The Pennsylvania State University
The Graduate School

DACA: A FEEDBACK CONTROL-BASED APPROACH TO APPROXIMATE COMPUTING

A Thesis in
Computer Science and Engineering
by
Ohyoung Jang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2012

The thesis of Ohyoung Jang was reviewed and approved* by the following:

Mahmut Taylan Kandemir
Professor of Computer Science and Engineering
Thesis Advisor

Chita R. Das
Distinguished Professor of Computer Science and Engineering

Lee Coraor
Associate Professor of Computer Science and Engineering
Director of Graduate Affairs

*Signatures are on file in the Graduate School.

Abstract

Inaccuracy in computation has usually been considered with a negative connotation, and therefore, conventional computing systems have always been designed with a strict notion of correctness. However, inaccuracy or approximation is not always bad since several application domains are intrinsically tolerant to varying degrees of relaxation in accuracy, and thus, such a property can be exploited for significant gains in application performance or fault-tolerance. This concept, known as approximate computing or soft computing, has been recently applied to several application domains primarily for quantifying the tradeoffs between potential performance gains and accuracy losses. A weakness of all these studies is that the error bound is not strictly limited. The main contribution of this work is a feedback control based scheme, called DACA, to approximate computing that can be used to constrain the loss in accuracy, while maximizing potential performance benefits. We demonstarate the design of the feeback controller to dynamically actuate the approximate computing mecahanism using an object tracking application, Bodytrack. In addition, we also propose a roll-back technique to control large variance in inaccuracy that cannot be easily controlled by the feedback mechanism. Our evaluations indicate that the proposed approach can improve performance by as much as 260%, while guaranteeing inaccuracy limit between 10% to 30%.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The accuracy of computations is one of the invariants in conventional applications, and therefore, inaccuracy in computation has usually been taken with a negative connotation. In particular, most computer systems, security applications, and database systems are built on the strict computational accuracy in mind. However, a large class applications can tolerate varying degree of inaccuracy, especially when the quality of the solution can be interpreted by users. Accuracy measurements in such applications are occasionally expressed by relative terms. In the Fuzzy logic [1], for example, a room temperature can be represented by one of three words (hot, warm, cold), not always by a number in Fahrenheit or Celsius. Similarly, many error tolerant applications such as multimedia, graphics, and data mining have built-in redundancy in computations, and therefore, may achieve acceptable results if a part of the computations is skipped. Such approximation in computing is a promising approach to enhance application performance, energy consumption and fault-tolerance if exploited intelligently. Specifically, applications with big data sets or embedded applications with real-time/QoS constraints can benefit significantly from approximate computing.

Approximate computing (also known as soft computing [2, 3], probabilistic computing [4] and stochastic computing [5]) refers to a collection of methodologies exploiting the tolerant nature of imprecision, uncertainty, partial truth, and approximation to achieving low cost solutions [6]. Prior research explored application-

specific approaches aiming at improving performance at the cost of accuracy [6, 7]. Systematic methods for achieving low cost solutions have also been investigated through skipping computation in several ways [8, 9, 10]. All these prior efforts tried to remove redundancy in computation for better performance.

However, most of the existing work on approximate computing are oriented towards quantifying the tradeoffs between potential performance benefits and accuracy losses. They do not address how to control the performance knobs not to exceed a specified or accepted level of inaccuracy. We believe this is an important aspect of approximate computing since many well-understood applications may specify their tolerable performance margin and given such a margin, one has to develop appropriate techniques to trigger approximate computing. To the best of our knowledge, there is no prior attempt to control the dynamics of inaccuracy through the entire execution of an application to satisfy customized requirements.

In this paper, we propose a Dynamic Approximate Computing Architecture (DACA) that tries to guarantee a given inaccuracy bound, while improving performance as much as possible. This architecture skips computation by skipping iterations in a loop, and dynamically controls the amount of the skipped iterations to keep the resulting inaccuracy under a reference inaccuracy bound provided by a user. In this work, we employ formal control theory to regulate the states of our system. Feedback control theory enables us to control the properties of a system, which are otherwise highly dynamic. It is also very effective in minimizing disturbance in the system. This property is a unique advantage of feedback control theory for applications whose system characteristics are unknown. Specifically, in the context of approximate computing, the benefits of a feedback control based architecture include (i) dynamically controlling the amount of computation to skip over the course of execution; (ii) limiting the high variance in inaccuracy due to the dynamics of input data set and system conditions; and (iii) theoretically guaranteeing system stability, settling time to the bounding inaccuracy, and overshoot of inaccuracy.

We also propose a roll-back strategy in DACA to avoid occasional high variations in inaccuracy that are difficult to control using feedback control theory solely. These variations mostly occur due to a specific state of a target system or the corresponding optimization algorithms. In most cases, it is not possible to predict

the moment at which the high variation will occur. Our strategy tries to minimize high variations by selectively rolling back a few computations and repeating them without skipping computation. Despite the overhead of the roll-back policy, our results indicate that our control architecture with the roll-back strategy achieves high performance gain.

We demonstrate the feasibility and advantages of the DACA policy by using a running example of an object tracking application, Bodytrack, which tracks a human body through video streams. DACA, when applied to Bodytrack, achieves as much as 260% performance gain while bounding inaccuracy between 10% to 30%. We also show the generality of the scheme by applying it to a data-mining application-k-means clustering.

Chapter 2

The Dynamic Approximate Computing Architecture

In this section, we describe the technical details of DACA. The main part of this architecture is a *feedback control loop* that controls the inaccuracy during runtime, with the goal of maximizing the performance of the target application (Bodytrack in our case). Note that, the use of our proposed scheme is not limited to a specific application like Bodytrack and it can be applied to other error tolerant applications. DACA is composed of six main components illustrated in Figure 2.1. These components are: Skipping Controller, Target System (Bodytrack), InputStream Source, State History Table, Inaccuracy Calculator and Solution Validator.

In Figure 2.1, Bodytrack (*Target System*) receives the number of iterations to be skipped (k) from the skipping controller. Input images are fed to Bodytrack

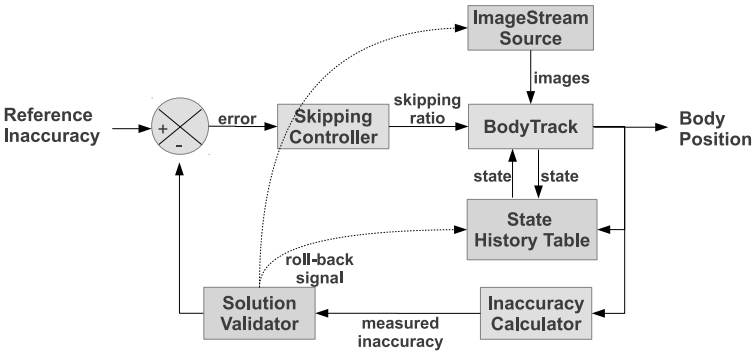


Figure 2.1: Feedback control DACA for Bodytrack.

from ImageStream Source, and Bodytrack approximately tracks a body position on the images by skipping k iterations in a loop, and Inaccuracy Calculator computes the inaccuracy with the tracked body position. In every frame, Bodytrack stores 4,000 particles and their likelihood in the State History Table and restores them when a roll-back signal is emitted from Solution Validator.

As will be discussed in Section 3, the inaccuracy can occasionally jump to high values, and therefore it may violate the maximum overshoot specification described in Section 4.1. This happens due to the unexpected change in the system state which is not captured by a statistical system identification strategy described in Section 4. Solution Validator detects this case by monitoring the measured inaccuracy, and emits roll-back signals to ImageStream Source and State History Table if the specification is not met. Receiving the first roll-back signal, the two components roll-back 1 frame, and feed images and states to Bodytrack again. The inaccuracy for the roll-backed frame is evaluated again, which is also monitored by Solution Validator. If it violates the specification again, Solution Validator sends another roll-back signal so that 4 frames are roll-backed; otherwise, DACA continues tracking a body in following frames. If the inaccuracy still fits the specification, 16 frames are roll-backed at last. The number of frames to roll-back is an adjustable parameter.

2.1 Feedback Control Theory

In this work, we employ a Single-Input, Single-Output (SISO) controller. A *closed-loop control system* with a SISO controller is illustrated in Figure 3.1. As an example of the closed-loop SISO controller, consider a cruise system on a car, which maintains the constant velocity of the car regardless of road conditions by controlling power that the engine generates. Suppose that the power is controlled by a *Controller* whose role is to adjust the engine power by controlling the engine's throttle position automatically. The desired velocity is called *Reference Input* in control theory terminology. A *Control Error* is calculated as the difference from the reference velocity and a measured velocity of the car (*System Output*) in this example. The controller takes the instantaneous error as input, and determines the amount of power generated by the engine in order to reduce the error as much

Terms	Phrases in DACA
Reference Input	Reference Inaccuracy
Control Input	Inaccuracy level we can more tolerate
System	Error tolerant application
System Output	Measured inaccuracy
Controller	Inaccuracy controller
System Model	the employed ARX model

Table 2.1: For terms in Control Theory and their corresponding phrases in DACA.

as possible. The *Transducer* is an optional component that converts the system output so that it can be compared with the reference input.

DACA can be applied to error tolerant applications that process large data sets. Skipping Controller periodically observes the difference between the reference inaccuracy (Reference Input) given by users and measured inaccuracy (System Output) generated by the target application, and advices the amount of computation to skip. For example, it may ask the application to skip more computation by 20% for the next period in execution if the measured inaccuracy is lower than the reference inaccuracy. The target application generates an approximated output which is later transduced into the same type as the reference input. Note that the controller knows the inaccuracy dynamics of the application as the amount of computation skipped is varied. The information is collected by building a system model in Control Theory. Table 2.1 parallels the formal terms in Control Theory to the corresponding phrases in DACA.

2.2 Inaccuracy Control

Skipping Controller follows the classic Single-Input and Single-Output controller [11]. It takes an error $e(i)$, and decides the amount of computation to skip, which is really the number of iterations to skip in Bodytrack, during execution for next frame. Error $e(i)$ represents the difference between reference inaccuracy Ref and a measured inaccuracy $q(i)$ at frame i:

$$e(i) = Ref - q(i).$$

We select a proportional-integral (PI) controller because it theoretically satisfies the performance specifications defined in Section 4.1. The discrete time domain form of the PI controller is:

$$u(i+1) = u(i) + (K_P + K_I)e(i) - K_P e(k-1),$$

where $u(i)$ is the number of iterations to skip at the frame i. K_P and K_I in this form are referred to as the controller gains, which are to be determined to meet the performance specifications of closed-loop systems.

Chapter 3

Example Application: Background

In this work, we use Bodytrack as our running example to illustrate the key concept we introduce. This computer vision application, from the PARSEC benchmark suite [12], tracks a human body through multiple video streams concurrently captured from several cameras. We use the *native* input set included in PARSEC, which contains four video streams with VGA (640x480) resolution, and 261 frames per stream. Figure 3.2 shows an example of how the tracked result for a frame is produced. A human body is modelled by ten cylindrical parts: head, torso, two upper legs, two lower legs, two upper arms, two lower arms. The estimated positions of individual parts are denoted using separate rectangles. Bodytrack takes as input the initial position of the person in the video stream from an external file, and generates the *estimated positions* of the person for every frame. It then outputs a vector of 31 numbers denoting the position, converts it to positions of 10 cylinders in 3D Cartesian coordinate system, and finally projects them onto four 2D images.

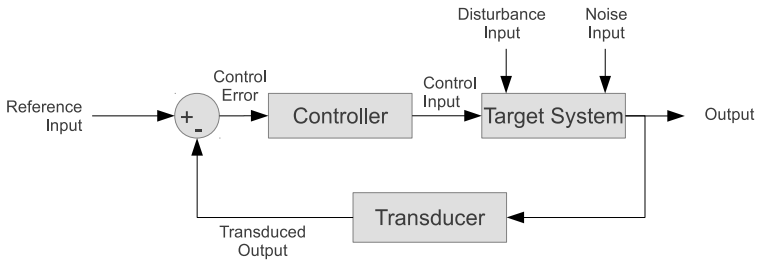


Figure 3.1: Single-Input Single-Output Control Model.

Inaccuracies in Bodytrack application is calculated by comparing the reference body position to the estimated position obtained by skipping computations. The reference body positions are attained from the execution of Bodytrack without skipping computation. Specifically, inaccuracies of a frame is the ratio of non-overlapped area between rectangles in the reference body positions and their corresponding rectangles in estimated positions. The non-overlapped ratio of each body on an image is first calculated, and the inaccuracy of a frame is acquired as the average inaccuracy of the 4 images in the frame, which can be formed as:

$$Inaccuracy = \frac{1}{4}\sum_{i=1}^{4}\left(1 - \frac{Area\left(r_1(i) \cap r_2(i)\right)}{Area\left(r_1(i) \cup r_2(i)\right)}\right)$$

where $r_1(i)$ refers to the rectangles of reference positions in i^{th} camera image on a frame, and $r_2(i)$ is its corresponding rectangles in the estimated positions. $Area(r)$, in the expression, computes the area of the rectangles r. Figure 3.3 compares the outputs of Bodytrack at a frame with several inaccuracy levels. Figure 3.3a shows the optimal body position obtained without any computation skipping up to the current frame. Note that identifying the distortion of outputs with the 10% inaccuracy is difficult. Note also that one might accept 30% or even 50% inaccuracy if one could obtain large performance benefits in return. Inaccuracy larger than 30% might be unacceptable, because the rectangle supposed to enclose a leg is off the leg. Because of this, we divide the inaccuracy range into three regions using relative terms: accurate(0% to 10%), acceptable(10% to 30%) and unacceptable($\geq$30%). In our experiments, we control the inaccuracy in the acceptable range.

Figure 3.4 shows the tradeoffs between performance and inaccuracy in this application. The amount of computation skipped is varied between 0% (original case) and 90%, and the observed inaccuracy/execution time values are recorded. The inaccuracy in this context is defined as the *average inaccuracy* through the entire video streams. We see that the normalized execution time decreases linearly as the amount of computation skipped increases, while the inaccuracy generally increases. For example, when we skip 60 percent of computation, the inaccuracy is slightly lower than 5%.

In addition to this intuitive observation, the *self-healing* nature of Bodytrack, as illustrated in Figure, 3.5 should also be taken into account to control the inaccuracy

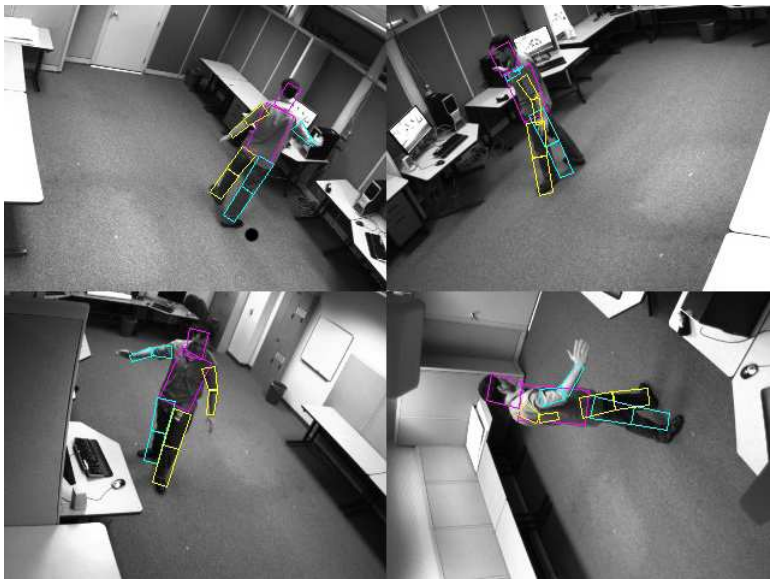
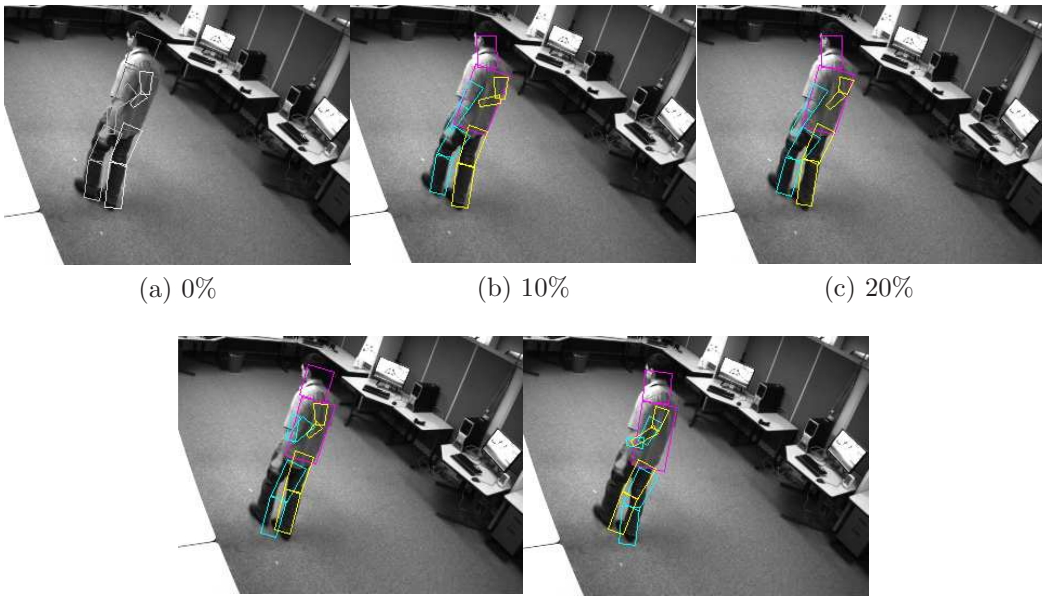


Figure 3.2: An example of tracking rectangles correspond to head, torso, upper legs, lower legs, upper arms, and lower arms.

over the whole execution of the application. In this figure, we skipped 90% of computation for the first 50 frames, which results in high inaccuracy of 32.4% at the frame 50. After that, the inaccuracy dramatically drops to less than 10% as we stop skipping computation.

Applications amenable to approximate computing exhibit three important characteristics: redundancy, adaptivity, and reduced precision [13], all of which Bodytrack possesses. Executing redundant computation may not improve the quality of the result. Skipping such redundancy leads performance benefits without a significant loss in accuracy. Further, since such workloads are designed with errors in mind, injecting a small error on purpose at a moment is expected to decrease to an acceptable level as an underlying algorithm progresses in its execution. In this paper, we inject artificial error by skipping computation, and *control* the amount of the error injected to maximize performance gain with an accuracy bound.

Simulated Annealing (SA), employed in Bodytrack, is an iterative algorithm to find a good solution to a global optimization problem in a large search space [14]. Objective functions are given to evaluate the quality of a solution. Prior to solving a problem, a large number of solutions are spread out through the search space. It is common to denote a solution by a list of numbers or a vector. At a high level,



(a) 0% (b) 10% (c) 20%

(d) 30% (e) 50%

Figure 3.3: Tracked body with inaccuracies: 0%, 10%, 20%, 30% and 50%

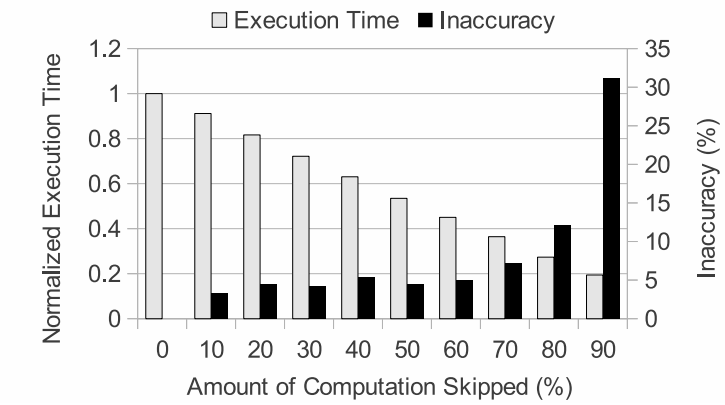


Figure 3.4: Performance-inaccuracy tradeoff. The x-axis indicates the amount of computation "skipped".

the operation of SA can be divided into three steps: (i) Random noise bounded by a threshold T are added to the solutions; (ii) The dirty solutions are evaluated by the objective functions; and (iii) High scored solutions are duplicated, while low scored solutions are eliminated so that the total number of solutions remains the same. After that, SA repeats these steps with smaller noise bound such as $T_{new} = alpha * T_{old}$, where $0 < \alpha < 1$. The iteration continues until estimated

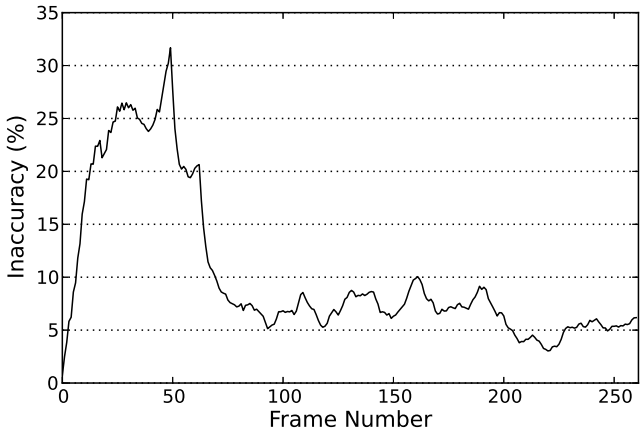


Figure 3.5: Self-healing: inaccuracy drops after stopping the computation skipping at frame 50.

solutions are good enough or we reach a pre-specified iteration limit.

In Bodytrack, 4,000 solutions, referred to as particles, are spread out through a solution space S, where $dim(S) = 31$. A particle forms a vector $\vec{p_i} \in R^{31}$. The "Update" function in "ParticleFilter.h" encloses the main loop of the SA algorithm, which iterates 5 times for estimating a body position at a frame with decreasing noise threshold $\vec{T} \in R^{31}$. In each iteration, the noise threshold $\vec{T}$, initially given from an external file, decreases by multiplying $\vec{T}$ with a scaling constant α, where $0 < \alpha < 1$. Particles are updated by adding uniformly distributed random values bounded by $\vec{T}$, and thus duplicated or eliminated according to their qualities, which are log-likelihood evaluated by an objective function. After the 5 iterations, the element-wise average of the 4,000 vectors is determined as a body position at the frame.

In our experiments, an inaccuracy occasionally jumps by a large degree, although it is quickly regulated by our control architecture. This occurs when a solution, possible body position, on the search space S moves close to a neighboring local optimum after polluted by random noise during the initial iterations of the main loop, and the rest of iterations are skipped. The solution would have adjusted close to the reference position if the rest of iterations are not skipped. Once the inaccuracy jumps happens, it violates the maximum overshoot specification described in Section 4.1 for a while. We avoid this severe inaccuracy jumps

by introducing a roll-back strategy which is a part of DACA.

Chapter 4

Controller Design

In this section, we present the details of how the controller is designed.

4.1 Performance Specification

In formal control theory, to design a controller, once the type of the controller is known, the next step is to determine the values of the controller parameters to control the behavior of the system. The followings are the main system characteristics that we consider in this work:

- System stability: A stable system generates bounded outputs when its control inputs are bounded. The number of iterations to skip in Bodytrack is bounded from 0 to 5, and therefore, the resulting inaccuracies have to be bounded.

- Settling time: T_s is the time it takes for a system output to converge to a given reference input. Our PI controller is designed to satisfy the settling time $T_s < 10$ frames.

- Maximum overshoot: M_p is the maximum amount that system output exceeds a steady state during a transient response. We set the upper bound of the maximum overshoot $M_p = 0.1$, meaning that the overshoot does not exceed a reference inaccuracy by more than 10%.

- Steady state error: In an accurate system, system output in steady state is equal to the reference input. PI controller theoretically guarantee the accuracy of a system if controller coefficients are properly selected.

4.2 System Identification

The primary difficulty in designing the controller in many computer systems lies in the lack of analytic models. Unlike physical systems, which is expressed by an explicit mathematical differential equations, it would not that easy to model computer systems analytically. As a result, *blackbox modeling* [11] is a common method to overcome the difficulty using statistical techniques. We assume that the number of iterations to skip is the primary factor to determine inaccuracy and to construct system models accordingly.

$$\begin{aligned} y(k+1) &= a_1 y(k) + \cdots + a_n y(k-n-1) \\ &+ b_1 u(k) + \cdots + b_m u(k-m-1) \end{aligned} \tag{4.1}$$

The *ARX model* shown above is an approximation of a non-linear system by a n^{th} order linear system around an operating point. We use the 1^{st} order approximation to express the dynamics of Bodytrack in inaccuracy, which captures sufficient information to design the Skipping Controller. The 1^{st} order ARX model is as follows:

$$y(k+1) = a_1 y(k) + b_1 u(k). \tag{4.2}$$

In Equation (4.2), $y(k)$ refers to the offset of the inaccuracy $\widetilde{y}(k)$ from the operating point $\overline{INACCURACY}$ at frame k, and $u(k)$ refers to the offset of number of iteration to skip $\widetilde{u}(k)$ from the operating point $\overline{SKIP}$. Operating points are constant points such that when Bodytrack skips $\overline{SKIP}$ iterations for a long time, the measured inaccuracy eventually converges to the $\overline{INACCURACY}$, which is called a *steady state*. The operating point $\overline{SKIP}$ is set to 2.5, the median of skip ranges from 0 to 5. The operating point $\overline{INACCURACY}$ is determined to 8.5, which is the average inaccuracy when 2.5 iterations are skipped in average.

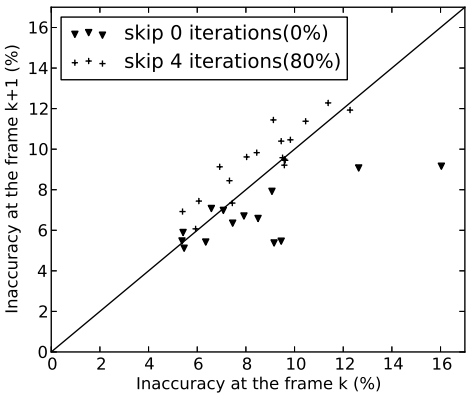


Figure 4.1: Inaccuracy changes when skipping 0, 4 iterations.

4.3 White Noise Input

To observe the dynamic behavior of our proposed system, we randomly choose the number of iterations to skip through the first 60 frames. For system identification with the blackbox model, white noise input method has been commonly used in control theory [15].

Figure 4.1 plots how the inaccuracy changes with the given number of iterations to skip. The horizontal axis and vertical axis denote the inaccuracy at k^{th} and $(k+1)^{th}$ frames, respectively. For example, the point on (x, y) denotes that the inaccuracy x at k^{th} frame changes to y at the next frame. A point over the 45° degree line indicates an increase in inaccuracy at the next frame. Similarly, a point under the line corresponds to a decrease in inaccuracy. As one can observe from Figure 4.1, the inaccuracy increases when we skip 4 iterations or 80 % of the total iterations, and decreases when no iteration is skipped.

4.4 Linear Least Square Estimator

In this section, we determine coefficients of the system model using Linear Least Square method [16]. This is a standard approach in estimating coefficients for a linear system which minimizes the *residual error*. This error can be expressed as follows for Equation (4.2),

$$r = \sum_{k=1}^{N} \left[y(k+1) - a_1 y(k) - b_1 u(k) \right]^2 . \tag{4.3}$$

By applying the white noise input, we obtain 60 equations in the form of Equation (4.2), and a Linear Least Square problem is obtained.

$$\vec{y} = A\vec{x}, \tag{4.4}$$

where $\vec{y} = (y_1, y_2, ..., y_{60})^T$, and $\vec{x} = (a_1, b_1)^T$. The *n-by-2* system matrix A is as follows:

$$A = \begin{pmatrix} y_0 & u_0 \\ y_1 & u_1 \\ \vdots & \vdots \\ y_{59} & u_{59} \end{pmatrix}$$

We solve this linear system problem by multiplying both sides of Equation (4.4) by the pseudo-inverse matrix $A^+ = (A^T A)^{-1} A^T$. Thus, we have $\vec{x} = (A^T A)^{-1} A^T \vec{y}$, and the two coefficients a_1 and b_1 in the vector $\vec{x}$ are determined by matrix calculations. It is well-known that the obtained coefficients a_1 and b_1 minimize the residual error [16]. The final solution of the coefficient vector $\vec{x}$ is determined as $(0.695, 0.645)^T$.

4.5 Controller Design

Having the system model given in Equation (4.2), the values of the controller parameters can be determined such that the controller performance specifications described in Section 4.1 are satisfied. The system model given in Equation (4.2) can be written as $G(z) = \frac{0.645}{z-0.675}$ in z-domain which is called *transfer function* in formal control theory terminology that relates the input and the output of the system in complex domain. Different characteristics of a control system can be obtained from the locations of the poles of the closed-loop system transfer function (the poles of a function are the roots of the equation when the nominator of the function is set to zero). Note that because our model and PI controller are first order systems, the close-loop system is a second order system, which, thus, has two poles we need to

located. We assume the poles are complex conjugate $re^{\pm j\theta}$. We choose the radius of the poles r such that $r < e^{-4/T_s} = 0.368$ which is known to be the upper bound of the radius that satisfy the settling time specification. We choose 0.35 for the radius r which also holds the stability specification as well because $r < 1$. We also decide θ to be 1.43 such that $\theta < \pi \frac{\log r}{\log M_p}$ to ensure the maximum overshoot specification. We, thus, construct a desired characteristic polynomial as follows:

$$z^2 - 2r\cos\theta z + r^2 = z^2 - 0.098z + 0.1225 = 0$$

The next step is finding two coefficients K_P and K_I by equating the characteristic polynomial with the denominator in transfer function of the closed-loop system $\frac{K(z)G(z)}{1+K(z)G(z)}$, where $K(z) = \frac{(K_P+K_I)z-K_P}{z-1}$. Finally, we choose K_P and K_I as 0.8876 and 1.588, respectively. The PI controller can be expressed as:

$$u(k+1) = u(k) + 2.47 * e(k) - 0.8876 * e(k-1).$$

Details of this procedure are described in [11].

Chapter 5

Experimental Evaluation

In this section, we evaluate DACA using Bodytrack application on an AMD-based quad-core machine.

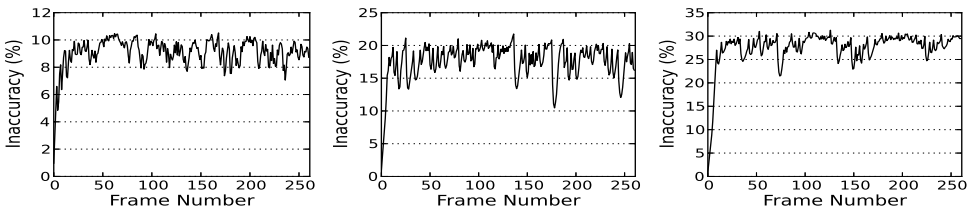
5.1 Model Verification

As we discussed earlier, the system model that we employ (as given in Equation (4.2)), captures the influences of the variations in control parameters $u(k)$ on the measured inaccuracy $y(k+1)$. To evaluate the accuracy of the model, we compare the predicted inaccuracy by the model $\hat{y}(k+1)$ and measured inaccuracy $y(k+1)$. In assessing the accuracy of the system model, the coefficient of determination R^2 and the mean absolute percentage error M are widely used. $R^2 \approx 1$ indicates the model is accurate; the accuracy of the model is reasonably acceptable with $R^2 > 0.8$. The M value is the average error of the model. These are calculated as follows:

$$R^2 = 1 - \frac{var(y - \hat{y})}{var(y)}$$

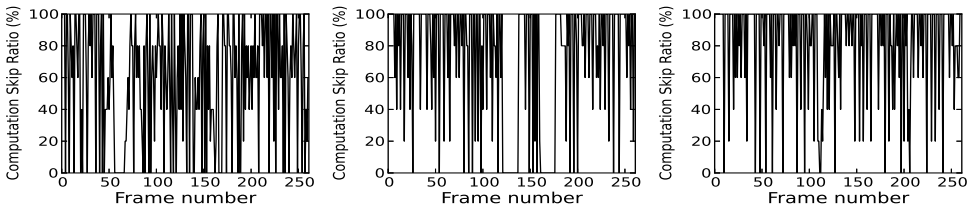
$$M = \frac{1}{N} \sum_{i=1}^{N} |\frac{y(i) - \hat{y}(i)}{y(i)}|$$

In our model, the value of the two coefficients R^2 and M are 0.893 and 0.087, respectively, which shows that our system model is accurate.



(a) 10% (b) 20% (c) 30%

Figure 5.1: Regulated inaccuracies with reference inaccuracies: 10%, 20%, and 30%.

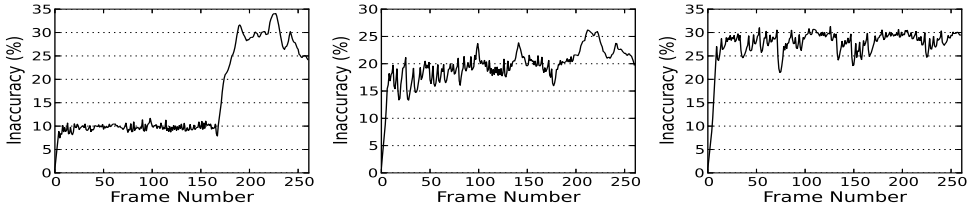


(a) 10% (b) 20% (c) 30%

Figure 5.2: The amount of computation skipped with reference inaccuracies: 10%, 20%, and 30%.

5.2 Inaccuracy Guaranteed Best Performance

In this section, we test DACA to quantity the performance benefits achieved with the guaranteed inaccuracy. All experiments shown below are performed with native input data set. Our experiment tries to control the inaccuracy between 10% and 30%. Figure 5.1 plots the dynamics of inaccuracy, with reference inaccuracies ranging from 10% to 30%. As can be observed from these results, our approach



(a) 10% (b) 20% (c) 30%

Figure 5.3: Regulated inaccuracies without roll-back strategy with reference inaccuracies: 10%, 20%, and 30%.

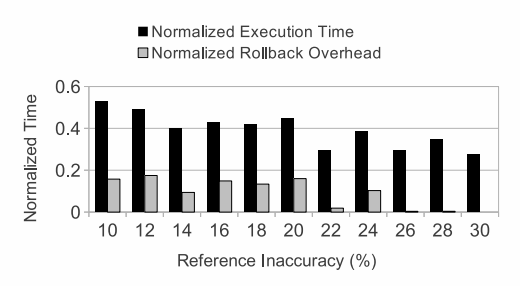


Figure 5.4: Normalized execution time and overhead of roll-back.

is able to limit the inaccuracy under a specified bound. Figure 5.2 shows the amount of computations skipped in percentage. In each frame, Skipping Controller skips computation from 0% to 100%(a body position is the same as the previous position) or 0 to 5 iterations in a loop. The skipping ratio largely fluctuates while the inaccuracies are regulated. On the other hand, the effect of roll-back strategy is presented in Figure 5.3. When DACA ignores roll-back signals for 10% reference inaccuracy, for instance, the inaccuracy is not appropriately regulated due to significant changes of internal states in Bodytrack.

Performance benefits due to our DACA is illustrated in Figure 5.4. All values in the figure are normalized to the execution time of intact Bodytrack, which does not skip computation. The normalized execution time with 10%, 20% and 30% inaccuracies are 0.52, 0.44 and 0.27 respectively. Thus, with 30% inaccuracy, DACA achieves speed-up of 3.6. In general, the execution time decreases as reference inaccuracy increases. The dominant reason is that the large inaccuracy margin in large reference inaccuracy lowers the chances of triggering roll-back signals, which significantly reduces the overhead of roll-back. On the other hand, Figure 5.5 compares the average inaccuracy and maximum inaccuracy to the reference inaccuracy through whole execution. The average inaccuracy is slightly less than the corresponding reference inaccuracy. For example, the average inaccuracy with 10% and 30% of reference inaccuracies are 9.25 and 28.26, respectively. This is because when no computation is skipped at a frame, the resulting inaccuracy tends to decrease more than the system model predicts. Note that the maximum inaccuracy does not exceed reference inaccuracies by more than 10%, holding the maximum overshoot specification.

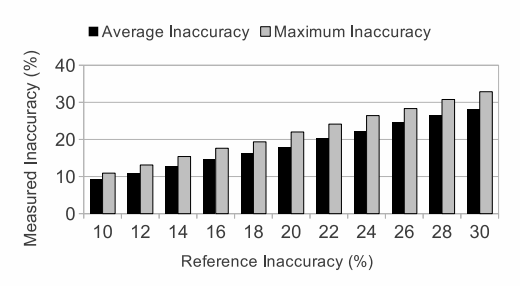


Figure 5.5: Average and maximum inaccuracies.

5.3 DACA for *k*-means

As stated earlier, DACA can be applied to applications from various domains, as long as (i) "inaccuracy" in an application can be defined, (ii) one can tolerate certain inaccuracy, and (iii) we have the knowledge to control the inaccuracy/performance tradeoff. For example, We applied DACA to a clustering algorithm, k-means, which is common in data-mining applications. k-means aims to partition n data points into k clusters, so that the distance between data points and the means of their clusters are minimized. In each iteration, k-means evaluates the membership of data points (partition that a point belongs to), and adjust the means of partitions accordingly. We skip computations in membership evaluation, which is the most time consuming portion, by assuming that data points, whose membership are changed at the previous iteration, will change at the next iteration. In every iteration, we first evaluate the membership of those data points, and then evaluate the rest of data point whose membership are not changed at the previous iterations. In this application, inaccuracy is defined as the fraction of data points whose membership changes (as compared to the original case). In monitoring the inaccuracy, k-means stops the iteration if the inaccuracy drops under a reference inaccuracy. Although now shown here due to space constraints, we observe about 24% improvement in performance when we keep the inaccuracy to 10% through DACA.

Chapter 6

Related Work

Prior work proposed approximate computing based methods targeting error resilient codes in a wide range of application domains including bio-informatics, material science, earth science, chemistry and data-mining [17, 18, 19]. In addition to these application-specific optimizations, prior work also explored programming language and compiler based techniques. For example, Rinard et al proposed a compiler framework, where computations are skipped through loop and code perforation, and through early-terminate at barrier synchronization points [8, 9]. In this work, a compiler generates codes for skipping computation around loops and thread barriers. Green [20] is a flexible compiler based system aiming at approximating computation at a function level. Two extra functions (an approximated function and a quality evaluation function) are supplied by a user, which are used to construct a quality model, and execute the rest of the program "approximately" at the cost of accuracy. The compiler based approximate computing methodologies have the simple assumption that the *loss of accuracy* depends on the amount of computation skipped. In practice, however, the loss of accuracy in the execution of an application with a large input data set fluctuates over the entire execution. Our proposed approach addresses the dynamics of the inaccuracy and controls the inaccuracy through the entire execution of an application.

[21] and [22] proposed a best-effort service model that can generate significant performance improvements in multicore platforms. In these works, dependencies across threads are relaxed, which eventually removes the necessity of thread communication and improves parallel speed-up of target applications. In addition to

these software based approaches, architectural supports for approximation for low-power consumption have been actively investigated [23, 24]; A set of instruction and implementations with low-power consumption are introduced. Truffle [25] proposes the dual-voltage operations, with a high voltage operations for precise operations and low voltage operations for approximate ones. Compiler techniques are usually accompanied to identify instructions (mostly from hints by users) which can be replaced for the proposed low-power instructions. Unlike all prior research, our work proposes a controlled architecture for approximate computing, where we honor the specified inaccuracy level throughout the entire execution of an application by dynamically skipping computation, and thereby improving performance.

Chapter 7

Future Works

Despite the high performance gain with limited inaccuracy, current implementation has limitations in practice. Inaccuracy Calculator component assumes that it has the best body positions for all frames, and it calculates inaccuracy based on the information. However, it is very unlikely that we would run an application again if we have the best result. To make our approach feasible in practice, inaccuracy can be obtained without the information.

One alternative way to infer an inaccuracy without comparing to the best result is predicting an inaccuracy using a statistical method, extrapolation. As described earlier, Bodytrack uses 4,000 particles or positions per frame in tracking a human body from which loglikelihood values are evaluated later. Using the loglikelihood value as a predictor for inaccuracy, we can predict the inaccuracy with larger number of particles. In this case, the control knob would be the number of particles rather than number of iterations as used in this paper. For example, if a user set the maximum number of particles as N, and at a frame, if the Bodytrack uses $\frac{4}{5}N$ particles at a frame, it is possible to calculate loglikelihood of estimated positions with $\frac{1}{5}N$, $\frac{2}{5}N$, $\frac{3}{5}N$, $\frac{4}{5}N$ particles. After that, we can extrapolate the data to predict the loglikelihood when it would be if N particles are used. Comparing the predicted loglikelihood to the measured loglikehood with $\frac{4}{5}N$ particles, we can estimate inaccuracy at the frame without the information of best positions. Figure 7.1 plots the loglikelihood over the number of particles, which gives us the intuition that loglikelihood is predictable out of the range in some extend. Loglikelihood inacreases as the number of particles increases, and it is almost linear if we focus

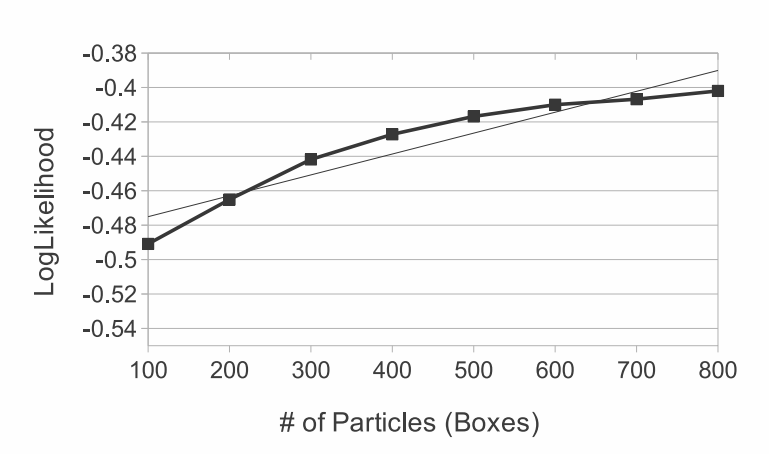


Figure 7.1: Loglikelihood over number of particles

on the region that the number of particles is larger than 500.

We are also interested in extending our soft computing scheme to hardware architecture. Specifically, on a S-NUCA system with large number of cores, shared L2 cache modules are spread over cores, which incurs the L2 cache latency varies depending on the distance between a core that issues a memory request and a cache that holds the requested cache line. Assuming all memory access evenly contribute to results, skipping the memory access whose latency is large benefits the performance in total execution cycles at most. Since memory access latency is tightly correlated to the distance which can be obtained by hardware before the request is issued, this idea will be feasible in practice, and the performance gain is expected to be high. Beside the load instruction, other costly instructions such as mul, div may be skipped or replaced for a low-power instructions with less accuracy. This instruction level approximation is proposed for GPGPU which extensively executes arithmetic instructions, but not for general purpose CPU yet.

Programmer should identify the code section in which instructions can be skipped or replaced, because not all the load instruction can be skipped. A single skip of load instruction may crash the application at all. Thus, it is the programmer's role to denote the code section, and thus a compiler decides the specific instructions to skip or replace according to the hint that a programmer gives. The hint may be expressed by a pragma in C language as:

```
#pragma approximate(p)
```

```
{
    statement 1;
    statement 2;
    ...
}
```

The p argument of the pragma indicates the portion of instructions to be skipped. For example, p is given by 0.2, 20% of load instructions accessing L2 cache are skipped. Once a compiler replaces the load instructions with load.approx instructions, it is decided on runtime by a CPU whether it is skipped or not according to the location of corresponding cache lines.

Chapter 8

Conclusion

In this paper, we propose a feedback control theory-based scheme, DACA, which regulates the inaccuracy of an error-tolerant application throughout the entire execution, and can therefore, achieve significant performance gains by controlled computation skipping. We propose a SISO controller to dynamically monitor the error margin and activate the compuation skipping based on the margin. The effectiveness of the DACA design is demonstrated using the Bodytrack application, where we can achieve at least 2 fold improvement in performance when the error margin is limited to 30% without affecting the quality of perception significantly.

Bibliography

[1] H. Zimmermann, *Fuzzy set theory–and its applications*. Springer, 2001.

[2] H. Nguyen and V. Kreinovich, "Towards theoretical foundations of soft computing applications," in *Proceeding of the Conference on Artificial Intelligence for Applications*, 1995, pp. 368–373.

[3] P. Bonissone, "Soft computing: the convergence of emerging reasoning technologies," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 1, no. 1, pp. 6–18, 1997.

[4] A. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *Proceeding of the Annual Symposium on Foundations of Computer Science (FOCS)*, 1977, pp. 222–227.

[5] N. Shanbhag, R. Abdallah, R. Kumar, and D. Jones, "Stochastic computation," in *Proceeding of the Design Automation Conference (DAC)*, 2010, pp. 859–864.

[6] S. Mitra, S. Pal, and P. Mitra, "Data mining in soft computing framework: A survey," *IEEE transactions on neural networks*, vol. 13, no. 1, pp. 3–14, 2002.

[7] S. Mitra and Y. Hayashi, "Bioinformatics with soft computing," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 36, no. 5, pp. 616–635, 2006.

[8] M. Rinard, A. Agarwal, S. Sidiroglou, S. Misailovic, H. Hoffmann *et al.*, "Using code perforation to improve performance, reduce energy consumption, and respond to failures," 2009.

[9] M. Rinard, "Using early phase termination to eliminate load imbalances at barrier synchronization points," in *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications (OOPSLA)*, vol. 42, no. 10, 2007, pp. 369–386.

[10] J. Meng, S. Chakradhar, and A. Raghunathan, "Best-effort parallel execution framework for recognition and mining applications," in *Proceeding of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2009, pp. 1–12.

[11] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback control of computing systems*. Wiley Online Library, 2004.

[12] C. Bienia, S. Kumar, J. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[13] X. Li and D. Yeung, "Exploiting soft computing for increased fault tolerance," in *Workshop on the Architectural Support for Gigascale Integration (ASGI)*, 2006.

[14] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, p. 671, 1983.

[15] L. Ljung, *System Identification*. Wiley Online Library, 1999.

[16] L. Richard, "Burden and j. douglas faires," *Numerical Analysis*, 2004.

[17] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J. Andre, D. Barkai, J. Berthou, T. Boku, B. Braunschweig *et al.*, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, p. 3, 2011.

[18] S. Ahern, S. Alam, M. Fahey, R. Hartman-Baker, R. Barrett, R. Kendall, D. Kothe, O. Messer, R. Mills, R. Sankaran *et al.*, "Scientific application requirements for leadership computing at the exascale," Technical report, Oak Ridge National Laboratory. http://www. nccs. gov/wp-content/media/nccs_reports/Exascale_Reqms. pdf, Tech. Rep., 2007.

[19] T. Hinke and J. Novotny, "Data mining on nasa's information power grid," in *Proceedings of the International Symposium on High-Performance Distributed Computing (HPDC)*, 2000, pp. 292–293.

[20] W. Baek and T. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, vol. 45, no. 6, 2010, pp. 198–209.

[21] J. Mengte, A. Raghunathan, S. Chakradhar, and S. Byna, "Exploiting the forgiving nature of applications for scalable parallel execution," in *Proceeding of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[22] S. Chakradhar and A. Raghunathan, "Best-effort computing: re-thinking parallel software and hardware," in *Proceedings of the Design Automation Conference (DAC)*, 2010, pp. 865–870.

[23] V. Gupta, D. Mohapatra, S. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *Proceeding of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2011, pp. 409–414.

[24] J. Liang, J. Han, and F. Lombardi, "On the reliable performance of sequential adders for soft computing," in *Proceeding of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2011, pp. 3–10.

[25] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.