Analysis and Characterization of Inherent Application Resilience for Approximate Computing*

Vinay K. Chippa†, Srimat T. Chakradhar‡, Kaushik Roy† and Anand Raghunathan†
† School of Electrical and Computer Engineering, Purdue University
‡ Systems Architecture Department, NEC Laboratories America
†{vchipp,kaushik,raghunathan}@purdue.edu, ‡chak@nec-labs.com

ABSTRACT

Approximate computing is an emerging design paradigm that enables highly efficient hardware and software implementations by exploiting the inherent resilience of applications to in-exactness in their computations. Previous work in this area has demonstrated the potential for significant energy and performance improvements, but largely consists of ad hoc techniques that have been applied to a small number of applications. Taking approximate computing closer to mainstream adoption requires (i) a deeper understanding of inherent application resilience across a broader range of applications (ii) tools that can quantitatively establish the inherent resilience of an application, and (iii) methods to quickly assess the potential of various approximate computing techniques for a given application. We make two key contributions in this direction. Our primary contribution is the analysis and characterization of inherent application resilience present in a suite of 12 widely used applications from the domains of recognition, data mining, and search. Based on this analysis, we present several new insights into the nature of resilience and its relationship to various key application characteristics. To facilitate our analysis, we propose a systematic framework for Application Resilience Characterization (ARC) that (a) partitions an application into resilient and sensitive parts and (b) characterizes the resilient parts using approximation models that abstract a wide range of approximate computing techniques. We believe that the key insights that we present can help shape further research in the area of approximate computing, while automatic resilience characterization frameworks such as ARC can greatly aid designers in the adoption approximate computing.

Categories and Subject Descriptors

B.7.0 [**INTEGRATED CIRCUITS**]: General

General Terms

Algorithms, Design

Keywords

Inherent Application Resilience, Approximate Computing, Resilience Characterization

1. INTRODUCTION

Inherent application resilience is the property of an application to produce acceptable outputs despite some of its underlying computations being incorrect or approximate. It is prevalent in a broad spectrum of applications such as digital signal processing, image, audio, and video processing, graphics, wireless communications, web search, and data analytics. Emerging application domains such as Recognition, Mining and Synthesis (RMS) [1], which are expected to drive future computing platforms, also exhibit this property in abundance. The inherent resilience of these applications can be attributed to several factors: (i) significant redundancy is present in large, real-world data sets that they process, (ii) they employ computation patterns (such as statistical aggregation and iterative refinement) that intrinsically attenuate or correct errors due to approximations, and (iii) a range of outputs are equivalent (*i.e.*, no unique golden output exists), or small deviations in the output cannot be perceived by users.

For inherently resilient applications, functionality is defined on a continuous scale of *output quality*. Therefore, applications should produce outputs of acceptable quality rather than a unique "correct" output. Approximate Computing is a new design approach that leverages inherent resilience through optimizations that trade off output quality for improved performance, energy efficiency or other metrics. Effectively, approximate computing techniques relax the traditional requirement of exact (numerical or Boolean) equivalence between the specification and implementation.

Several previous efforts have explored approximate computing in software [8,9,15] and hardware [2-8,18,19], with promising results. Software techniques typically improve performance by skipping computations or reducing the use of costly operations such as inter-thread synchronization, whereas hardware techniques modify the design at various levels of abstraction to introduce tradeoffs between output quality and efficiency. These efforts have established the significant potential of approximate computing, and there is increasing interest in its use with the growth in inherently resilient applications. However, several challenges still need to be addressed before approximate computing can move from its initial stages of exploration to broader adoption.

First, the property of inherent application resilience and the various application characteristics that contribute to it need to be understood comprehensively across a broader spectrum of applications. Second, designers require tools that quantitatively evaluate the resilience of a given application, and identify the parts of the application that are amenable to approximate computing. Finally, there is a need for a systematic methodology that can help designers to quickly evaluate various approximate computing techniques for a given application, or a given technique across a wide range of applications.

In this work, we make two key contributions to broaden the scope and applicability of approximate computing. The first contribution of our work is the analysis and characterization of inherent application resilience present in a suite of 12 widely used recognition, mining and search applications. We demonstrate the high degree of resilience existing in these applications that emphasizes the scope and potential of approximate computing. We present several new insights into the nature of application resilience and its relationship to various application characteristics. These insights serve as guidelines for future research in the area of approximate computing.

Our second contribution is an Application Resilience Characterization (ARC) framework that (i) partitions a given appli-

*This material is based upon work supported in part by the National Science Foundation under Grant No. 1018621

DAC'13, May 29 - June 07, 2013, Austin, TX, USA.

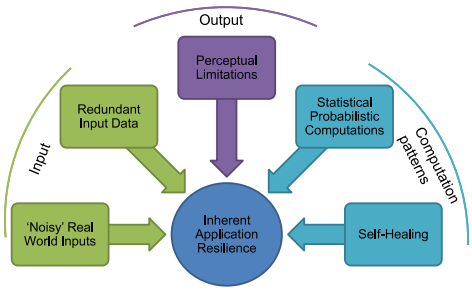


Figure 1: Various sources of inherent application resilience

cation into potentially resilient parts, which may be subject to approximate computing, and sensitive parts, which may not, and (ii) characterizes the potentially resilient parts in greater detail to evaluate the applicability of various approximate computing techniques. For this purpose, we propose the use of "approximation models" that efficiently abstract a wide range of approximate computing techniques to enable their quick evaluation. The proposed ARC methodology is realized using the `valgrind` dynamic binary instrumentation framework. We believe that tools such as the ARC framework will greatly aid designers in adopting approximate computing as an additional avenue for design optimization.

The rest of the paper is organized as follows: We present a qualitative analysis of application resilience in Section 2. We describe related efforts and place our contribution in their context in Section 3. The ARC framework is described in Section 4 and its implementation is detailed in Section 5. Section 6 describes various insights derived from the application of the ARC framework to our suite of benchmarks.

2. INHERENT APPLICATION RESILIENCE

Inherent Application Resilience is defined as the property of an application to produce acceptable outputs in spite of some of its underlying computations being incorrect (or approximate). The various sources that contribute to inherent application resilience are shown in Figure 1, and can be classified into three categories.

Inputs: These applications process input data that is noisy and redundant. The robustness to noise in the input data and the fact that similar data is processed several times (redundancy), often manifests as resilience to approximations.

Outputs: There does not exist a unique golden output, *i.e.*, a range of outputs are considered equally acceptable. Moreover, these applications generate output for consumption by humans, whose perceptual limitations imply that minor variations that cannot be discerned are acceptable.

Computation Patterns: These applications employ statistical computations that result in attenuation or cancellation of errors. Moreover, due to the iterative nature of computations in these applications, errors due to approximations in one iteration can potentially get healed/recovered in subsequent iterations.

3. RELATED WORK

Approximate computing has been applied to the design of hardware building blocks such as arithmetic units and entire datapaths, either through voltage overscaling [2, 3, 4] or through logic simplification [5, 6]. A systematic approach to apply approximate computing to hardware design at various levels of design abstraction was presented in [7, 8]. The application of approximate computing to programmable processors has been recently explored [9]. Approximate computing techniques in software [10, 11, 12] have been proposed to improve the performance and parallel scalability of applications on general-purpose computing platforms. These efforts evaluated specific approximate computing techniques for a limited set of applications, leaving open the question of whether approximate computing is applicable in a broader context.

Approximate computing is related to, but distinct from error-resilient computing, which seeks to lower the overheads

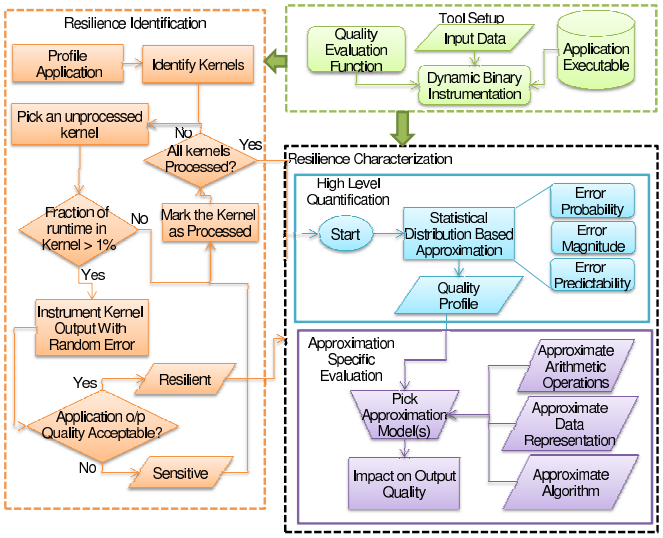


Figure 2: Overview of the ARC framework

of fault-tolerance from hardware defects or transient faults in hardware, by exploiting error masking at the application level, or by using low-cost algorithmic fault tolerance [13]. In approximate computing, "errors" are created intentionally, and are typically predictable (and can therefore be modeled). Automatic error injection frameworks that study the robustness of applications under faults have been extensively studied in the fault tolerance community [14]. A recent effort [15] employed an automatic error injection framework to study the resilience of signal processing applications using random bit flips. These techniques are inadequate for evaluating inherent application resilience, since they either (i) inject arbitrary faults (*e.g.*, random bit-flips), which often do not reflect the nature of approximate computing techniques, or (ii) consider any deviation from the golden numerical output of the program as unacceptable, oblivious to the continuum of output quality that we are concerned with.

The ARC framework proposed in this paper provides designers with a tool set to easily evaluate approximate computing for a new application. Additionally, the insights derived from our characterization of a broad range of applications provide guidelines for further efforts in the area of approximate computing.

4. APPLICATION RESILIENCE CHARACTERIZATION FRAMEWORK

In this section, we describe the proposed Application Resilience Characterization (ARC) framework that can be used to quantitatively evaluate the inherent resilience of applications. The framework, shown in Figure 2, consists of two major steps: (i) identification of potentially resilient computations and (ii) characterization of these computations through approximation models.

The inputs to the framework are the application program, a representative input data set and a user-defined quality evaluation function. The quality evaluation function processes the output of the application and evaluates the output quality as a numerical value. The quality evaluation function is application-specific and must be provided by the user; however the ARC framework itself is general. The outputs of the ARC framework include a list of the resilient computations in the application and the results of evaluating various approximation models on the resilient computations.

The overall approach taken in both steps of the ARC framework is to utilize dynamic binary instrumentation to introduce random errors or controlled approximations into specific computations as the application executes, and observe the resulting application behavior. We next describe the two steps of the ARC framework in detail.

4.1 Resilience Identification

Even the most resilient applications contain both resilient and sensitive computations. Approximate computing techniques should be targeted towards resilient computations while avoiding the sensitive ones. The first step of the ARC framework identifies potentially resilient computations by using Dynamic Binary Instrumentation (DBI) to inject errors into the results of computations as the application executes.

Although ARC uses software implementations of applications on a general-purpose platform (due to their widespread availability), our intent is really to evaluate the resilience of the algorithmic computations rather than all instructions in the software implementation. Therefore, we first partition the instructions in the program into computation kernels as follows. We consider innermost loops that account for over a specified fraction of the program's execution time (we used 1% in our experiments) as atomic kernels. As the program executes over the provided input data set, we add random errors to the program variables that are modified in a kernel and used outside it, *i.e.*, the kernel's outputs. If the application crashes or hangs, or produces an output that does not meet a relaxed output quality criterion, we mark the kernel as sensitive; otherwise, it is marked as potentially resilient. We use unconstrained random errors and a relaxed output quality criterion since the objective of this step is only to identify *potentially* resilient kernels; in the second step of the ARC framework, we use approximation models and the user-provided quality evaluation function to further evaluate and characterize these kernels.

For completeness, each instruction that lies outside the processed loops is considered as a separate kernel. However, in our experiments, virtually all resilient computations were found to be kernels generated from loops.

4.2 Resilience Characterization

Once potentially resilient kernels are identified in the first step of the ARC framework, the next step characterizes their resilience to provide insights into the applicability of various approximate computing techniques. The resilience characterization step uses the same strategy as the identification step *i.e.*, execute the application under DBI on the provided input data, inject errors in the kernels, and evaluate application behavior. However, there are two key differences. First, the errors introduced in the kernels are derived from approximation models that model the effects of various approximate computing techniques. The key objective of the approximation model is to (i) quantify the resilience using generic attributes of the approximations such as error probability, magnitude and predictability of errors introduced, and (ii) evaluate the impact of a single or a class of approximate computing techniques on the application output quality. Second, we use the quality evaluation function provided by the user. For a given approximation model, a quality profile is generated that characterizes the application output quality as a function of the model's parameters.

The approximation models used for resilience characterization are described in more detail in the following subsections.

4.2.1 High-level Approximation Models

Any approximate computing technique can be thought of as introducing errors in the computations that are being approximated. However, these errors are usually constrained by design such that the application output quality is not drastically impacted. To quantify an application's resilience at a high level (*i.e.*, independent of any specific approximate computing technique), we propose a *statistical approximation model*, in which the errors produced due to approximate computing techniques are modeled using a statistical distribution (Figure 3). The statistical approximation model is parameterized by three high level parameters: error probability, error magnitude and error predictability. Error probability determines the rate at which errors are produced by the approximation, and is denoted by the area under the statistical distribution (not including the error-free case). The error magnitude and error predictability constrain the numerical value of the error and correspond to the mean and variance of the error distribution. The ARC framework employs this model to generate a quality profile of the application as a function of these parameters. This model is very useful in the early stages of the design cycle, as it gives insights into the resilience of the application and helps narrow down design choices without the significant effort needed to implement various approximate computing techniques.

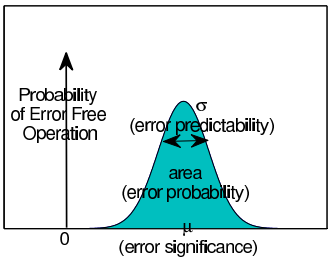


Figure 3: Statistical approximation model

4.2.2 Technique-specific Approximation Models

The statistical approximation model proposed to quantify resilience may not adequately reflect a specific approximate computing scheme. Therefore, we propose three approximation models that abstract important classes of approximate computing techniques.

Approximation of Arithmetic Operations

Many approximate computing techniques are applied to arithmetic units such as adders [17], multipliers *etc.* We propose the bit error profile model to represent the effect of these approximations. A bit error profile specifies the probability that each bit in the output of an arithmetic operation has an error. During resilience characterization with this model, the outputs of arithmetic operations in resilient kernels are modified in accordance with the chosen bit error profiles. For example, the bit error profile of an approximate adder proposed in [16] is presented in Figure 4. A bit error profile model may optionally specify conditional error probabilities based on the values of the operation's inputs.

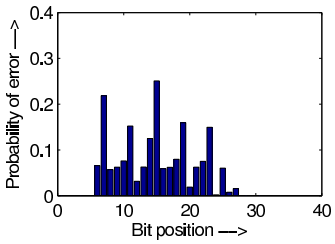


Figure 4: Bit error profile for an approximate adder [16]

Approximate data representation

Some approximate computing techniques exploit application resilience by employing approximate data representations. Bit truncation is one such commonly employed technique, where approximations are introduced by reducing the data width used to represent variables in hardware or software. Approximate data representations may also utilize different number systems such as logarithmic and residue number systems. To evaluate such techniques, we transform the kernel's inputs into the chosen approximate data representation, perform all kernel computations in the chosen representation, and transform the output back to the original representation.

Algorithm level approximations

In this model, we consider approximate computing techniques that modify the algorithm being implemented at a coarser granularity *e.g.*, an iteration of a loop might get skipped. We model the impact of these techniques as computation skipping - the kernel's execution is skipped with a specified probability.

5. EXPERIMENTAL METHODOLOGY

In this section, we describe the experimental setup used to implement and evaluate the proposed ARC framework. The basic functionality of ARC framework is implemented using valgrind, a popular dynamic binary instrumentation framework. valgrind enables easy and efficient instrumentation of a program by first translating it into an "Intermediate Representation"(IR) that is processor independent. Any tool im-

Table 1: The applications and the data-sets used to evaluate the proposed resilience characterization methodology

Application	Algorithm	% Runtime in resilient kernels	dominant kernel (Contribution to runtime)
Document Search	Semantic Search Index	90	Dot Product Computation (86)
Image Search	Feature Extraction	78	Dot Product Computation (71)
Hand Written Digit Classification	Support Vector Machines (SVM): Testing	94	Dot Product Computation (89)
Hand Written Digit Model Generation	Support Vector Machines (SVM): Training	97	Dot Product Computation (93)
Eye Detection	Generalized Learning Vector Quantization (GLVQ): Testing	89	Distance Computation (83)
Eye Model Generation	Generalized Learning Vector Quantization (GLVQ): Training	96	Distance Computation (92)
Image Segmentation	K-means Clustering	74	Distance Computation(66)
Census Data Modeling	Neural Networks: Multi Layer Back Propagation	62	Matrix Vector Multiplication (42)
Census Data Classification	Neural Networks: Forward Propagation	79	Matrix Vector Multiplication (64)
Nutrition and Health Information Analysis	Logistic Regression	65	Dot Product Computation (48)
Digit Recognition	K-Nearest Neighbors	96	Distance Computation (92)
Online Data Clustering	Stream Cluster	77	Distance Computation (68)

plemented using valgrind is allowed to instrument this IR and the instrumented IR is then executed on the host machine. By implementing the ARC framework using valgrind, we are able to easily apply the framework to any application without the need to modify the application source code.

We apply the ARC framework on a benchmark suite consisting of 12 widely used recognition, data mining and search applications, along with representative input data. All the applications are annotated with appropriate quality evaluation functions that translate application outputs into a numerical measure of output quality. The applications and the underlying algorithms of the benchmark suite are presented in Table 1. A detailed explanation of the applications and datasets, and the results of resilience characterization are presented in the supplemental section.

6. RESULTS AND INSIGHTS

We characterized the resilience of the applications in the benchmark suite using the ARC framework and present the results in this section. In the first step of the framework, we identify the resilient kernels of the application. The results for this step are presented in Table 1, column 3. It can be seen that, across all the applications in the benchmark suite, the fraction of the application's run-time that is spent in resilient kernels ranges from 67% to 96%. This demonstrates the high degree of inherent resilience present in these applications and underscores the potential for approximate computing. In most of these applications, there exists a single compute kernel that dominates the execution time. We present the dominant kernels for the benchmark suites in column 4 of the Table 1, and their contribution to program execution time.

On average, these applications spend 83% of their run-time in resilient kernels, out of which 74% belongs to the dominant kernel. Therefore, the bulk of the resilience can be exploited by focusing approximate computing design efforts on the dominant kernel. In order to apply suitable approximate computing techniques, it is important to understand application resilience in greater detail. While the raw data (results of resilience characterization) are presented in the supplemental section, we devote the remainder of this section to presenting several insights that we derived from our experiments.

6.1 Granularity of Approximation Matters

The efficiency of approximate computing techniques greatly depends on how the application is realized (dedicated hardware *vs.* software on a programmable processor). For example, consider an application with vector dot product as the dominant kernel. The dot product kernel may be resilient in the sense that controlled approximations to its result lead to acceptable program outputs. A hardware module that implements dot products in an approximate manner [4] may therefore be utilized. However, realizing the dot product kernel on a general purpose processor introduces instructions for loop control and pointer arithmetic. Introducing approximations at the granularity of instructions in this software implementation may lead to a very different conclusion about application resilience.

To study this effect, we expanded all loop kernels that were identified by ARC into machine instructions and performed error injection at the instruction level. On an average, we observed that the scope of approximate computing reduces by a factor of 57% when approximations are applied to individual instructions within kernels. This observation is consistent with prior efforts that have characterized application resilience at the granularity of processor instructions [18, 19]. Therefore, it is important to evaluate inherent application resilience in the appropriate implementation context.

6.2 Fail Small or Fail Rare

Consider the quality profile shown in Figure 5 for the image search application. This quality profile was generated using the statistical approximation model, with varying values of error probability (error rate), error magnitude (error mean), and error predictability (error variance). The quality profile is presented in the form of a 3-D slice plot, where each slice depicts the output quality for a given error rate (x-axis) and each block in a slice represents the output quality for a specific error mean (y-axis) and variance (z-axis). The output quality is color coded such that white regions represent no degradation in the output quality and darker regions represent higher degradation. In the case of image search, the output quality of

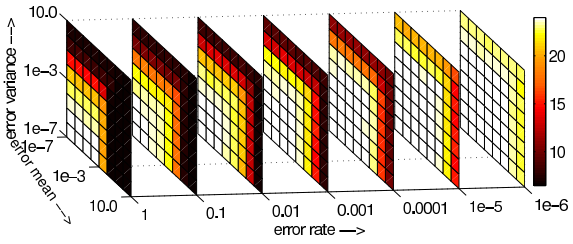


Figure 5: Resilience characterization of image search

is measured as the number of correct search results (out of the top 25) that match the reference golden output. It can be seen

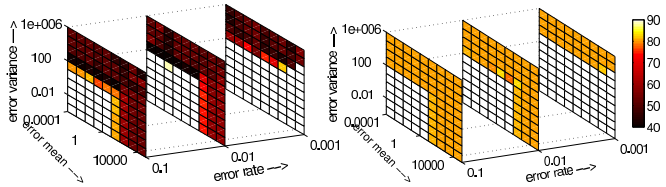
from the white regions in Figure 5 that the output quality of the application remains acceptable in the following two cases:

- Fail small: If the magnitude of the errors introduced by the approximations remains small, the application can tolerate even very frequent errors (left bottom regions in the left most slice).
- Fail rare: An approximation can introduce errors of arbitrarily high magnitude and still result in acceptable output quality as long as the errors are introduced very rarely (right most slice in the slice plot).

This insight of "fail small or fail rare" was observed across all applications in the benchmark suite. Therefore, designers should aim to develop approximation schemes that are constrained in either the rate or the magnitude of the approximations introduced.

6.3 Computation Patterns that Enhance Resilience

As described in Section 2, computation patterns are an important factor that contribute to application resilience. To quantitatively study this phenomenon, we consider the eye detection application (using the GLVQ algorithm) and compare



(a) GLVQ Testing (b) GLVQ Training

Figure 6: Impact of iterative computations on application resilience for eye detection application

the resilience of the training phase that builds the eye detection model with the testing phase that employs the trained model to detect eyes. Although the computation hot spot is the same in both parts, *i.e*, distance computation, the training phase employs an iterative convergence algorithm that keeps refining the model in each iteration, while the testing phase does not. The comparison of the quality profiles is presented in Figure 6. It can be seen that for the same point in the approximation space, the training phase that employs iterative computations is more resilient than the testing phase, demonstrating the contribution of iterative computations to resilience. Similar characteristics were observed in other recognition applications such as handwriting digit recognition (Support Vector Machines), census data analysis (Neural Networks) that contain training and testing phases. The impact of computation

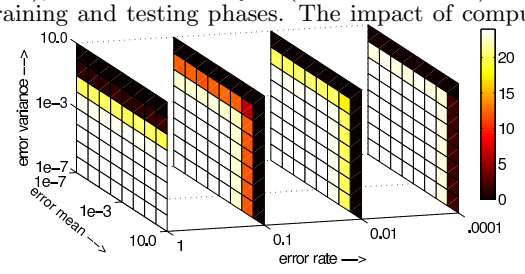


Figure 7: Impact of relative computations on resilience (Document search application)

patterns on resilience is observed in a different form in the document search application (implemented using Semantic Search Indexing). In this application, the input search query and the text documents in the database are represented in terms of vectors and the score of each document is calculated by computing the dot product of its vector with the input query. In the resilience characterization of this application, we observed that if the variance of the errors introduced due to approximations is constrained to a small value and the errors are introduced in all the computations (rate = 1), then the magnitude of errors can be arbitrarily large without the application quality being impacted. This is because, if errors of similar magnitude are introduced in all the computations, the numerical order of the document scores remains unchanged. This can be seen in the bottom region of the first slice shown in Figure 7. The resilience in such applications can be exploited effectively by constraining the variability of errors introduced due to the approximations.

6.4 Scale of Data Matters

Resilience is a function of scale, since redundancy in the input data is one of the major contributors to resilience. In this section, we analyze the impact of the scale of input data on application resilience. We consider the training phase of hand-written digit recognition, and perform experiments to evaluate its resilience by varying the size of the input data. The results of this evaluation are presented in Figure 8. It can be seen that the resilience of the application improves, as denoted by the lower degradation in the output quality, as we increase the number of samples in the training data set. This result quantitatively demonstrates the impact of redundancy in input data on application resilience. It is essential that the design of approximate computing techniques and their evaluation be performed in the context of representative input data scales.

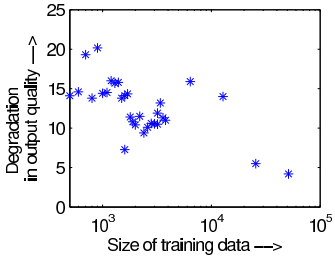
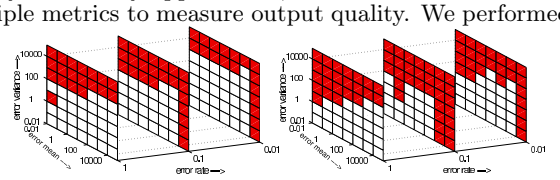


Figure 8: Impact of size of input data

6.5 Impact of Quality Metric

Inherent application resilience stems from the fact that application functionality is defined on a continuous scale of output quality. For many applications, there exists a choice between multiple metrics to measure output quality. We performed ex-



(a) (b)

Figure 9: Impact of quality metric: (a)mean centroid distance *vs.* (b)percentage of mis-clustered points

periments to study how the choice of quality metric impacts the degree of resilience. We consider the image segmentation application implemented using k-means clustering, and compare two commonly used quality metrics, namely (1) percent mis-clustered points and (2) mean centroid distance. The results are presented in Figure 9 in the form of a slice plot with white regions representing the acceptable quality levels. We normalized the ranges of these quality metrics such that degradation in output quality of less than 1% could be deemed acceptable. It can be seen that the quality profile corresponding to the mean centroid distance metric has a significantly larger white region (acceptable output quality), suggesting that the application is able to tolerate more aggressive approximation if mean centroid distance is the quality metric rather than percentage of mis-clustered points. These results demonstrate that, in addition to the computation patterns and input data, the context in which the application is used (encoded in the quality metric) significantly impacts resilience.

6.6 Application-aware Approximation

Approximate computing techniques can be implemented and applied in an application-aware or application-agnostic manner. In order to compare the effectiveness of these techniques, we consider the image segmentation application implemented using k-means clustering, and consider 2 types of approximations, one that skips computations randomly and the other that skips computations in an application-aware manner using a technique called early termination [20]. The comparison, presented in Figure 10, shows that the early termination technique results in much better output quality compared to random computation skipping for the same number of computations being affected. Therefore, it is important to understand the semantics of the application and apply approximation techniques accordingly to optimally exploit inherent resilience.

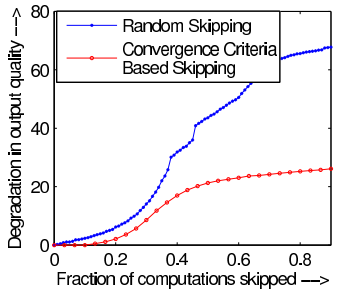


Figure 10: Impact of application semantics

6.7 Synergy between Approximation Techniques

Once the inherent resilience of an application is established, it can be exploited using several different approximate computing techniques. We performed experiments to analyze interactions between approximation techniques that are simultaneously applied to an application. We consider hand-written digit recognition using the SVM algorithm and simultaneously apply application aware computation skipping at the algorithm level and a voltage scaled implementation [4] of the dot product computation. It can be observed that these two approximation techniques can be employed in a synergistic manner, *i.e.* the approximations at the algorithm level can be applied on top of approximations in arithmetic operations without further degrading the output quality represented by vertical and horizontal parts of the contours. However, in some regions (curved portions of the contours), employing both techniques together results in worse output quality compared to a single technique. Therefore, designers should consider combining multiple approximate computing techniques to maximally exploit application resilience.

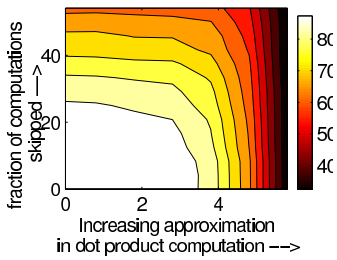


Figure 11: Synergy between algorithm level and arithmetic operation approximations

In summary, we have characterized the resilience of a broad set of applications using the proposed ARC framework and quantitatively established the high degree of resilience present in a broad range of applications. We presented key insights that quantitatively evaluate the relationship between resilience and various application characteristics such as computation patterns, input data, quality metric *etc.*

7. CONCLUSIONS

In this paper, we proposed an Application Resilience Characterization (ARC) framework for the analysis and characterization of resilience. We implemented the ARC framework using the dynamic binary instrumentation tool valgrind and applied it to a suite of 12 recognition, mining and search applications. We quantitatively established the high degree of resilience present in these applications, demonstrating the scope for approximate computing techniques. We performed experiments to explore the relationship between inherent resilience and various application characteristics such as computation patterns, scale of input data, quality metric *etc.* We believe that these key insights, along with the ARC framework, can greatly aid designers in the adoption of approximate computing.

8. REFERENCES

[1] P. Dubey. A Platform 2015 Workload Model Recognition, Mining and Synthesis Moves Computers to the Era of Tera. White paper, Intel Corp., 2005.

[2] Rajamohana Hegde and Naresh R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proc. Int. Symp. on Low Power Electronics and Design*, pages 30–35, 1999.

[3] Krishna V. Palem, Lakshmi N. Chakrapani, Zvi M. Kedem, Lingamneni Avinash, and Kirthi Krishna Muntimadugu. Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects. In *CASES*, pages 1–10, 2009.

[4] Debabrata Mohapatra et.al. Design of Voltage Scalable Metafunctions for Multimedia, Recoginition and Mining Applications. In *Proc. DATE*, pages 950–955, 2011.

[5] Vaibhav Gupta et.al. IMPACT: Imprecise Adders for Low-Power Approximate Computing. In *Proc. ISLPED*, pages 409–414, 2011.

[6] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *VLSI Design*, pages 346–351, 2011.

[7] V.K. Chippa, D. Mohapatra, A.Raghunathan, K. Roy, and S.T. Chakradhar. Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency. In *DAC'10*.

[8] Vinay K. Chippa, Anand Raghunathan, Kaushik Roy, and Srimat T. Chakradhar. Dynamic Effort Scaling : Managing the Quality Efficiency Tradeoff. In *Proceedings of the 48th Design Automation Conference (DAC'11)*, pages 603–608, San Diego, California, USA, 2011. ACM.

[9] Hadi Esmaeilzadeh et.al. Architecture Support for Disciplined Approximate Programming. *SIGARCH Comput. Archit. News*, 40(1):301–312, March 2012.

[10] S. T. Chakradhar and A. Raghunathan. Best-effort Computing: Re-thinking Parallel Software and Hardware. In *Proc. DAC*, pages 865–870, 2010.

[11] W. Baek and Trishul M. Chilimbi. Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *Proc. PLDI*, pages 198–209, 2010.

[12] Sidiroglou-Douskos et.al. Managing Performance vs. Accuracy Trade-offs with Loop Perforation. ESEC/FSE, pages 124–134, 2011.

[13] Larkhoon Leem et.al. ERSA: Error Resilient System Architecture for Probabilistic Applications. In *DATE*, pages 1560–1565, 2010.

[14] Mei-Chen Hsueh et.al. Fault Injection Techniques and Tools. *Computer*, 30(4):75–82, April 1997.

[15] Jason Cong and Karthik Gururaj. Assuring Application-Level Correctness Against Soft Errors. ICCAD, pages 150–157, 2011.

[16] S. Venkataramani et.al. SALSA: Systematic Logic Synthesis of Approximate Circuits. Proc. DAC, pages 796–801, 2012.

[17] R. Amirtharajah et.al. A Micro-Power Programmable DSP Using Approximate Signal Processing Based on Distributed Arithmetic. In *JSSC*, pages 337–347, 2004.

[18] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. Raise: Reliability-aware instruction scheduling for unreliable hardware. In *Proc. ASP-DAC*, pages 671–676, 2012.

[19] H. Duwe. Exploiting application level error resilience via deferred execution. Master's thesis, University of Illinois at Urbana Champaign, 2013.

[20] J. Meng et.al. Best-Effort Parallel Execution Framework for Recognition and Mining Applications. IPDPS, pages 1–12, 2009.

[21] Department of machine learning. *www.nec-labs.com/research*.

[22] Thorsten Joachims. Advances in Kernel Methods. chapter Making large-scale support vector machine learning practical, pages 169–184. MIT Press, 1999.

[23] Yann Lecun and Corinna Cortes. The MNIST Database of Handwritten Digits.

[24] D. Martin et.al. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, 2001.

[25] Richard M. Yoo, Anthony Romano, and Christos Kozyrakis. Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System.

[26] Cheng-Tao Chu et.al. Map-Reduce for Machine Learning on Multicore. In *Advances in Neural Information Processing Systems 19*, pages 281–288. 2007.

[27] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[28] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

SUPPLEMENTAL SECTION

A. BENCHMARK SUITE: OVERVIEW AND RESULTS

A.1 Document Search

- Algorithm: Semantic Search Index
- Source code: An industry scale implementation developed at NEC Labs, America [21]
- Data set: A subset of Wikipedia pages, America [21]
- Dimensionality: 100
- No of pages to search: 1863573
- Quality function: No of correct results in top 25 results
- Description: This application takes a text query as input and gives out the top 25 documents based on their similarity to the input query. The input query is first transformed into a vector of dimensionality 100. The application contains the vector representations of all the documents in the wiki sample. The dot product of the query vector with a document vector computes the similarity score between them. The documents are then sorted based on their similarity score and the top 25 documents are considered for our quality evaluation. The quality profile of this application is shown in Figure 7

A.2 Image Search

- Algorithm: Feature Extraction
- Source code: An industry scale implementation developed at NEC Labs, America [21]
- Data set: A database for 7700 pre-classified images taken from NEC Labs, America [21]
- Dimensionality: 128
- No of image categories: 765
- Quality function: No of correct results in top 25 results
- Description: This application takes a query image as input, analyzes its content and outputs image categories sorted based on their similarity to the query image. In this application, the image is first converted into a feature map that is then classified using an SVM classifier to determine the category the image is closest to. We consider the feature extraction step for resilience analysis. The quality profile of this application is presented in Figure 5

A.3 Handwritten Digit Recognition and Model Generation

- Algorithm: Support Vector Machines (Training and Testing)
- Source code: SVM light [22]
- Data set: MNIST database [23]
- Dimensionality: 784
- No of output classes: 10
- Training data size: 10,000
- Testing data size: 60,000
- Quality function: classification accuracy calculated in terms of percentage of correctly classified points
- Description: This application consists of two phases. The training phase generates the model of classification using a labeled training data. This phase of the application is formulated as a quadratic optimization problem and can be solved using off-the-shelf solvers [22]. The model generated from the training phase is used in testing to classify a new unlabeled data point. The quality profile of training and testing are shown in Figures 12 and 13 respectively.

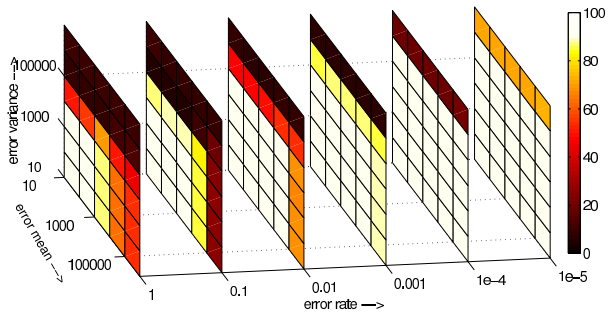


Figure 12: Hand written digit recognition: SVM Training

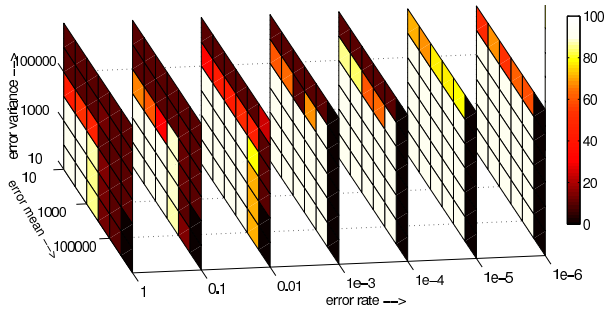


Figure 13: Hand written digit modeling: SVM Testing

A.4 Image Segmentation

- Algorithm: K-means Clustering
- Data set: Berkeley image segmentation dataset [24]
- Source code: An open source implementation taken from PHOENIX [25]
- Dimensionality: 3 (red, green and blue pixel values)
- No of clusters: 4
- Data size: 154401
- Quality function: Percentage of mis-clustered points and mean centroid distance
- Description: This application takes an image as input and segments it based on the Red, Green and Blue components of the pixels. The algorithm start off with random cluster centroids and assigns the pixels to centroids they are closest to based on Euclidean distance. The centroids are then recalculated as the mean of the points assigned to them. These two steps are performed iteratively until a convergence criterion is satisfied. The quality profile of this application for two quality metrics, mean centroid distance and % mis-clustered points is presented in 14 and 15.

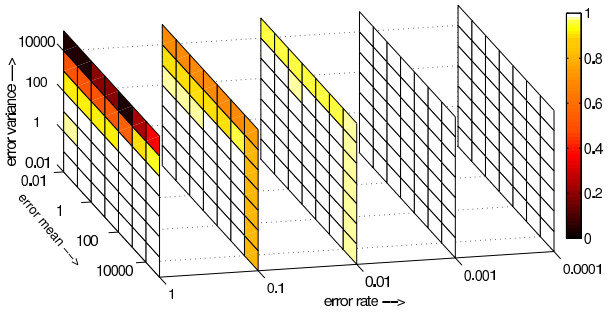


Figure 14: Image segmentation with mean centroid as quality metric: K-means Clustering

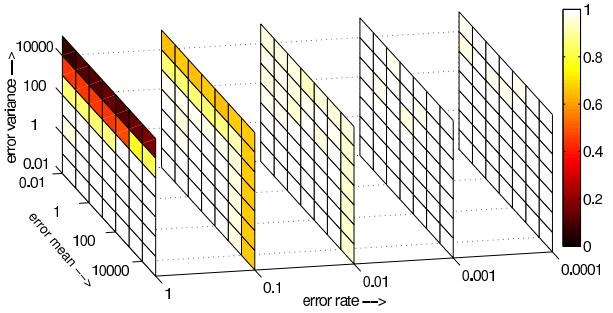


Figure 15: Image segmentation with percentage of misclustered points as quality metric: K-means Clustering

A.5 Eye Detection (Model Generation and Classification)

- Algorithm: Generalized learning vector quantization (Training and Testing)
- Source code: An industry scale implementation developed at NEC Labs, America [21]
- Data set: Set of eye and non-eye images from NEC Labs, America [21]
- Dimensionality: 500
- No of output classes: 2
- Training data size: 6000
- Testing data size: 1467
- Quality function: Classification accuracy represented in terms of percentage of correctly classified points.
- Description: This application consists of training and testing phases. In the training phase, the classification model is generated using a set of reference vectors which are updated using a labeled training data set. For each training vector, the closest reference vector from the same category as the training vector is modified such that it moves closer to the training vector. The closest reference vectors from the other categories are updated such that they move away from the training vector. The testing phase takes an unlabeled data as input and computes its distance from all the reference vectors. The input data is then labeled with the category of the reference vector it is closest to. The quality profiles of the training and testing phases of this application are depicted in Figures 6b and 6a

A.6 Census Data Classification (Model Generation and Classification)

- Algorithm: Neural Networks (Training and Testing)
- Source code: Single thread version from Mapreduce benchmark suite [26]
- Data set: UCI census database [27]
- Dimensionality: 14
- No of output classes: 2
- training data size: 32,560
- testing data size: 16,282
- Quality function: Classification accuracy represented in terms of percentage of correctly classified points.
- Description: This application employs Neural Networks to estimate the salary of a person based on the information in the census database. A 3-layer neural network is trained using back propagation algorithm to determine the weights of connections in neural network. In the testing phase, the feed forward algorithm is employed on the trained network to determine a person's salary. The quality profiles for the back propagation (training) and forward propagation (testing) parts of this application are presented in Figures 16 and 17

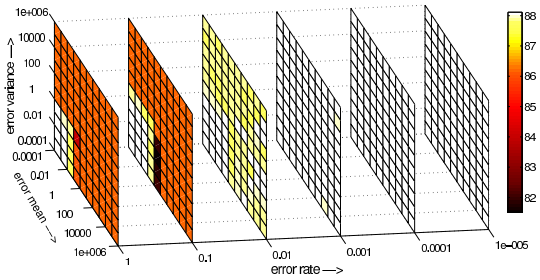


Figure 16: Census data modeling: Neural Networks Back Propagation

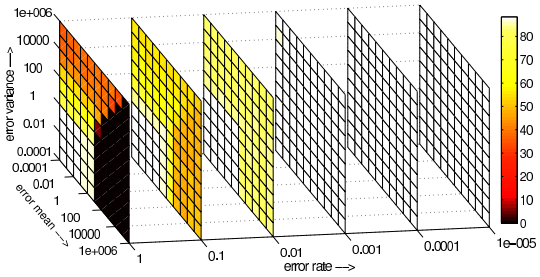


Figure 17: Census data analysis: Neural Networks Forward Propagation

A.7 Nutrition and Health Information Analysis

- Algorithm: Logistic Regression
- Source code: Single thread version from Mapreduce benchmark suite [26]
- Data set: National Health and Nutrition Examination Survey
- Data size: 17000
- Dimensionality: 15
- No of output classes: 2
- Quality function: Percentage classification accuracy
- Description: In this application, logistic regression is used to assess the likelihood of a disease or health condition as a function of risk factors. We use a quadratic optimization based implementation that generates a linear

model to minimize the error on the labeled training data. The dot product operation between the training data and the model is the dominant kernel computation in the quadratic optimization. The quality profile of this application for approximations in the dot product kernel is presented in Figure 18

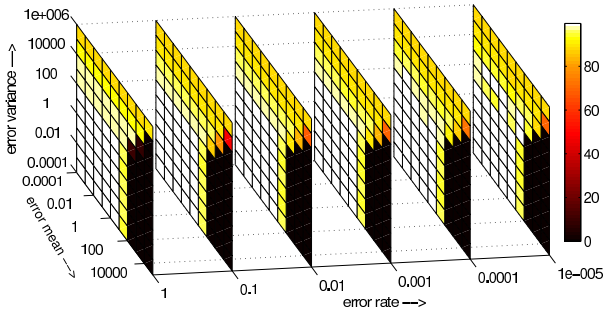


Figure 18: Health and nutritional information analysis: Logistic Regression

A.8 Digit Recognition

- Algorithm: K-Nearest Neighbors
- Source code: Self implementation
- Data set: UCI digit recognition database [27]
- Dimensionality: 64
- Training data size: 3823
- Testing data size: 1797
- No of output classes: 10
- Quality function: Percentage classification accuracy
- Description: This application uses the K-nearest neighbors algorithm to recognize digits from images. The algorithm computes the Euclidean distance between the test image and all training images. The training images are ranked based on the proximity to the test image, and the top K among them are identified. A majority vote among these nearest training images determines the classification. The quality profile, shown in Figure 19, illustrates the significant amount of resilience present in the distance computations.

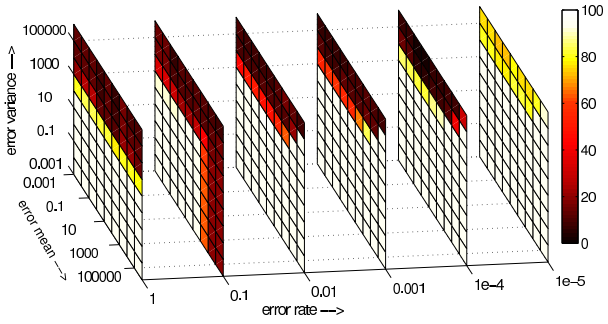


Figure 19: Digit recognition: K-Nearest Neighbors

A.9 Online Data Clustering

- Algorithm: Streamcluster
- Source code: Parsec benchmark suite [28]
- Data set: simmedium dataset provided with Parsec [28]
- Dimensionality: 64
- Data size: 8192
- No of output classes: Adaptive according to the input data
- Quality function: Mean centroid distance
- Description: Streamcluster finds the optimal number of clusters from a stream of input points where each input point is assigned to the closest cluster in terms of Euclidean distance from the cluster centroid. For each incoming set of points, the application employs heuristics [28] to determine the optimal number of clusters. The distance computation kernel is considered for approximation and the corresponding quality profile is presented in Figure 20.

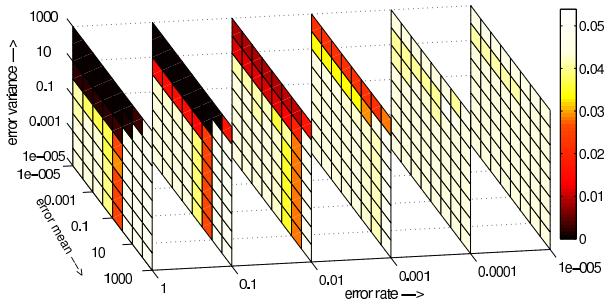


Figure 20: Online data clustering: Streamcluster