On Accuracy-Performance Tradeoff Frameworks for Energy Saving: Models and Review

Chunbai Yang, Changjiang Jia, W. K. Chan, Y. T. Yu
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
{chunbyang2, cjjia.cs}@gapps.cityu.edu.hk, {wkchan, csytyu}@cityu.edu.hk

Abstract—**A class of research, which explores energy saving via the accuracy-performance tradeoff ability of applications with soft outputs, is emerging. Many such applications implement various notions of precision improvement in their iterative computations and output approximated results. Computing more precisely often requires greater computational effort; hence there is a tradeoff space between precision and cost as long as the output accuracy remains within a targeted bound. This paper reviews the state-of-the-art accuracy tradeoff frameworks, and presents abstract model representations to facilitate our comparison of these frameworks in terms of robustness, performance, profiling, run-time feedback control, and specification of accuracy metrics.**

Keywords—accuracy-performance tradeoff; accuracy-aware computing; energy-aware computing; programming framework

I. INTRODUCTION

Energy efficiency is becoming a first class citizen in the design of computing systems ranging from small mobile devices to large data centers. Based on the data over the past sixty years, Koomey et al. [9] showed that the electrical energy efficiency of computing systems roughly doubled at every eighteenth month. Koomey [8] further reported that the electricity consumed by computing systems worldwide doubled from 2000 to 2005 and then increased by 56% in the subsequent five years. In the year 2010 alone, they projected data centers to consume 1.1−1.5% of the electricity expended worldwide. The electricity sector is a hundred of billion dollar business. Even a fractional saving in electricity consumption would yield huge economic benefit.

Many studies have proposed methods to reduce energy consumption at different architectural levels and in different application domains. Examples include new designs of flip-flops and latches (at the circuit component level), the use of dynamic voltage and frequency scaling technology [7] to lower the processor clock speed (at the processor level), energy efficient protocol designs (at the network level), and new server scheduling algorithms to maximize the number of idle power-consuming nodes and their durations (at the grid computing level). Parthasarathy [11] summarized the ideas of these proposals as methods to look for energy usage inefficiency in a particular application domain and then optimize the solutions accordingly for energy saving.

Soft computing is a class of applications that compute approximated results by best effort. Computing better results tends to correlate with greater computational effort via, for example, executing more rounds of iteration of certain pieces of code to compute more accurate intermediate data or make more attempts to search for better solutions. Many application domains like video and image encoding, NP-hard problems, genetic algorithms, show soft computing qualities.

One approach (e.g., [2]) to address the energy concern is to perturb the control flow of a soft computing application to reduce the effort required to compute outputs that could be in the proximity of the original outputs over the same inputs. This approach attempts to generate versions of the application that yield different levels of output accuracy, unlike classical attempts that always aim at achieving the same output accuracy. In this paper, we re-examine the recently proposed energy-saving programming frameworks with accuracy-energy tradeoff ability in this approach.

We first present a generalized and abstract model to facilitate a systematic comparison of existing frameworks in terms of robustness, performance, accuracy profiling, run-time feedback control, and specification of accuracy metrics. We also outline our viewpoints to address the challenges identified as we present the frameworks.

The rest of this paper is organized as follows. Section II reviews the scope of applicability of existing frameworks. Section III formalizes the problem, proposes an idealized model, identifies its limitations and revises it into an approximation model. Section IV elaborates various dimensions of the approximation model based on which existing work is then reviewed. Section V concludes the paper.

II. BACKGROUND

A. *Potential types of applications to be benefited*

Existing work such as [2][10][14] identified that the applications with a range of acceptable output quality rather than just a binary one (accept or reject) are suitable for studying accuracy tradeoff. These applications typically can tolerate inaccuracy in the output to a certain extent.

For instance, an application may spend more rounds of iteration in a regression inference to obtain a more accurate dataset or make more (randomized, systematic, or heuristic) attempts to search for a better solution. At the tail part of the iterative process, executing a large amount of statements may only marginally improve the overall result. A common accuracy tradeoff strategy is to identify and remove some low-impact iterations by perturbing the application control flow. Thus, applications with many loop structures are candidates that may benefit from such frameworks.

Other potential candidates include applications that use the divide and conquer strategy. For instance, a small amount of incorrectly rendered pixels in a picture may not cause a substantial effect on the overall quality of the picture.

B. *Energy-saving utility for four types of computing systems*

To better understand energy saving, we also examine the energy consumption patterns of different types of computing systems to identify the applicability of accuracy-performance tradeoff frameworks. Computing systems can be roughly classified [3] into four types: (1) stationary, small scale computing systems, (2) mobile devices, (3) server clusters, and (4) high performance computing clusters. These computing systems differ greatly in their energy consumption components, costs and constraints.

As illustrated in Fig. 1, different types of devices within the category of data centers may have quite different power consumption distributions. With such a large heterogeneous infrastructure, it is reasonable to examine in details the workload models, energy consumption modes and other energy-related attributes in different categories.

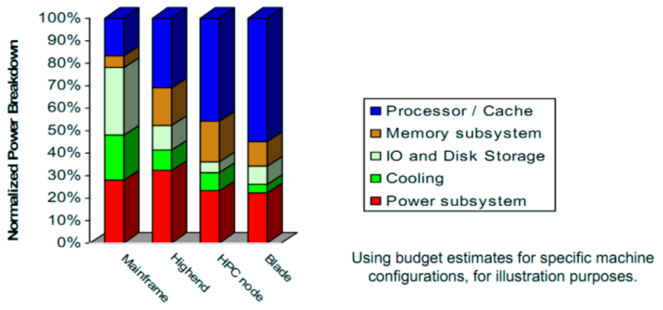

Figure 1. Normalized power breakdown for different types of machines in data centers (reproduced from [3])

1) *Stationary, small scale computing systems*

Personal computers are typical stationary, small scale computing systems. Such a computing device is usually static in terms of location mobility and has a direct access to a relatively long and stable period of power supply. As such, energy is often not a critical constraint during its operation. The accuracy tradeoffs in these computing systems are often not aimed at reducing energy consumption but rather at improving the performance or reducing the response time.

2) *Mobile devices*

The computing capabilities of mobile devices such as smart phones and tablet computers have been increasing rapidly, whereas improvement in energy storage capability of portable batteries has not been able to keep pace with the corresponding increase in power requirement. Energy hence has become a prime concern in the hardware and software designs of these devices. In terms of operating patterns, mobile devices are inactive in most of their time. Many research efforts focus on lowering the energy consumption at idle time.

An analysis [4] on smart phone usage shows that GSM module and display account for the majority of power consumption. As such, it is relatively difficult to achieve significant energy saving by applying an accuracy-energy tradeoff framework to reduce the computation performed by an application. Several studies focus on exposing accuracy tradeoff from the hardware to software level with extended operating system support to conserve energy [10][12][13].

3) *Server clusters*

Data centers are being built worldwide. The power costs, including electricity bills and the money spent on heat dissipation, cover a large proportion of the total operational costs of data centers. Hence, using an accuracy-energy tradeoff to save energy is attractive to data center operators.

A server cluster usually performs a large number of computations and holds a large volume of data. There is a great potential for harvesting energy by providing less accurate services (in terms of QoS).

Service requests to a server cluster are usually not evenly distributed and, hence, the workload typically fluctuates between peak and valley loads from time to time. To ensure quality of services, data centers are usually over-provisioned. Many computing nodes of a server cluster are kept idle, only to cater for the top loading period. These idle nodes consume energy even when they stand by, yet they contribute little to the computation tasks to fulfill all the service requests. As shown in Fig. 2, the average CPU utilization is usually very low. However, energy consumption is not proportional to its utilization. Hoffmann et al. [6] presented a system that attempted to trade output accuracy for sustaining response time at peak loads. It dynamically tunes the configuration variables to exploit the accuracy-energy tradeoff.

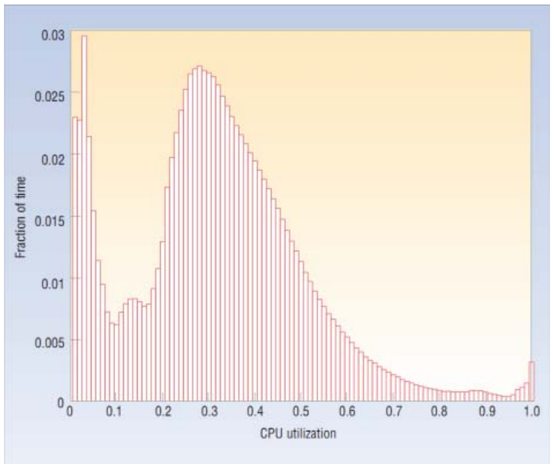

Figure 2. Average CPU utilization of more than 5,000 servers during a six-month period (reproduced from [3])

4) *High performance computing clusters*

A typical goal of high performance computers is to handle computationally-intensive batch tasks, which usually do not require a strict response time and their operating environments are relatively stable. These properties can be exploited for accuracy tradeoff.

III. Accuracy Tradeoff Models

To facilitate the discussion of the accuracy tradeoff problems and the existing solution frameworks, we first present an idealized model (which we call I-Model) for the problem behind these frameworks. Then we present an approximation model (which we call A-Model) that generalizes the approaches adopted by existing work to relax the limitations of I-Model.

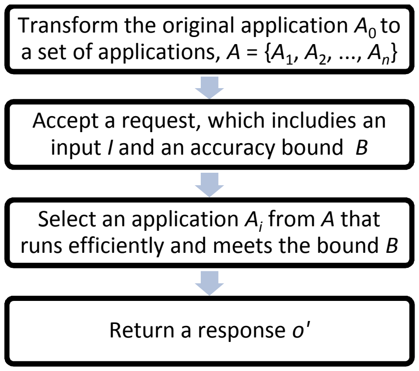
Transform the original application A_0 to a set of applications, $A = \{A_1, A_2, ..., A_n\}$
Accept a request, which includies an input I and an accuracy bound B
Select an application A_i from A that runs efficiently and meets the bound B
Return a response o'

Figure 3. Basic procedure of an accuracy tradeoff framework

A. I-Model: An Idealized Model

1) The Problem

The workflow of an accuracy tradeoff framework F is illustrated in Fig. 3 and described as follows. Given an application A_0, the framework F first generates $A = \{A_1, A_2, \ldots, A_n\}$, a set of adapted versions of A_0 with different levels of accuracy. Notice that it is equivalent to consider the set of adapted applications A as a single abstract one with a set of configuration variables X to distinguish each single version. After the transformation, F accepts a service request, manipulates the configuration variables X and efficiently produces a result, which is expected to satisfy a given output accuracy bound. A high level specification of the problem is expressed in equation (1),

$$\begin{aligned} F_{\mathrm{T}}(A_0) &= A \\ F(I,B,A) &= o' \end{aligned} \tag{1}$$

where F_{T} is the transformation process, F is the accuracy tradeoff framework, I is the given input, B is the bound on the output accuracy that the adapted application is expected to meet, and o' is the response to service requests, which is efficiently produced while meeting the bound B.

This problem may be divided into two sub-problems: the construction of an accuracy tradeoff space, namely A and the search of the configuration variables of A (that is, the particular version of A) within the constructed space. We discuss these two sub-problems in the next two subsections.

2) Construction of accuracy tradeoff space

The first sub-problem is to construct the accuracy tradeoff space. The framework F firstly identifies and extracts the tunable configuration variables of the application A_0 which impact on the performance of request processing as well as the response accuracy. This can be done directly by programmers who use certain accuracy-varying frameworks to develop A_0 [1][2][5][10][13], or F may automatically generate such versions from A_0 [6][12][14]. It is assumed that A_0 has the potentials to trade some extent of output accuracy for performance gain, and the potentials can be controlled by a set of configuration variables tunable by F. A high level specification of A is expressed in equation (2),

$$A(I,X) = o \tag{2}$$

where A and I denote, respectively, the given application and input in equation (1), and X denotes the set of configuration variables tunable by the framework F. For instance, X can be replaceable algorithms with different accuracy, differently perforated loops or different configuration values for some accuracy-related variables. The symbol o means collectively the output and the system behavior of A over the input I under the setting X. Notice that here o in equation (2) differs from o' in equation (1) in that the former does not guarantee to meet any accuracy bound B or efficiently process I. This induces the search presented in the next subsection.

Let us use an i-dimensional vector $\langle I_1, I_2, \ldots, I_i \rangle$ to stand for the input I and an n-dimensional vector $\langle X_1, X_2, \ldots, X_n \rangle$ to stand for the set of configuration variables X. Then a space of i+n dimensions is induced. Each point in this tradeoff space corresponds to an output of the application. The framework also requires a metric M to assess o that measures the output accuracy and performance of producing the outputs. In equation (3) below, the variables a and p denote the accuracy and performance assessed by M, respectively.

$$M(o) = \langle a, p \rangle \tag{3}$$

Note that we have simplified the equation (3) to assume that the measurement can be compared to the corresponding measurement on the version A to determine output accuracy.

3) Search for the most efficient configurations

Combining equations (2) and (3), we get equation (4).

$$M(A(I,X)) = \langle a, p \rangle \tag{4}$$

The second sub-problem is a typical search problem. The framework searches, within the constructed space, for the set of configuration variables X so that A has the best performance under the constraint due to the bound B.

$$\text{Goal: To search for an instance of } X \text{ with } max(p) \text{ satisfying the constraint } a \geq B \tag{5}$$

In summary, equation (1) is transformed into equation (4) and goal (5). We name the model as I-Model. To find an optimal solution, I-Model performs an exhaustive search over the accuracy tradeoff space, assesses the performance and accuracy of each point in the constructed space, and reports the most efficient result.

B. Approximation model

1) Limitations of I-Model

I-Model has several limitations. First, during the search in the goal (5), the accuracy tradeoff space can be too large to be explored exhaustively, and the total cost of assessments shown in equation (4) can be impractically huge. That is, performing an on-demand search over the accuracy tradeoff space is challenging with possibly unaffordable run-time overhead. A workaround solution is to perform the searches before run-time. However, the input I of the application is only known at run-time and, in most cases, enumerating every possible input is still unaffordable. At the same time, for applications with volatile run-time accuracy requirement, B is also only known at run-time. Without knowing I and B, the accuracy tradeoff space cannot be searched efficiently. Hence, I-Model needs to be adapted for practical use.

2) *Accuracy tradeoff space projection*

We now present an approximation model, which we call A-Model. In order to bring forward the search prior to run-time, all the unknown factors, namely I and B in equation (1), should be either eliminated from the search space or reduced to certain restricted and enumerable ranges of values: the dimensions of input I are removed by projection, and the accuracy bound B is replaced by a set of ranges.

In A-Model, the original $i+n$ dimensions of I-Model are projected into an n-dimensional space, keeping only the set of configuration variables X. This projection is realized at a training phase. Sample inputs are executed with a fixed set of configuration variables, say X'. The corresponding outputs are then used to estimate the population of the results for X' which is just one point in the n-dimensional space. In this way, an estimation function $\hat{A}$ is generated for the entire n-dimensional space as the counterpart of A in equation (4). This estimation function estimates an accuracy $\hat{a}$ and performance $\hat{p}$ via the measurement function M without considering the value of I, as expressed in equation (6) below.

$$M(\hat{A}(X)) = \langle \hat{a}, \hat{p} \rangle \qquad (6)$$

For instance, the framework F may implement $\hat{A}$ by interpolating the results of the input samples, or simply by calculating the mean values from these results.

3) *Search for the best configuration X*

The search in A-Model differs from the one in I-Model. In I-Model, the goal of a search is to find an optimal response to the service request, whereas in A-Model the goal is to find the best values for the set of configuration variables X that result in the best estimated adapted version. Hence, the goal of the search is revised as follows.

Goal: To search for an instance of X with $max(\hat{p})$ satisfying the constraint $\hat{a} \geq B$ (7)

4) *Accuracy tradeoff profile*

Considering equation (6) and goal (7) together and making the search transparent, we obtain a *profile function P* as follows,

$$P(B) = X \qquad (8)$$

where P generates a vector X when given the accuracy bound B. Here X provides an estimation of the best adapted version according to training data.

Through projection of the accuracy tradeoff space, the issue of unknown I is alleviated. Now we look at let us consider the issue of unknown B.

The accuracy bound B may be provided at compile-time or at run-time. For example, when porting an application from one operating environment to another with non-identical and static accuracy or performance requirements, such as transplanting a program from running on a stationary computer to be run on a server machine, the profile P in equation (8) can be determined before run-time if the accuracy bound B does not vary at run-time.

If B can only be provided at run-time, existing work usually provide a set of enumerable accuracy levels for users to choose (say, via different service level agreements). Then P can be computed at every accuracy level in advance.

In summary, (1) in I-Model is now approximated by equations (2) and (8) in A-Model. Because equation (8) has the potential to be computed before run-time, A-Model may adjust the behavior of the application to deal with different accuracy requirements at run-time.

IV. Dimensions and Challenges

The A-Model has captured certain essential aspects of many accuracy tradeoff frameworks. However, different frameworks still vary diversely in details. In this section, we discuss certain dimensions of the model and investigate what the existing problems and challenges in these dimensions are. We will also discuss the solutions proposed by existing work and examine their merits and drawbacks.

A. *Accuracy tradeoff approaches*

One key concern for accuracy tradeoff frameworks is how to build a set of adapted applications A in equation (1) with different accuracy levels. The ability to provide variable accuracy is intrinsic to the problem that the applications are designed to solve but not how they are constructed. Applications with variable output accuracy differ from those with fixed accuracy in their attempts to expose and manage the accuracy tradeoff ability. Existing work has proposed many different approaches to explore this capability and construct the accuracy tradeoff space. We classify these approaches into two broad categories: application level approaches and software-hardware collaboration approaches.

1) *Application-level approaches*

Approaches in this category explore the tradeoff ability purely at the software layer. They aim at finding the built-in low-impact computations at execution. Energy consumption is cut down by decreasing the total amount of computations. Pure application-level approaches have the advantages of being transparent to lower layers such as operating systems and hardware. Such an approach assumes that the given applications can be adapted to provide variable-accuracy behavior, and they can run on general computing systems without the need to use specialized hardware. However, application-level approaches usually are highly coupled with special computation patterns, which severely limit the scope of their applicability in practice.

Baek et al. [2] propose a system to allow programmers to define precise functions and their replaceable approximate functions. The system also supports loop skipping, such as skipping the trailing portion of iterations. By defining these approximated functions and loops, the system facilitates programmers to specify the accuracy tradeoff ability of their programs. Ansel et al. [1] develop a new language and compiler to support the specification of five categories of variable accuracy choices: variable algorithmic choices, variable cutoff values, switches of data storage types, synthesized functions to determine values from dynamic input, and configuration of user-defined parameters.

Hoffmann et al. [6] design a semi-automatic system to expose all the configuration variables of an application facilitated by programmers. Their system identifies a set of accuracy-related variables from an application and constructs dynamic knobs to tune these variables. As such, their system transforms a manually-tuned application into a dynamically-tuned one in which the involved configuration process is performed automatically at run-time. Sidiroglou et al. [14] perform loop perforation to automatically identify redundant loops in an existing application. The combinations of different perforated loops with different perforated rates create the accuracy tradeoff space of the application.

2) *Software-hardware collaboration approaches*

Software-hardware collaboration approaches expose the accuracy tradeoff ability between hardware and software. When constructing the application, programmers can directly control the performance of hardware to utilize the accuracy tradeoff. There are several hardware techniques supporting variable accuracy or reliability, such as dynamic voltage frequency scaling [7], width reduction in floating point operations, and different DRAM refresh rates [10]. An advantage of general software-hardware collaboration is that it has the potential of achieving more effective tradeoff than using a pure software approach. However, in order to expose these accuracy-related hardware controls, hardware layer, OS layer and application layer all need specialized supports.

Liu et al. [10] utilize different DRAM refresh rates to save refresh-power requirements of memory chips. Their technique allows programmers to partition data into critical and non-critical ones that can be stored in separate regions of the memory chips operating at different refresh rates. For non-critical data, the application should be able to tolerate memory errors due to reduction in refresh rate. In essence, their technique trades the reliability of memory data (which can be seen as accuracy from the perspective of the application) for energy saving. Pekhimenko et al. [12] propose a system which predicts the rates of cache misses so as to reduce the overall memory latency perceived, thus creating an accuracy tradeoff space for the application.

EnerJ [13] extends Java to support the definitions of approximated data storage, computation and algorithms. Programmers can also specify approximated data types or classes (which can contain approximated member functions) in addition to traditional ones. To support this language, a hardware architecture is developed utilizing many hardware techniques such as voltage scaling, DRAM refresh rate, and SRAM supply voltage.

B. Robustness and performance: tackling A-Model failures

When trading accuracy for energy or performance, failures may occur because an application usually can tolerate a certain but not the full extent of low accuracy or may result in deadlocks, crashes or other issues, such as violation of the required performance (say, QoS constraint).

The sources of failures lie in the accuracy tradeoff space specified in equation (2). First, some points in the space may lead to crashes, deadlock or nonsensical outputs, which can be modeled by outputs with an accuracy bound.

Second, in A-Model, the produced output is only a best estimation. In general, there is a difference between the best estimated output and the actual output due to the uncertainty in two steps. The first step is the projection of the accuracy tradeoff space defined in equation (6), which yields a profile based on partial information of the application. The second step is the search specified in goal (7). Since the accuracy tradeoff space can be too large for practical search even after projection, the result is in most cases sub-optimal.

It is worth noting that, due to the nature of soft computing, one cannot precisely define the concept of failure in absolute terms until the user provides an accuracy bound.

On the other hand, there are points in the accuracy tradeoff space which definitely lead to failures, such as those giving results with zero or even negative performance gain, or those causing the application to crash. We refer to those objectively-defined failures as *critical failures* and all other failures as *soft failures*. Intuitively, critical failures are concerned with the robustness issue while soft failures are more concerned with the performance issue.

1) *Robustness issue: critical failure*

Critical failures are obviously not tolerable. They cause unacceptably large deviation from the desired outputs (even though we may not know the expected outputs) or damage the performance as well as the accuracy. Unlike soft failures, critical failures may be handled before run-time, mainly by adopting approaches that fall into two categories.

The first category monitors potential critical failures when constructing the accuracy tradeoff space. This is usually achieved by heuristics, namely the design of code by programmers, facilitated with mechanisms like type check and approximated data flow control. These approaches try to minimize the potential failure points in the accuracy tradeoff space and are widely used in frameworks which require manually created applications A defined in equation (1).

The second category attempts to develop a strategy to remove the potential failure points after constructing the accuracy tradeoff space. These approaches are widely used in frameworks which create the set of adapted applications A with variable accuracy level. They heavily rely on the training set that generates estimated values in equation (6). During the training phase, these approaches assess and filter out the values of the configurable variables X that potentially lead to failures.

The *Green* framework [2] defines a set of specifications to which programmers should conform when specifying the accuracy tradeoff. Other robustness issues due to the accuracy tradeoff are left to be handled by the programmers. As the rigid specification in *Green* allows very little flexibility, programmers can provide a relatively good guarantee of robustness. Ansel et al. [1] extend the flexibility of their specification and support finer granularity in specifying the accuracy tradeoff than what *Green* does. However, the robustness and correctness of the application is still left to be manually handled by the programmers. As such, the approach may suffer from some unknown potential robustness issues, which pose a threat to applications that use their framework. *EnerJ* provides finer granularity with approximate language components to specify the accuracy tradeoff. In order to reinforce robustness, *EnerJ* performs static type check. Besides, *EnerJ* proposes an explicit control named *endorsement* from approximated data to precise data. It further forbids the use of approximated data in predicates of decision statements in a program. These features greatly alleviate the safety issue. *Fikker* [10] leaves the responsibility of correctly partitioning critical data from non-critical data to programmers. As *Fikker* uses an aggressive approach of trading reliability of data for performance, wrong partitioning of critical data is possible, which may result in critical failures and, hence, is undesirable.

The *PowerDial* [6] system is related to the second category as well. It asks the user to provide a set of representative inputs, a set of specified configuration parameters, and a range of values for each parameter. Then it executes all combinations of the representative inputs and configuration parameters. This exhaustive approach may consume a huge amount of time when the configuration parameters are too many or within a large range, which limits its assurance on robustness. Sidiroglou et al. [14] propose to perform criticality testing to filter out the loops that result in critical failures. It tests loops in turn with different perforation rates to see whether failures occur. The problem here is that only the local effect of one loop at a time is considered, thus providing no guarantee for the global effect when loops with different perforated rates are combined. Pekhimenko et al. [12] identify the cache misses of an application and maps the misses to code fragments. The quality of the training data is of vital importance to its success in avoiding critical failures at run-time. In spite of these drawbacks, all reported empirical results are very good.

When the accuracy tradeoff space is large, owing to the limited number of attempts made, the probability of identifying points of critical failures is low irrespective of the approaches. Let us consider equation (2). If the application A is a sequential and deterministic program, then the output of the application is solely determined by I and X. However, if the application has internal concurrency or non-deterministic behavior, merely comparing the accuracy in the application outputs may be inadequate, while comparing the program states may incur severe run-time overhead. Methods to adapt existing exhaustive testing strategy or framework training as required by (2) still require further research.

2) *Performance issue: soft failure*

Soft failures usually result in tolerable, though often not desirable, deviations from the original output, manifested in two aspects: the output accuracy is beyond the desired bound, or the application performance is not fully exploited. To avoid soft failures, the two sources of failures mentioned in the second paragraph of Section IV.*B* should be tackled.

The first source of failure is the projection of primitive accuracy tradeoff space through training. A more precise profile function reduces the output deviation. This can be achieved by utilizing more training data at the expense of higher data preparation and training overhead. Another method is to perform better profile modeling from training data. This will be discussed in the next subsection.

The second source of failure is due to the non-optimality of search. Exhaustive search is common in accuracy tradeoff frameworks [6][14], but due to its high cost, greedy search is usually provided as an alternative [10][12][13]. Pekhimenko et al. [12] use genetic algorithms to search through the space. When the initial tradeoff space is too large to be effectively searched, the size of accuracy tradeoff space can be managed by searching within a selected subspace. A large accuracy tradeoff space has a higher chance of retaining better outputs with high accuracy and good performance, but searching in a larger space is more costly, and an inadequate search may produce even less desirable results than searching in a smaller space. Limited by the tradeoff approaches and design of systems, however, existing work often constructs relatively small tradeoff spaces. When the accuracy tradeoff space has the potential to be large, we should well balance its size. A typical approach to achieve the balance is by filtering out the less promising regions in the tradeoff space. The criticality testing done in [14] attempts to filter out critical failure points, which also serves the objective of reducing the scale of the tradeoff space.

Other than directly tackling the sources of uncertainty, run-time feedback loop, as to be discussed in Section IV.*D*, is also a very important approach to minimize soft failures.

C. *Profiling issue: handling uncertainty*

As mentioned in Section III.*B*, profiling is to generate an estimation function in equation (6) that utilizes training data to handle the uncertainty caused by dimension projection. However, a profile may fail if the actual results deviate too much from the original results estimated by the profile function. At least three factors contribute to the failures of profiling: insufficient amount of training, unrepresentative training data compared with real ones, and too much uncertainty introduced by projection. In many cases, the first two factors can be handled by programmers, whereas the last factor is critical and heavily related to the characteristics of the application. Frameworks should directly address this uncertainty and determine whether their projections work. This problem has not been considered by existing work.

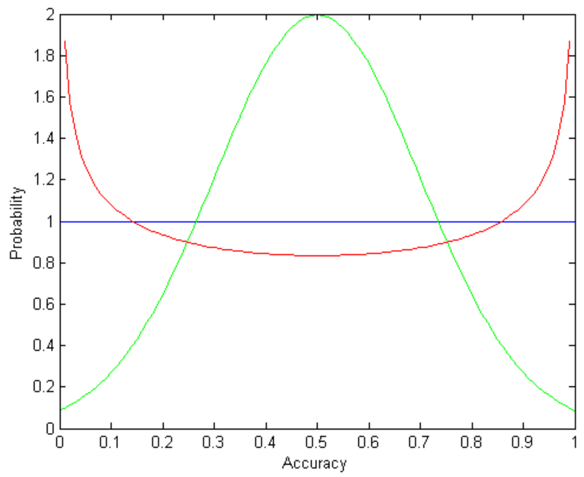

Figure 4. Normal distribution (green line), uniform distribution (blue line) and beta distribution (red line) of resulting accuracy

In many existing studies, given an *X* in equation (6), the mean value of every training input variable is used by the framework to estimate the output accuracy in the application. Chippa et al. [5] argue that static estimation of the output through training profile leads to error overshoot or only partially exploited performance gain. They point out that the degree of resilience may heavily depend on the input data.

Modeling the uncertainty by probability, Fig. 4 shows three possible distributions of the resulting accuracy of a large amount of training data with respect to the same configuration *X*. If the training results form a normal distribution as shown by the green line, the mean value is reasonably representative of the range of values. However, if the results are uniformly distributed as shown by the blue line, the mean value is a poor estimate of the other values. When the distribution is bipolar, as shown by the red line, the actual output may fluctuate fiercely, often showing great departure from expectation if estimated by the mean.

Some service requests may require a stringent accuracy guarantee and other requests can accept a more aggressively compromised accuracy to acquire sufficient performance gain. To meet diverse requirements, the uncertainty can be incorporated into the search criteria or the search process itself. Our current idea is that if the uncertainty is modeled by a probability distribution using training data as samples, a distribution of outputs can be constructed for each specified *X*. Then a confidence interval can be computed to provide a statistical guarantee of the accuracy levels.

D. Run-time feedback control

As mentioned in subsection IV.*B*, soft failure is due both to the nature of partial information induced by projection and also to the non-optimality of search, which cannot be fully solved before run-time, performing recalibration and adjustment via feedback control loops at run-time is promising. Chippa et al. [5] conduct an experiment in the context of a variable accuracy framework on support vector machine. Their results, reproduced graphically in Fig. 5, show that static configurations generated by equation (8) may greatly violate the target accuracy bound while sometimes leave performance gain unexploited.

Feedback control attempts to dynamically adjust the system behavior during the process of approaching the specified goal. It requires continuous monitoring of the process. However, the goal of A-Model is to achieve optimal output for one input, namely the particular service request. Also, the process is atomic in the sense that it cannot be monitored and adjusted continuously.

To incorporate feedback control, we need to modify the goal of A-Model from minimizing the failures in a single execution to minimizing the failures in a set of executions within a given period of time. Both *Green* [2] and *PowerDial* [6] specify this period as a continuous range of time with fixed accuracy requirement so that information of previous executions can be utilized to adjust subsequent ones. In subsection IV.*B*, the two processes which introduce failures are identified, namely, building profile and searching. Run-time feedback control can help minimize failures by refining these two processes at run-time

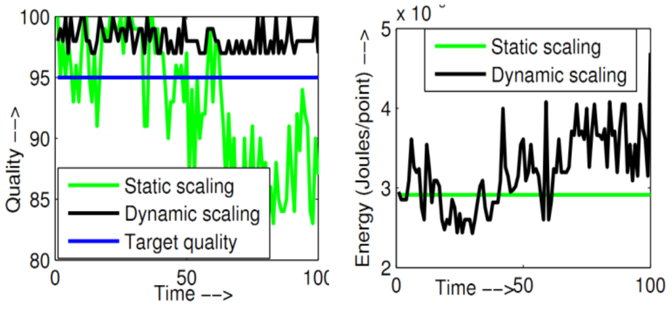

Figure 5. Variation in quality and energy with static and dynamic scaling (reproduced from [5]

Green [2] utilizes a refinement process to regressively revise the values of the configuration variables *X* in goal (7) in correspondence to the prior output. At the beginning, *Green* tunes its application based on the profile generated at the training phase. At run-time, when *Green* detects degradation in accuracy below the specified bound, it performs adaptation with the aim of improving the accuracy level until the bound is again satisfied. The adaptation strategy adopted in *Green* [2] is as follows. First, it orders all the variable nodes by their QoS loss or performance gain sensitivity. Then it sequentially picks the top nodes one by one and increases their accuracy until the most precise state of these nodes is reached or the accuracy bound is satisfied. This strategy was reported to converge quickly [2]. However, *Green* does not reduce the accuracy level when it is above the bound. Thus, the accuracy may be raised to and maintained at a high level for a long period of time. Thus, *Green* provides relatively strict guarantee on its accuracy. Unlike *Green*, *PowerDial* [6] focuses on the measurement, monitoring and feedback control of system performance rather than accuracy. When the performance is higher or lower than the expected level, *PowerDial* performs slowdown or speedup adaptation, respectively, to restore its satisfaction of the performance requirement. Adaptation is achieved based on the calibration profile obtained at the training phase.

Green [2] and *PowerDial* [6] only refine the search process since their profiles are static at run-time. In *Green*, the training profile mainly serves to facilitate fast convergence of system behavior. In *PowerDial*, after the initialization of dynamic knobs, the profile is only used as a measure of performance to determine speedup or slowdown. However, convergence takes time even if it is fast. If the context of the framework is volatile (that is, the accuracy requirement fluctuates frequently), the feedback control on search refinement may fail to achieve accuracy or performance convergence in time. On the other hand, when handling the same accuracy or performance requirement in another time period, the frameworks have to repeat the same feedback control. An idea to address this problem is to refine the profile with data at run-time to improve the accuracy of the profile. It is like active learning in machine learning, taking the run-time data as new training data to refine the profile.

E. *Specification of accuracy metrics*

Accuracy metrics are subjective, and how to define them poses a critical challenge in designing accuracy tradeoff frameworks. It is also hard to determine the scale of accuracy. The concrete definition of accuracy depends on the specific application tuned by the framework. So, to support a wide variety of applications, the framework has to provide a generic interface and specification for accuracy metrics.

The most intuitive approach to specify accuracy metrics is by comparing the approximate result with the precise one. The precise result is naturally a good baseline to evaluate the accuracy and performance of its approximate version. Many research studies [2][6][12][14] have applied this idea in their frameworks. They infer some numeric variables from the output and calculate the weighted difference from the precise output. This approach is direct, relatively objective and intuitively applicable to many applications. However, to perform the comparison, obtaining the precise output is mandatory. It takes a lot of overhead to compute both the approximated and precise outputs. This can be acceptable at the training phase, but is challenging to do at run-time.

Green [2] performs run-time measurement of accuracy at every several time instances. It assumes that the application results during feedback control are continuous and use the result of these points as samples to extrapolate to the overall condition. This assumption limits its scope of applications, because it cannot handle highly fluctuating accuracy-based requirement adequately. Despite sharing the same scheme of accuracy metrics, *PowerDial* [6] adopts a different approach. It gives up accuracy monitoring at run-time, but constantly watches for the performance-related conditions. It allows users to specify a minimum accuracy threshold before profiling. At run-time, it focuses on maintaining the performance at a specified level with no guarantee of maintaining the accuracy threshold. This approach may not behave well for accuracy stringent applications.

One other approach is to design a low overhead accuracy metric without the need for computing the precise output. With a low overhead accuracy metric, run-time feedback control is more accessible to this class of techniques.

V. Conclusion

In this paper, we have presented an idealized model (I-Model) and an approximation model (A-Model) of accuracy tradeoff frameworks. Each model describes the process of constructing accuracy tradeoff space to provide different accuracy levels and for the dynamic tuning of the applications to satisfy users' accuracy requirement. I-Model pinpoints the key aspect of approximation, while A-Model highlights the areas of differences and similarity among existing work. We have also revisited existing work and discussed selected dimensions which play important roles in achieving the optimal results. These dimensions are accuracy tradeoff approaches, issues of robustness, performance and profiling, run-time feedback control, and specification of accuracy metrics. Through comparing the work in the literature, we have presented challenges and opportunities for future research on several points, including the balance between the size of tradeoff space and the extent of cost-effective search, the modeling and handling of uncertainty due to dimension reduction, as well as the importance of feedback and development of low-overhead accuracy metric.

Acknolwedgement

This work is supported in part by the General Research Fund of the Research Grant Council of Hong Kong (project no. 111410). All correspondences should be sent to Dr. W.K. Chan.

References

[1] J. Ansel, Y. Wong, C. Chan, and M. Olszewski, "Language and compiler support for auto-tuning variable-accuracy algorithms," in Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 2011, pp. 85–96.

[2] W. Baek and T. M. Chilimbi, "Green: a framework for supporting energy-conscious programming using controlled approximation," ACM SIGPLAN Notices, vol. 45, no. 6, pp. 198–209, 2010.

[3] L. Barroso, "The case for energy-proportional computing," Computer, vol. 40, no. 12, pp. 33–37, 2007.

[4] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). USENIX Association, Berkeley, 2010, pp. 21–21.

[5] V. Chippa, A. Raghunathan, K. Roy and S. Chakradhar, "Dynamic effort scaling: Managing the quality-efficiency tradeoff," in Proceedings of Design Automation Conference (DAC), 2011, 48th ACM/EDAC/IEEE, pp. 603–608.

[6] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011, pp. 199–212.

[7] E. Y. Y. Kan, W. K. Chan, T. H. Tse, "EClass: An execution classification approach to improving the energy-efficiency of software via machine learning," Journal of Systems and Software, vol. 85, no. 4, pp. 960–973, 2012.

[8] J. G. Koomey. "Growth in Data center electricity use 2005 to 2010," CA: Analytics Press, Oakland, 2011.

[9] J. G. Koomey, M. Sanchez, H. Wong, and S. Berard, "Implications of historical trends in the electrical efficiency of computing," IEEE Annals of the History of Computing, vol. 33, no. 3, pp. 46–54, 2011.

[10] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011, pp. 213–224.

[11] R. Parthasarathy, "Recipe for efficiency: Principles of computing," Communications of the ACM, vol. 53, no. 4, pp. 60–67, 2010.

[12] G. Pekhimenko, D. Koutra and K. Qian, "Approximate computing: Application analysis and hardware design," unpublished.

[13] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, 2011, pp. 164–174.

[14] S. Sidiroglou, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 2011, pp. 124–134.