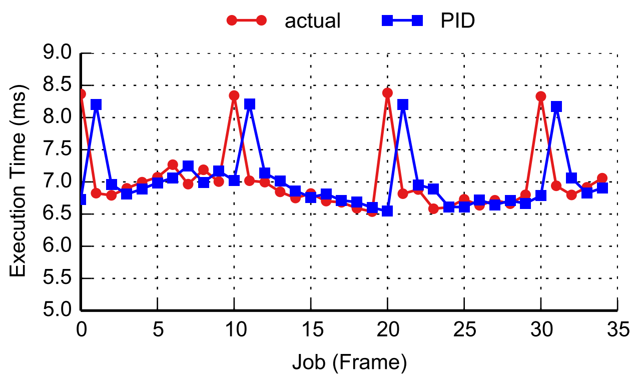

Figure 3: Actual execution time and execution time predicted by PID controller for H.264.

Reactive Control: A number of previous studies proposed using reactive control approaches to adjust DVFS levels. Some studies investigated using job execution time history to predict future job execution time, and set DVFS levels accordingly [10, 18]. Others proposed using control theory-based approaches, such as PID control [11–13]. Most of these studies target software-based systems, but some of them also consider hardware accelerators [18, 21]. These approaches are simple to implement, and work well for applications whose execution time varies slowly with time. However, many applications and accelerators do not fall into this category. For applications with rapid changes in job-to-job execution time, reactive decisions to adjust DVFS levels tend to lag behind actual changes, leading to deadline misses. Figure 3 shows an example how a PID-based controller mis-predicts job execution time for H.264 video decoding. When actual execution time shows spikes occasionally, the PID controller's prediction lags behind by one frame, causing one under-prediction and one over-prediction, which leads to one job missing deadline and one job running at unnecessarily high frequency around the spike. Apart from lagging behind in decision making, reactive control approaches can not be applied in some cases. For example, when browsing a website, the images sent to the JPEG decoding accelerator usually do not show correlation with previous or next images, rendering any reactive control approaches ineffective.

Predictive Control: There have been a few studies that looked at using predictive approaches to predict execution time and set DVFS levels accordingly. The target applications include interactive games [14], video players [22], web browsers [15,16] and servers [17]. Predictive control has been shown to outperform reactive control for these applications. However, all of these studies target software-based systems. Moreover, these approaches use application-specific features for predic-

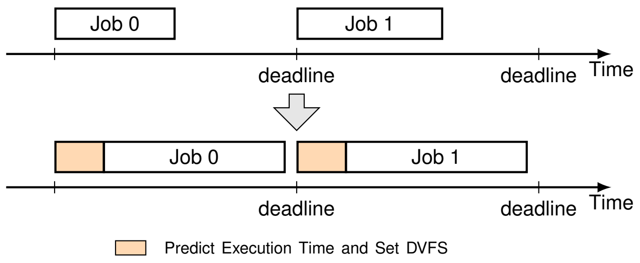

Figure 4: Operation of predictive DVFS.

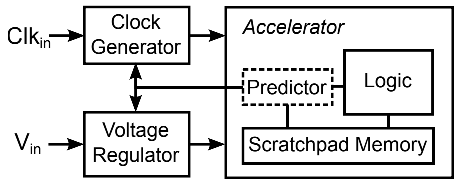

Figure 5: Accelerator with execution time prediction-based DVFS.

tion, which require domain-specific knowledge to obtain. Predictive control of DVFS for hardware accelerators is largely unexplored. As more and more hardware accelerators are deployed in applications today, it is necessary to take them into account in controlling DVFS. In this paper, we propose a predictive DVFS framework for hardware accelerators. In addition, our prediction framework uses features automatically extracted from hardware, which eliminates the need for domain-specific knowledge in obtaining features.

3. PREDICTIVE DVFS FRAMEWORK FOR HARDWARE ACCELERATORS

In this section, we propose a framework for controlling accelerator DVFS based on execution time prediction. At high level, our goal is to predict the lowest DVFS level each job can run at without violating response time requirements. This can be done in two steps: first, we predict the execution time for each job at a nominal frequency (i.e. without doing DVFS). After that, we predict what the execution time would be at each DVFS level, and choose the lowest level that meets response time requirements.

Figure 4 illustrates how accelerators operate with predictive DVFS. For each job, a predictor is run first to obtain an estimate of the execution time of the job. Then the best DVFS level is set according to predicted execution time. After frequency and voltage change stabilizes, the accelerator's main logic starts execution. Figure 5 shows the block diagram of an accelerator with execution time prediction-based DVFS. For each job, the predictor informs the clock generator and voltage regulator the frequency and voltage to be used. Access to the scratchpad memory is time-multiplexed between the predictor and the main computation logic. Although in our implementation the predictor directly controls DVFS circuitry, the control can also be done in software through the operating system.

We have the following design goals:

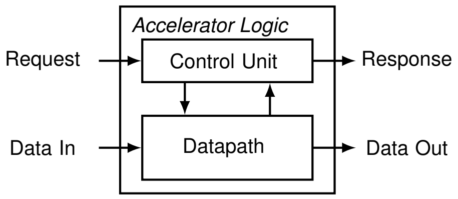

Figure 7: Control-Datapath structure of an accelerator.

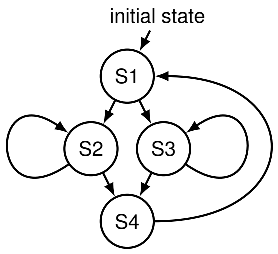

Figure 8: Example Finite State Machine in control unit.

- **Look-ahead**: Instead of reacting to changes in job execution time, the DVFS controller looks ahead into upcoming jobs and predicts what the execution time would be before actually running the jobs.
- **General**: The DVFS framework should be general and applicable to a wide range of accelerators. To this end, the framework does not use application-specific knowledge.
- **Automated**: The DVFS controller should be generated by an automated flow with minimal manual effort.
- **Low overhead**: The DVFS controller should have low overhead in terms of area and energy, or increased design complexity.

Figure 6 shows the high-level flow for our DVFS framework based on execution time prediction. It consists of two parts. The offline part works during the design process of the accelerator to generate the predictor. The online part shows the operation of the predictor during accelerator execution.

Although we only investigate DVFS in this paper, this approach can also be applied to other methods for performance-energy trade-off, such as dynamically reconfiguring accelerators to different performance-energy points, etc.

3.1 Source of Execution Time Variation

In Section 2.3, we showed that accelerators can have significant input-dependent execution time variations. Here we describe where these variations come from. Figure 7 shows a typical high-level structure of an accelerator. It mainly consists of two parts: control unit and datapath. The control unit is responsible for handling requests and responses, as well as orchestrating computation in the datapath by generating various control signals. The datapath performs computation on the input data to generate the output, and also generates various signals for the control unit to make decisions.

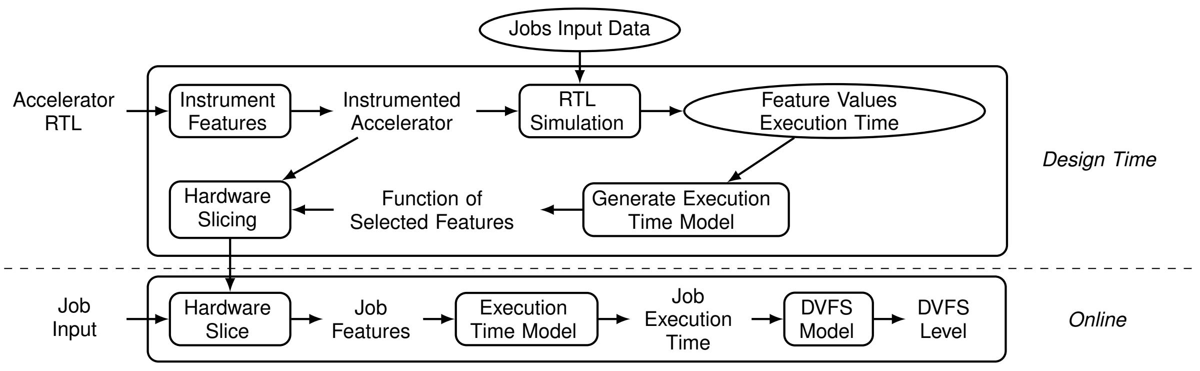

Figure 6: Execution time prediction flow.

The control unit is usually composed of one or more Finite State Machines (FSMs). Figure 8 shows an example FSM from the control unit of an accelerator. In state S1, the accelerator reads a piece of input data. Then, depending on the value, the FSM transitions to either state S2 or S3 to perform computation. When computation is done, the FSM transitions to state S4 to generate an output, then transitions back to S1 to process the next input. The computation in state S2 and S3 can take different number of cycles (for example, 50 and 100 respectively). This is the major source of execution time variation.

A job usually uses multiple inputs. For example, an image consists of multiple macroblocks. If we know the decisions made by the control FSMs when processing each input, and the processing time of each computation state, we can predict execution time, and consequently the best DVFS level to run the job at.

3.2 Features from Hardware Accelerators

In this section, we describe the features we use to represent the decisions of the control unit. Here a feature refers to a measurable property that can be extracted during accelerator execution. We also show how these features and the control decisions they represent correlate with execution time.

We observe that control unit decisions are embedded in the state transitions of the control unit FSM. For example, in Figure 8, a state transition from S1 to S2 means the control unit decides to perform some computation associated with state S2. If we count the number of state transitions from S1 to S2, we can know the number of times computation associated with S2 has taken place. Thus, we use *state transition count* (STC) as a type of feature in our prediction model.

However, knowing state transition counts is not enough. We also need to know how each state transition impacts execution time of the accelerator. This can be divided into two cases: If the latency of a computation state is fixed, we can use statistical regression to figure out how much time the computation takes given enough training data. If the latency is variable depending on input, statistical regression can only estimate the average latency, which is not enough to make good predictions. We observe that in this case, there is usually a counter to keep track of whether the computation is finished. For example, when the computation starts, the control unit sets the counter to the latency of the computation. In each cycle the counter is decremented until it reaches zero, signaling the end of computation. The *range* of the counter correlates with the computation's impact on execution time. In a decrementing counter, range can be obtained from the counter's initial value. In an incrementing counter, range can be obtained from the counter's final value before a reset. Thus, we use several counter-related features: 1) *initialization count* (IC), which is the number of times each counter is initialized. 2) *average initial value* (AIV), which is the average value a counter is initialized to. 3) *average pre-reset value* (APV), which is the average of a counter's final value before a reset. The last two features capture the range of each counter. Table 1 summarizes the features we use in our prediction model.

Feature	Source	Granularity
STC	FSM	Each source-destination states pair
IC	Counter	Each counter
AIV	Counter	Each counter
APV	Counter	Each counter

Table 1: Summary of features in prediction model.

3.3 Identifying and Obtaining Features

Manually annotating and modifying FSMs and counters in hardware accelerator designs would be too tedious and not feasible for large designs. Moreover, many accelerators are third-party IPs and system designers are not familiar with their internals. Thus, we propose an automated approach to identify and extract such features in hardware accelerators based on a static analysis of RTL code of accelerators.

The first step of the analysis is to identify FSMs and counters in the RTL, as these are the sources of features. To achieve this, a behavioral RTL of hardware accelerators is first transformed to a structural RTL using a synthesis tool. We use Yosys [23], which is an open-source synthesis suite. After that, we use an algorithm to find FSMs in the design based on techniques from a previous study [24] on extracting FSMs from a gate-level netlist. The algorithm works by analyzing the RTL and look for specific structures related to FSMs. Similar to identifying FSMs, counters are also identified by RTL analysis.

The next step is to instrument the accelerator so that it records feature values during its operation, as illustrated in the offline stage of Figure 6. This is done through RTL analysis and instrumentation. For state transition counts, we extract each FSM's transition table and compute the criteria for each state transition to take place. For each source-destination pair, we instrument the RTL to generate a signal whenever the transition criteria is met, and record the number of times the signal is asserted using a register. With this, we can simply read out the register's value to get a state transition count. Similarly, for initialization counts, we compute the criteria for the counter to be initialized and instrument the RTL to generate a signal when the criteria is met. To keep track of an average initial value and an average pre-reset value, we use registers which are controlled by the initialization criteria. Note that we do not actually have to calculate the average, it is sufficient to record the sum of these values and the prediction model will take care of scaling the values to obtain average. All these steps are done automatically without manual effort using our RTL analysis and instrumentation framework implemented inside the Yosys open-source Verilog analysis and synthesis suite.

After instrumenting the accelerator, we run RTL simulations with a training set of job input data to obtain the feature values as well as execution time for each job.

3.4 Prediction Model

After obtaining the features and execution time for each job, we develop a model that takes feature values and maps them to execution time. The model is then trained using the feature values and execution time data from training runs. We have the following design goals for our prediction model: (1) Accurate prediction. (2) Low overhead in terms of time, area, and energy. A simple model which uses a small number of features is preferred. (3) Conservative prediction, which means when there is a trade-off to be made between a deadline miss and less energy savings, the model should avoid deadline miss even though it may use more energy.

With these design goals in mind, we use a linear model to map feature values to execution time. Linear models are very simple to evaluate at runtime by calculating the dot product of feature values and model coefficients, which is just a series of multiply accumulate operations. The linear model can be written as

$$\bar{y} = \mathbf{x}\boldsymbol{\beta}$$

where $\bar{y}$ is the predicted execution time, $\mathbf{x}$ is a vector of feature values, and $\boldsymbol{\beta}$ is a vector of model coefficients. Table 2 summarizes the variables in our prediction model.

To train a linear model, the most common way is to use a least squared error as a metric. That is, we try to minimize the following term

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|^2$$

However, this commonly used approach has major drawbacks in the context of DVFS control: first, it uses all feature values to calculate the target function, despite the fact that only a few features are often sufficient to predict the execution time. Second, this approach tries to minimize both positive and negative errors. However, in the context of DVFS, it is more important to minimize negative errors to reduce deadline misses.

Variable	Type	Description
$\bar{y}$	Scalar	Predicted execution time
$\mathbf{x}$	Vector	Feature values
$\boldsymbol{\beta}$	Vector	Model coefficients
$\mathbf{y}$	Vector	Profiled execution times
$\mathbf{X}$	Matrix	Profiled feature values
$\mathbf{X}\boldsymbol{\beta} - \mathbf{y}$	Vector	Prediction errors
α	Scalar	Under-predict penalty weight
γ	Scalar	Number of terms penalty weight
$\|\cdot\|$	Scalar	L2-norm (Euclidean distance)
$\|\cdot\|_1$	Scalar	L1-norm (sum of absolute values)

Table 2: Variables in prediction model.

To address the first issue, we use Lasso [25] to minimize the number of non-zero coefficients in our model by adding a penalty term to our model:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|^2 + \gamma\|\boldsymbol{\beta}\|_1$$

where γ is parameter empirically determined to reduce the number of non-zero coefficients without impacting modeling accuracy too much. To address the second issue, we separate positive and negative errors:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|pos(\mathbf{X}\boldsymbol{\beta} - \mathbf{y})\|^2 + \alpha\|neg(\mathbf{X}\boldsymbol{\beta} - \mathbf{y})\|^2 + \gamma\|\boldsymbol{\beta}\|_1$$

where $pos(x) = max(x, 0)$ and $neg(x) = max(-x, 0)$. We set $\alpha > 1$ to place more weight on negative errors.

It can be shown that the objective function we try to minimize above is convex. Thus, we can use a convex optimization solver to fit the model.

3.5 Hardware Slicing

Now we have a model to predict execution time from features. However, to obtain feature values for a job at run-time, we need to run the hardware accelerator with the job's input. This is not feasible since our goal is to predict execution time before running the hardware accelerator. To address this issue, we propose to generate a minimal version of the hardware accelerator, which we call a *hardware slice*. During runtime, the slice can be executed with the job's input to quickly calculate the feature values.

To generate such a slice, we use program slicing techniques on hardware description languages [26] to only keep the part of the original accelerator that computes the features of interest, while removing other parts of the hardware. This allows us to obtain a slice that is much smaller than the original hardware accelerator in terms of area.

However, executing such a slice would take the same number of cycles as the original hardware accelerator. This is not fast enough since we need to make predictions before running the hardware accelerator. The reason why a slice can not run faster is that the control unit is not aware that some parts of the hardware were removed, and still waits in certain states as if the original computation is still taking place. For example, in Figure 8, the FSM still transitions to S2, waits for a number of cycles, and then transitions to S4. This inefficiency can be removed by modifying the FSM transition table to remove the waiting behavior. This optimization does not affect the accuracy of the prediction because we still have the information about how long the FSM would stay in waiting states from counter initial values and pre-reset values. The resulting hardware slice efficiently calculates the control flow features of the original hardware accelerator.

3.6 DVFS Model

After obtaining an execution time prediction for a job under the nominal frequency, a DVFS model is used to predict what the execution time would be under different frequencies. We use a common model in literature [27] that decomposes execution time into memory time and compute time:

$$T = T_{memory} + C/f$$

where T is execution time, T_{memory} is the part of execution time when the accelerator is stalled waiting for memory, which does not scale with accelerator frequency. C is the number of cycles when the accelerator is not stalled, and f is the frequency of the accelerator. From the execution time prediction, we know

$$T_0 = T_{memory} + C/f_0$$

where T_0 is the predicted execution time, and f_0 is the nominal frequency. To predict the execution time under a different frequency, we need to know T_{memory} and C. We found that for the many accelerators, T_{memory} is negligible, for two reasons: 1) Many accelerators are computation-intensive rather than memory-intensive. 2) For accelerators that are memory-intensive, they usually have a local scratchpad memory and use a DMA engine to carefully coordinate data transfer from main memory to minimize stalls due to memory. Thus, the equation above can be simplified as

$$T_0 = C/f_0$$

Assuming T_{budget} is the time budget for the job, we can run the accelerator at

$$f = C/T_{budget} = f_0 T_0/T_{budget}$$

to minimize energy while meeting deadline.

In real hardware, however, there are only a few discrete frequencies the accelerator can run at. As a result, we need to round up f to the nearest frequency level. Also, executing the hardware slice and switching voltage/frequency takes some time, which needs to be deducted from T_{budget}. We can also add a margin to the predicted execution time to be conservative. After taking all these into account, we set the final frequency level to be

$$f = \lceil f_0(T_0 + T_{margin})/(T_{budget} - T_{slice} - T_{DVFS}) \rceil$$

and execute the accelerator, where $\lceil \cdot \rceil$ represents rounding to the nearest frequency level above.

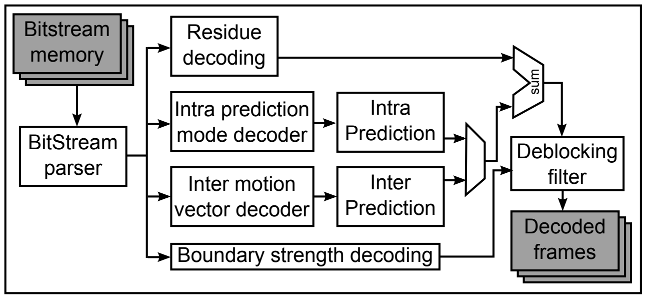

Figure 9: Architecture of H.264 decoder.

3.7 Case Study

In this section, we present a case study on execution time prediction using the H.264 decoder as an example. We discuss the features chosen by the framework and why these features can be used to predict execution time. We also discuss the details of the hardware slice, and show which parts of the original accelerator were kept and which parts were sliced out.

Figure 9 shows the high-level architecture of the H.264 decoder [28]. During decoding, the bitstream parser reads an H.264 bitstream from memory, and performs parsing and entropy decoding. Then, according to the prediction type, each macroblock is either sent to the intra prediction or inter prediction pipeline. The prediction output is then combined with the output of residue decoding. The result is then processed by the deblocking filter to generate the final picture. Each block in Figure 9 has its control unit and datapath.

In feature detection step, our framework detected 257 features related to FSMs and counters. Using Lasso, the number of features is reduced to only 7 while still maintaining good prediction accuracy with a worst-case error around 3%. Among these features, two of them are FSM transitions related to the number of transform coefficients in the residue decoding of a macroblock. The other 5 features come from the inter prediction (motion compensation) pipeline. They are counters that control the preloading of data used by inter prediction, as well as counters that control the computation of macroblocks. All these features are in the control unit of the corresponding blocks of the H.264 decoder, and thus do not involve the main computation datapath.

Since the most computation-intensive parts of the decoder are not involved in generating the features, the hardware slicing step of the framework was able to remove them, such as the datapath of intra and inter prediction, deblocking filter, etc. The slice only contains the bitstream parser and the control units of intra and inter prediction pipeline. As a result, the hardware slice was very small and energy-efficient compared to the full decoder. The area of the slice is 37,713μm^2, which is only 5.7% of the full decoder. In addition, the execution time optimization step of our framework was able to remove empty waiting states in the hardware slice, thus the slice only takes 5%-15% percent of the full decoder's execution time to generate features. As a result, the hardware slice only consumes 2.8% of the energy compared to the full decoder. Furthermore, predicting the execution time from features is very fast since the model is linear with only 7 coefficients.

For comparison, we also built a predictor based on application-specific features that we manually identified for H.264 using an approach proposed in a previous study [22]. These features were obtained using an H.264 bitstream analysis software. Surprisingly, the predictor using manually identified features had a worst-case prediction error around 10%, compared to 3% for our automatically generated predictor. Further inspection revealed that a subtle effect (long latency for blocks with quarter-pixel motion vectors) was not captured by the manually identified features. While carefully chosen application-specific features may improve prediction accuracy, obtaining them requires a deep understanding of the algorithm, which often can only be done by domain experts.

4. EVALUATION

In this section, we present the evaluation results for the proposed DVFS framework. We first discuss our evaluation methodology and experimental setup. Then, we show evaluation results for ASIC and FPGA-based accelerators, and discuss some extensions to our work.

4.1 Methodology

We use a vertically integrated evaluation methodology that uses a combination of circuit-level, gate-level and register-transfer-level modeling. Circuit-level modeling is used to characterize the voltage-frequency relationship. Gate-level modeling is used to build accurate area and energy models. And register-transfer-level modeling is used to accurately model the performance of hardware accelerators.

Circuit-level modeling: For ASIC accelerators, we characterize the relationship between voltage and frequency for our accelerators using SPICE simulations based on a method reported in literature [29]. For each accelerator, we used a chain of FO4 loaded inverters such that the total delay of the chain matches the cycle time of the accelerator at nominal voltage. Then we change the supply voltage and measure the voltage-delay curve and use that to model the accelerator's frequency under different voltage levels. For FPGA accelerators, the voltage-frequency relationship is obtained from published characterizations [30].

Gate-level modeling: For ASIC accelerators, we implemented each hardware accelerator using Synopsys Design Compiler, IC Compiler, PrimeTime PX with the TSMC 65nm standard-cell library characterized at 1 V. Detailed post-place-and-route gate-level simulations were used to obtain the power and energy of the accelerator's execution for a subset of the jobs at 1 V. Then we apply the voltage-frequency model and the frequency-execution time model to estimate the power and the energy consumption under different DVFS levels. For FPGA accelerators, the synthesis and place-and-route flow is based on Vivado 2014.2. Post-place-and-route simulations are used to obtain power and energy estimations, which are then scaled to different DVFS levels.

Register-transfer-level modeling: We use RTL simulations to determine the execution time of each job for our accelerators. We assume the accelerators are not bandwidth-limited and the DMA controller transfers data from the main memory to the accelerator's scratchpad before executing each job.

4.2 Experimental Setup

We use a set of seven benchmark accelerators in our evaluation, including an H.264 video decoder [28], a JPEG encoder [31], a JPEG decoder [32], a molecular dynamics accelerator [33], a stencil filtering accelerator used in image processing [33], an Advanced Encryption Standard (AES) accelerator [34], and a Secure Hash Algorithm (SHA) accelerator [35]. Note that even though some hardware accelerators are traditionally throughput-oriented, they can have response time requirements when used as a part of a frame-based or interactive application. For example, when a user is playing a DRM-protected video, a crypto accelerator has to decrypt the data for each frame before a certain deadline. As another example, when a smartphone camera shoots in a burst mode, the JPEG engine has to encode each picture before a certain deadline. Table 3 lists the accelerators, describes what a task is in each accelerator, and the workloads we use to train and test the DVFS controller.

For ASIC accelerators, we use six equally-spaced voltage levels that span from the nominal voltage at 1 V (high performance/energy point) to 0.625 V (low performance/energy point). The frequency corresponding to a voltage is determined using the voltage-frequency relationship obtained from circuit-level modeling. For FPGA accelerators, we use seven equally-spaced voltage levels from 1 V to 0.7 V. We assume voltage regulators are off-chip. Switching time for off-chip voltage regulators are typically in the range of 10μs [36]. In our evaluation, switching time is conservatively set to 100μs to account for potential overheads in case changing DVFS levels involves device drivers. However, we do note that there are faster DVFS switching techniques in literature [29, 36], which could further reduce DVFS switching overhead to the range of tens of nanoseconds.

We set the deadline for each job at 16.7ms, which corresponds to the 60fps screen refresh rate in most of today's devices. We compare our prediction-based DVFS controller with the following schemes:

1. **baseline**: The baseline runs at constant voltage and frequency. Each accelerator runs at 1 V and the frequency it is synthesized at.
2. **pid**: The PID-based controller uses prediction errors from previous jobs together with a control-theory based algorithm to determine the execution time of the next job. For each accelerator, we tuned the PID controller's parameters to achieve the best prediction accuracy. A margin is added to PID controller's output to reduce the number of deadline misses. We tried different margins and chose 10% as the amount that balances deadline miss rate and energy savings.

Bmark.	Description	Task	Workload (Train)	Workload (Test)
h264	H.264 video decoder	Decode one frame	2 videos (600 frames, same size)	5 videos (1500 frames, same size)
cjpeg	JPEG encoder	Encode one image	100 images (various sizes)	100 images (various sizes)
djpeg	JPEG decoder	Decode one image	100 images (various sizes)	100 images (various sizes)
md	Molecules/physics simulation	Simulate one timestep	200 steps (particle pos. changes)	200 steps (particle pos. changes)
stencil	Image filtering	Filter one image	100 images (various sizes)	100 images (various sizes)
aes	Adv. Encryption Standard	Encrypt a piece of data	100 pieces of data (various sizes)	100 pieces of data (various sizes)
sha	Secure Hash Function	Hash a piece of data	100 pieces of data (various sizes)	100 pieces of data (various sizes)

Table 3: Summary of benchmarks and workloads.

Bench-mark	Area (μm^2)	Freq. (MHz)	Execution Time (ms) Max	Avg.	Min
h264	659,506	250	11.46	7.56	6.50
cjpeg	175,225	250	13.90	5.22	0.88
djpeg	394,635	250	14.79	3.78	1.82
md	31,791	455	15.52	7.11	0.80
stencil	10,140	602	15.97	5.92	1.41
aes	56,121	500	16.19	4.62	1.94
sha	19,740	500	12.94	4.11	1.11

Table 4: Summary of ASIC implementation results.

3. **prediction**: This is our prediction-based DVFS controller. A 5% margin is added to its prediction. Its predictions are usually fairly accurate so only a small margin is needed.

4.3 Results for ASIC Accelerators

Implementation Results.

Table 4 shows the area, frequency, and execution time statistics for the benchmark accelerators. The area numbers are from place-and-route results. The frequency numbers are shown for nominal voltage at 1 V. The execution time statistics are obtained at nominal voltage and frequency. Large execution time variations are observed.

Execution Time Prediction Accuracy.

Figure 10 shows box-and-whisker plots of prediction error of our scheme for each benchmark. The box extends from the 25% to 75%, with a line at the median. The whiskers extend from the box to show the range of the data. Positive numbers correspond to over-prediction and negative numbers correspond to under-prediction. For most benchmarks, the prediction error is negligible, indicating the effectiveness of our approach. The JPEG decoder showed higher prediction error. This is because some of its execution time variations cannot be accurately modeled using the extracted features. Specifically, some of the FSMs in the decoder stay in a state for a variable number of cycles which cannot be obtained using a corresponding counter. However, our slice-based predictor still captured the majority of its execution time variations. In addition, the convex optimization-based prediction framework is conservative and leads to very few under-predictions.

Energy Savings and Deadline Misses.

Figure 11 shows the comparison of normalized energy and deadline misses between different DVFS schemes. The energy numbers are normalized to the baseline. The baseline always runs at constant frequency and thus

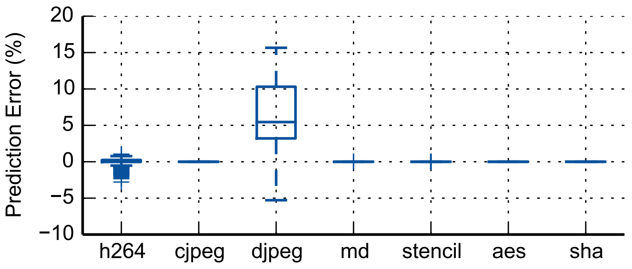

Figure 10: Errors of slice-based execution time prediction. The box extends from the 25% to 75%, with a line at the median. The whiskers show the range of the data. Outliers are shown as individual points.

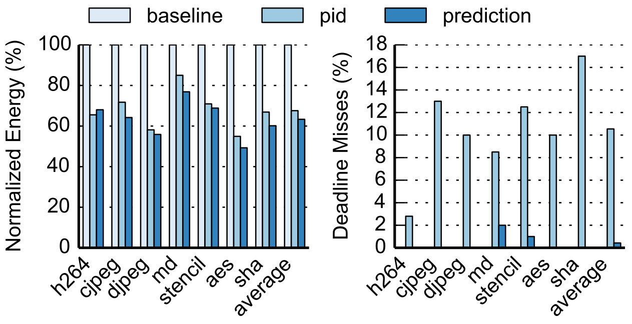

Figure 11: Normalized energy and deadline misses of different DVFS schemes.

has the highest energy but no deadline misses. On average, our schemes achieved 36.7% energy savings across all benchmarks. PID-based DVFS controller has 4.3% higher energy consumption than our scheme. In addition, the PID controller often chooses lower DVFS levels than needed which leads to many deadline misses. On average, the PID-based controller misses 10.5% of the deadlines while our prediction-based controller misses only 0.4% of the deadlines.

Overheads of Execution Time Prediction.

The hardware slice for our prediction-based DVFS has overheads in terms of area, energy, and time. The prediction slice takes up extra die area, and consumes energy during execution. Also, since the prediction slice is run before the actual job, the time needed to run the slice reduces the amount of time left to run the job, which in turn reduces the opportunity to run the job at a lower frequency. Figure 12 shows the overheads of the slice. Slice energy is normalized to the actual job's en-

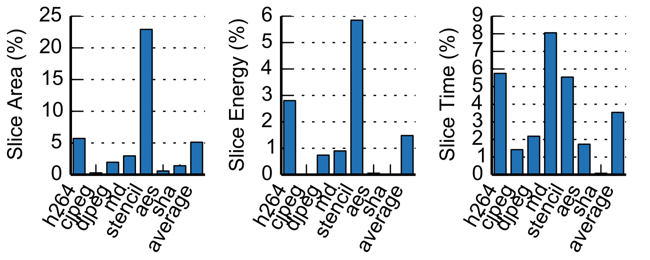

Figure 12: Area, energy and execution time overhead of prediction slice.

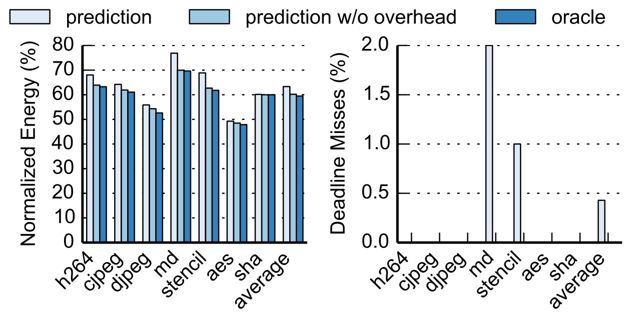

Figure 13: Normalized energy and deadline when overhead is removed.

ergy. Slice area is normalized to the accelerator's area. Slice time is normalized to the job's deadline. On average, the prediction slice adds 5.1% area overhead to the baseline accelerator. Running the prediction slice takes about 3.5% of the time budget, while adding 1.5% energy overhead to the job on average. The energy overhead is low because the slice is small compared to the full accelerator, and only runs for a short period. Besides the overheads introduced by the hardware slice, DVFS switching also has overheads since the voltage and frequency takes time to stabilize. Note that the results shown in Figure 11 includes these overheads.

To better understand how these overheads impact energy savings and deadline misses, we show the results for the prediction scheme when the overheads of hardware slice and DVFS switching are removed. In addition, we also show the results for an oracle scheme that always sets a best DVFS level for each job, and without DVFS switching overhead. Figure 13 shows that by removing these overheads, energy savings are improved by 3.1%, from 36.7% to 39.8%. Deadline misses are reduced from 0.4% to 0%. The oracle scheme has 40.5% energy savings, which is 0.7% higher than the prediction scheme without overheads. Both the oracle scheme and the prediction scheme without overhead have zero deadline misses. This shows that the prediction scheme with overhead removed is very close to optimal. It also indicates that the deadline misses we see in the prediction scheme without overhead is not due to misprediction. Instead, it is because insufficient time budget is left after the slice finishes execution, so that even running at highest frequency will miss the deadline. This only happens to jobs whose execution time is very close to, or

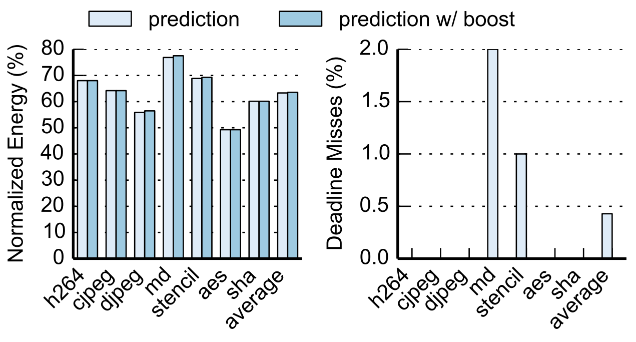

Figure 14: Normalized energy and deadline misses with voltage boosting.

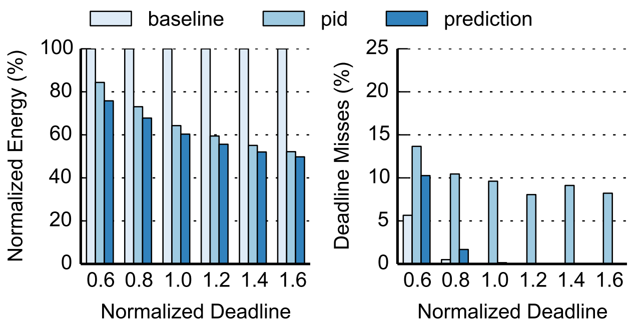

Figure 15: Normalized energy and deadline misses when varying deadlines (averaged across all benchmarks).

even longer than the length of the deadline, which are usually rare.

These rare misses can be eliminated by boosting voltage temporarily for these jobs. With execution time prediction, the DVFS controller knows when the time budget left is not enough, and can boost DVFS level accordingly. Figure 14 shows normalized energy and deadline misses when we introduce a boost level at 1.08 V. With voltage boosting, deadline misses are eliminated while only increasing normalized energy by 0.24%.

Sensitivity Study on Varying Deadlines.

Figure 15 shows the normalized energy and deadline misses when we vary the job deadline from 0.6x to 1.6x of the deadline used before. Due to space limit, we only show results averaged across all benchmarks. Since our DVFS model is aware of the deadline, it will use lower DVFS levels to save energy when deadlines are longer, and use higher DVFS levels trying to meet response time requirements when deadlines are shorter. When the deadline is shorter than 1.0x, the prediction-based DVFS controller starts showing misses. This is mostly due to the deadlines being too short to meet for some jobs even when running at highest frequency, which is why the baseline policy also shows misses. When the deadline is increased, the prediction-based DVFS controller achieves more energy savings while still meeting all deadlines. Note that the execution time predictor does not need to be retrained when the deadline changes. Only a new deadline needs to be set in the

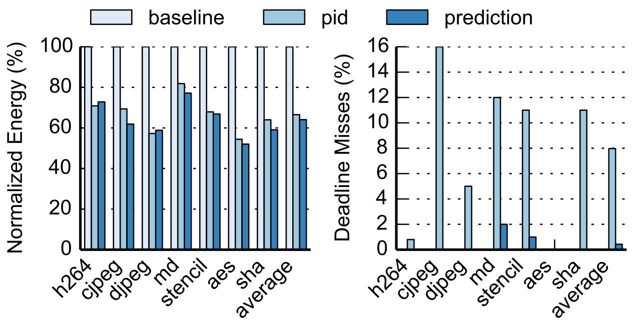

Figure 16: Normalized energy and deadline misses for FPGA-based accelerators.

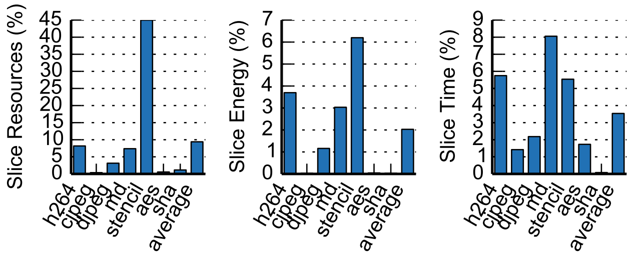

Figure 17: Area, energy and execution time overhead of prediction slice for FPGA accelerators.

DVFS model. The PID-based controller, on the other hand, still shows misses even with longer deadlines because it often uses the wrong DVFS level due to low prediction accuracy.

4.4 Results for FPGA-based Accelerators

In this section, we show results for FPGA-based accelerators. The target FPGA device we use is Xilinx Kintex-7. The execution time prediction accuracy for FPGA accelerators is similar to the accuracy for ASIC accelerators because the features used for prediction are from RTL level, and the prediction model is able to adapt to differences in operation frequency.

Figure 16 shows normalized energy and deadline misses of different DVFS schemes for FPGA-based accelerators. Overall, FPGA-based accelerators achieved 35.9% energy savings with the predictive DVFS framework, while missing 0.4% of the deadlines. The numbers are comparable to ASIC results.

Figure 17 shows the overheads of the slice for FPGA-based accelerators. On average, the prediction slice use 9.4% resources (average of LUT/DSP/BRAM) of the baseline accelerator. Running the prediction slice consumes 2% of the energy of the job on average, while using about 3.5% of the time budget. The resource overhead for `stencil` appears large because the accelerator only uses a small number of LUTs for control logic while using DSP blocks for main computations. As a result, the relative overhead is shown to be high even though the absolute resource usage of the prediction slice is quite low.

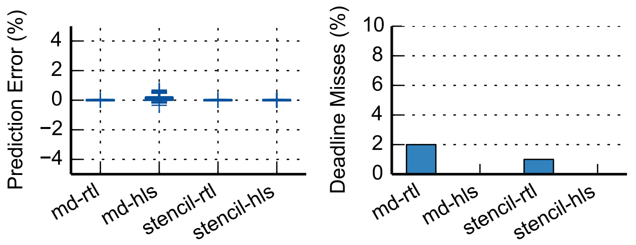

Figure 18: Comparison of prediction errors and deadline misses between slicing at RTL and HLS level.

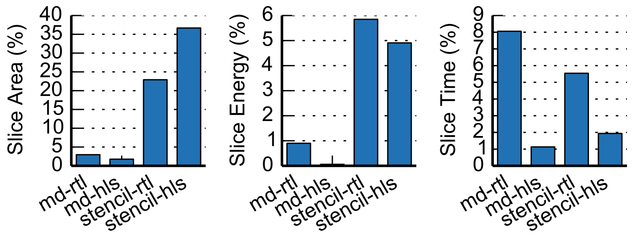

Figure 19: Comparison of area, energy and execution time overhead between slicing at RTL and HLS level.

4.5 Extensions

Accelerators Generated using High-Level Synthesis.

High-Level Synthesis (HLS) allows designers to write accelerators in a high-level programming language such as C, and synthesizes them into RTL descriptions. To use our framework with HLS-generated accelerators, one way is to analyze the RTL generated by HLS, extract features, and build a predictor by slicing the RTL. However, if analysis can be done during the HLS process, it can potentially enable optimizations that can not be easily performed at RTL level. For example, instead of slicing the RTL to obtain a hardware slice, we can use program slicing [37] on the C code to obtain a software slice that calculates the control flow features, and then synthesize the sliced C code into hardware. The HLS tool can perform optimizations during the synthesis, resulting in a slice that calculates feature values faster. This leaves more time budget to run the accelerator itself, reducing the possibility of deadline misses due to insufficient time budget after slice execution.

We compared these two approaches using the `md` and `stencil` accelerators [33], which have C versions available. Figure 18 shows that the prediction accuracy of both approaches are very high, but when the hardware slice is generated using HLS from sliced C code, the deadline misses are gone. This implies the slice generated using HLS runs faster because we know from Section 4.3 that the deadline misses in `md` and `stencil` are caused by insufficient time budget left after the slice finishes execution rather than mispredictions. This can be verified by looking at Figure 19, which shows the slice execution time for the HLS approach is much shorter.

Software-based Predictors.

Some accelerators have a software version with the same function, either because they are generated using HLS, or because they have a software implementation (e.g. ffmpeg for H.264). In these cases, instead of building hardware predictor, we can run a software predictor on the CPU to predict the execution time of the accelerator. We experimented with this idea on the H.264 decoder, and achieved good prediction accuracy. We do not include detailed results here due to the space limit.

5. RELATED WORK

5.1 Dynamic Voltage and Frequency Scaling

DVFS is a widely studied technique for reducing the energy of computation. For applications without response time requirements, simple interval-based scheduling algorithms [19] are often used. For example, the Linux power governors [9] are interval-based. However, interval-based algorithms do not perform well for workloads with large variability.

Researchers have studied DVFS in the context of hard real-time systems. One approach uses worst-case execution time (WCET) analysis of tasks to guide DVFS settings [38]. Although it guarantees that deadlines are met, it can be overly conservative since actual execution time can be much shorter than worst-case execution time. As a result, this approach is limited to hard real-time systems where deadlines must be strictly met.

DVFS has been studied in the context of resource management in datacenters to achieve energy proportionality, as well as controlling tail latency. PEGASUS [12] is a feedback-based DVFS controller that utilizes request latency statistics to make power management decisions. However, it only responds to slowly-changing workload variations. Adrenaline [17] uses workload metrics to predict tail queries for web services, and boosts DVFS levels accordingly. However, the workload metrics it uses are application-specific and manually identified, thus not generalizable to other applications.

DVFS has also been studied in the context of hardware accelerators. Linux implements interval-based governors in its `devfreq` framework [39] for controlling DVFS of hardware accelerators. As with other interval-based scheduling algorithms, these governors have the same issues when dealing with workloads that show large variability.

A number of studies have investigated using workload metrics to predict the execution time in order to inform DVFS decisions for interactive games [14], video decoding [22], and web browsing [15,16]. However, most of these studies use application-specific metrics that requires domain-specific knowledge to obtain.

A recent study investigated DVFS settings for mobile applications that use multiple hardware accelerators [18]. The key idea is to consider multiple devices together instead of individually. Their applications had little job-to-job execution time variation so a simple history-based approach worked well. However, we show that many applications have large job-to-job execution time variation. Thus our approach can potentially expand the applicability of their work.

5.2 Execution Time Prediction

Worst-case execution time analysis is a well-studied topic in hard real-time systems [40] and has been applied to DVFS for these systems [38]. WCET tries to calculate an upper bound of a job's execution time under all possible inputs. However, it does not estimate a job's execution time given a specific input.

Mantis [41] is an execution time prediction framework for smartphone applications, which uses automatically-extracted program features and thus is general across different applications. The high-level approach of Mantis is similar to our work. However, Mantis only considers software programs. Our work proposes an execution time prediction framework for hardware accelerators and investigates its application to DVFS.

6. CONCLUSION

In this paper, we presented a framework for generating DVFS controllers for hardware accelerators based on execution time prediction. Using features from hardware operation, the controller predicts a hardware accelerator's execution time and the appropriate DVFS level that minimizes energy while meeting response time requirements. Our approach achieves 36.7% energy savings on average for ASIC accelerators compared to running each accelerator at a constant frequency, with only 0.4% deadline misses. The energy savings are only 3.8% less than an optimal DVFS scheme. We also show that by introducing a boost level, the deadline misses can be completely eliminated while still achieving 36.4% energy savings. The DVFS controller has a low cost, adding only 5.1% area to accelerators on average.

Acknowledgments

This work was partially supported by the Office of Naval Research (ONR) grant N0014-15-1-2175. We thank the Batten Research Group and Apsel Research Group at Cornell University for their help in the circuit-level modeling of DVFS. We thank Daniel Lo for many insightful discussions, and also sharing his optimization infrastructure for the execution time prediction model. His work on prediction-based DVFS for processing cores provided insights to this work.

7. REFERENCES

[1] "Qualcomm Snapdragon 800 Product Brief." `https://www.qualcomm.com/documents/snapdragon-800-processor-product-brief`.

[2] "Inside the Apple A7 from the iPhone 5s." `http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-a7/`.

[3] "Chipworks Disassembles Apple's A8 SoC." `http://www.anandtech.com/show/8562/chipworks-a8`.

[4] C. Ju, T. Liu, K. Lee, Y. Chang, H. Chou, C. Wang, T. Wu, H. Lin, Y. Huang, C. Cheng, T. Lin, C. Chen, Y. Lin, M. Chiu, W. Li, S. Wang, Y. Lai, P. Chao, C. Chien, M. Hu, P. Wang, F. Yeh, Y. Huang, S. Chuang, L. Chen, H. Lin, M. Wu, C. Chen, R. Chen, H. Y. Hsu, and K. Jou, "A 0.5nJ/Pixel 4K H.265/HEVC Codec LSI for Multi-Format Smartphone Applications," in *2015 IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, 2015.

[5] "Cortex-A7 Processor." `http://www.arm.com/products/processors/cortex-a/cortex-a7.php`.

[6] J. Park, I. Hong, G. Kim, Y. Kim, K. Lee, S. Park, K. Bong, and H. Yoo, "A 646GOPS/W Multi-Classifier Many-Core Processor with Cortex-Like Architecture for Super-Resolution Recognition," in *2013 IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, 2013.

[7] G. Kim, Y. Kim, K. Lee, S. Park, I. Hong, K. Bong, D. Shin, S. Choi, J. Oh, and H.-J. Yoo, "A 1.22TOPS and 1.52mW/MHz Augmented Reality Multi-Core Processor with Neural Network NoC for HMD Applications," in *2014 IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, 2014.

[8] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. R. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Proc. the ACM/IEEE 41st Int'l Symp. on Computer Architecture (ISCA)*, 2014.

[9] D. Brodowski, "CPU Frequency and Voltage Scaling Code in the Linux^{TM}Kernel." `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`.

[10] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, "Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder," in *Proc. the 2002 IEEE/ACM Int'l Conf. on Computer-aided Design (ICCAD)*, 2002.

[11] Y. Gu and S. Chakraborty, "Control Theory-based DVS for Interactive 3D Games," in *Proc. the 45th Annual Design Automation Conf. (DAC)*, 2008.

[12] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards Energy Proportionality for Large-scale Latency-critical Workloads," in *Proc. the ACM/IEEE 41st Int'l Symp. on Computer Architecture (ISCA)*, 2014.

[13] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mossé, J. Mars, and L. Tang, "Octopus-Man: QoS-Driven Task Management for Heterogeneous Multicores in Warehouse-Scale Computers," in *Proc. the 21st IEEE Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[14] Y. Gu and S. Chakraborty, "A Hybrid DVS Scheme for Interactive 3D Games," in *14th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2008.

[15] Y. Zhu and V. J. Reddi, "High-performance and Energy-efficient Mobile Web Browsing on Big/Little Systems," in *Proc. the 19th IEEE Int'l Symp. on High-Performance Computer Architecture (HPCA)*, 2013.

[16] Y. Zhu, M. Halpern, and V. J. Reddi, "Event-Based Scheduling for Energy-Efficient QoS (eQoS) in Mobile Web Applications," in *Proc. the 21st IEEE Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[17] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, "Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting," in *Proc. the 21st IEEE Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[18] N. C. Nachiappan, P. Yedlapalli, N. Soundararajan, A. Sivasubramaniam, M. T. Kandemir, R. Iyer, and C. R. Das, "Domain Knowledge Based Energy Management in Handhelds," in *Proc. the 21st IEEE Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015.

[19] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," in *Proc. the 1st USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, 1994.

[20] "Samsung Exynos Linux Kernel Drivers." `https://github.com/hardkernel/linux/blob/odroidxu3-3.10.y/arch/arm/mach-exynos/include/mach/exynos-mfc.h`.

[21] V. Sze, D. F. Finchelstein, M. E. Sinangil, and A. P. Chandrakasan, "A 0.7-V 1.8-mW H.264/AVC 720p Video Decoder," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 11, pp. 2943–2956, 2009.

[22] M. Roitzsch, S. Wächtler, and H. Härtig, "ATLAS: Look-Ahead Scheduling Using Workload Metrics," in *Proc. the 19th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2013.

[23] C. Wolf, "Yosys Open SYnthesis Suite." `http://www.clifford.at/yosys/`.

[24] Y. Shi, C. W. Ting, B. Gwee, and Y. Ren, "A Highly Efficient Method for Extracting FSMs from Flattened Gate-level Netlist," in *Int'l Symp. on Circuits and Systems (ISCAS)*, 2010.

[25] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, 1996.

[26] E. Clarke, M. Fujita, S. Rajan, T. Reps, S. Shankar, and T. Teitelbaum, "Program Slicing of Hardware Description Languages," in *Conf. on Correct Hardware Design and Verification Methods*, 1999.

[27] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting Performance Impact of DVFS for Realistic Memory Systems," in *Proc. the 45th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, 2012.

[28] K. Xu and C. Choy, "Low-power H.264/AVC Baseline Decoder for Portable Applications," in *Proc. the 2007 Int'l Symp. on Low Power Electronics and Design*, 2007.

[29] W. Godycki, C. Torng, I. Bukreyev, A. Apsel, and C. Batten, "Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks," in *Proc. the 47th Int'l Symp. on Microarchitecture (MICRO)*, 2014.

[30] A. Beldachi and J. L. Núñez-Yáñez, "Run-time Power and Performance Scaling in 28 nm FPGAs," *IET Computers & Digital Techniques*, vol. 8, no. 4, pp. 178–186, 2014.

[31] "Video compression systems." `http://opencores.org/project,video_systems`.

[32] "JPEG Decoder in Verilog." `http://opencores.org/project,djpeg`.

[33] B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. M. Brooks, "MachSuite: Benchmarks for Accelerator Design and Customized Architectures," in *2014 IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2014.

[34] "AES (Rijndael) IP Core." `http://opencores.org/project,aes_core`.

[35] "SHA cores." `http://opencores.org/project,sha_core`.

[36] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System Level Analysis of Fast, Per-Core DVFS using On-chip Switching Regulators," in *Proc. the IEEE 14th Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2008.

[37] M. Weiser, "Program slicing," in *Proc. the 5th Int'l Conf. on Software Engineering*, pp. 439–449, 1981.

[38] D. Shin, J. Kim, and S. Lee, "Low-energy Intra-task Voltage Scheduling Using Static Timing Analysis," in *Proc. the 38th Annual Design Automation Conf. (DAC)*, 2001.

[39] M. Ham, "devfreq: Generic Dynamic Voltage and Frequency Scaling (DVFS) Framework for Non-CPU Devices." `https://github.com/torvalds/linux/tree/master/drivers/devfreq`.

[40] P. P. Puschner and A. Burns, "Guest Editorial: A Review of Worst-Case Execution-Time Analysis," *Real-Time Systems*, vol. 18, no. 2, pp. 115–128, 2000.

[41] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B.-G. Chun, L. Huang, P. Maniatis, M. Naik, and Y. Paek, "Mantis: Automatic Performance Prediction for Smartphone Applications," in *Proc. the 2013 USENIX Annual Technical Conf.*, 2013.