A Review Of Semantic Abstraction Techniques Are Used To Collect, Analyze, And Interpret Enormous Amounts Of Raw Data

Dr.Satish Kumar, Associate Professor, Department of Computer Science, Government College Narnaul, Distt. Mohindergarh, Haryana, India, s9467723723@gmail.com

ABSTRACT

For some applications, precision is crucial, yet for others, neither exact computation nor precise outcomes are necessary. For instance, a photo taken with an Visually, an 8 MP camera and a 10 MP camera are comparable. In fields like computer science speech recognition, graphics, searching, machine learning, and physical simulation, Perfect solutions may not be possible. The Approximate Computing (AxC) paradigm is gaining traction as a novel architecture for delivering better energy efficiency and performance solutions by sacrificing correctness. Researchers have proposed various AxC approaches to improve hardware and software performance. In this work, we have explored and proposed some software approximation techniques in the domain of program analysis and optimization. Program Analysis focuses on program optimization and the correctness of the program. This work presents a novel idea for transforming a program into an equivalent approximate program. The approximated version of the program can run in the non-approximated hardware and be more efficient in terms of computation time, resource utilization, memory usage, and power consumption. The first proposed method finds the original program's cold segments using the edge profiling method and applies approximation as these parts have less effect on the final result. The technique uses the distortion metric for verification of the output quality. The subsequent work uses the concept of randomized loop perforation and applies it to the existing preferential path profiling algorithm for k-iterations. This algorithm produces the compact number for the path identifier of the Approximately Most Executing (AME) paths, reducing the overhead of path profiling with an array instead of a hash table. Finally, we propose a software approach to function memoization. The objective is to improve computing efficiency by avoiding the actual execution of the function. The proposed method uses a rule to help in decision making, whether to search the look-up table or go for the actual computation. This decision-making model is implemented through Bloom Filter and Cantor's pairing function. The proposed model contains a bypass algorithm implemented through C++ code that identifies the trivial computations from the input argument of the candidate function. We conducted Several experiments using the AxBench and WCET suite benchmarks. We found that our result outperforms some of the methods proposed so far in terms of speedup and quality of results (QoR).

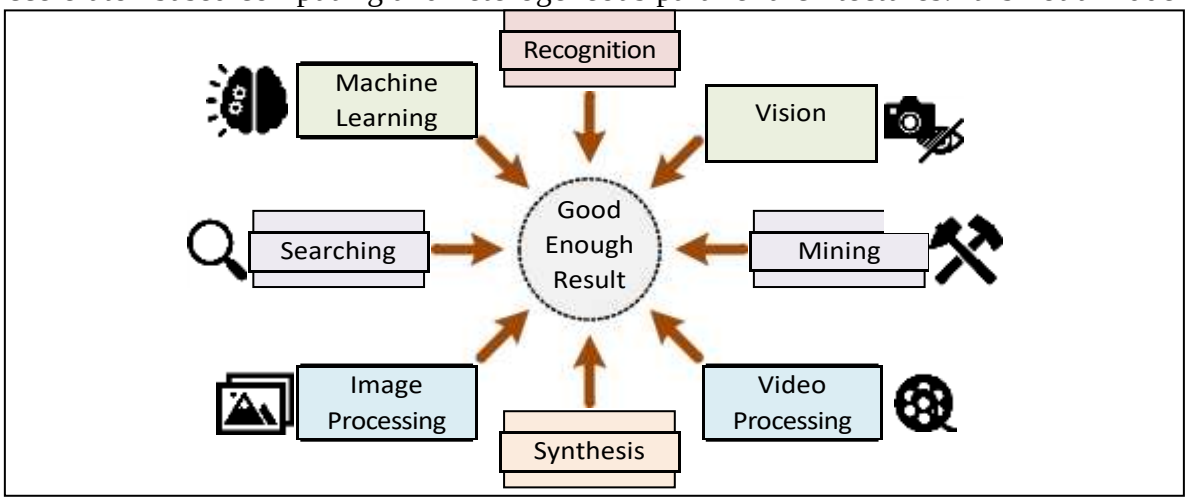
KEYWORD: Approximate Computing, Path Profiling, Loop Perforation, Approximate Path, Program Analysis, Function Memoization, Bypass Computation

INTRODUCTION

Since their inception, computers have evolved into little devices with a wide range of capabilities. Computer programs have developed to cover work from many different varied dimensions. Semantic abstraction techniques are used to collect, analyze, and interpret enormous amounts of raw data. For the most part, the cloud and data centers are in charge of keeping the information on the omnipresent and boundless World Wide Web current. The execution of workloads via data recognition and interpretation is a crucial function of mobile and handheld devices. Mobile devices are more likely to collect information from the real world and from the data that users feed to them. These gadgets are intelligent and aware of their context to carry out complex tasks. That they are future technology goes without saying.
some parts of the results are an outcome of poor computations by taking approximations and imprecise sightings into account. Similar error-tolerant characteristics can be seen in many other major application domains of the computing systems like graphics, image processing etc. Significant growth can still be expected for computing platforms in future and this demand is very unlikely to die with time. On the contrary, technology scaling doesn't receive any such benefits and is sure to diminish over time. As a result of which computing platform designers are facing a very serious problem due to the efficiency gap which keeps

Figure 1.1: Applications works with good enough results

widening(figure1.3). They wander off to innovate in unexplored areas to improve performance and energy efficiency of any specific computing platform. This exploration has led to the opening of new research areas like near-threshold computing, accelerator-based computing and heterogeneous parallel architectures. It is not a hidden



fact that application domains possessing prominent resilience are the primary reasons which contribute to a large part of the growth in computing workloads. These include domains like mining, audio/video processing, recognition, graphics, data analysis, search and mining. If an improvement along the dimension of computing efficiency is anticipated, then error resilience exploitation by design optimization could lead to the desired growth.

A leap in designing systems which are ”energy-efficient” and ”error-tolerant” has been taken by researchers and it is named as ”Approximate Computing”(AxC). This step will lead to opening up of new explore-worthy directions in computing. The main objective behind AxC is to achieve ‘good enough’ results along with reduced amount of energy consumption that is increase in saving of energy. This is also done by applications which don’t require exact computations for viable results. The reason being, higher energy losses is more overwhelming requirement than quality loss when a system is confirmed. AxC offers the perfect balance between performance and power consumption. This perfect harmony is created by loosening the knot of strict requirements placed on accuracy and precision. Energy efficient software design is the need of the hour. Applications facilitate the consumption of energy in the hardware. Energy efficient systems can be obtained through improved efficiency of both the software as well as the hardware. Program analysis techniques are the backbone of design for such software systems and hence, should be prioritized. Analysis, quantification and optimization techniques should be selected with respect to energy efficiency. The software available in the market can be analysed tofind out that they are designed to satisfy multiple criteria. Software engineering’s most prominent goal is to develop software which are functionally correct and produce acceptable results. However, there are other secondary criteria which the software need to satisfy to achieve user satisfaction. These criteria are properties like usability, responsiveness, performance, energy efficiency etc. These properties can be divided into two phases using common engineering practices. In the first phase, the developers are concerned about the correctness of the computation. Traditional computing takes the notion of correctness very strictly and hence follows precise computing. Traditional method of precise computing gives accurate results costing extra time and consuming more energy. This is going to cost the data centres dearly as it would protrude scalability issues and also increase IT infrastructure cost. Performance which is a secondary property is improved by the developers in the second phase called as the ‘Optimization’. Performance optimization can typically be done by finding out

subcomputations that are candidates for optimization and reimplementing it. This make the program run faster. This is not an easy task to achieve as it calls for lots of development effort. If the optimization candidates are prioritized appropriately, developers will get an insight on the most promising candidates. This will save a lot of efforts and time. Hence, optimization process requires careful analysis and selection of most profitable optimization candidates.

We aim at discussing the software techniques of "Approximate Computing" that can be applied to a program/software. AxC is a technique that can be used to make program/software efficient.

Approximate Computing

Approximate Computing (AxC) is a design approach of computing platforms such that they produce "good enough" results along with the minimization of energy requirement [31]. Before the exploration of AxC, traditionally the computing platform were designed with the goal for correctness. AxC not only aims at correctness but also ensures low energy consumption.

Approximation on different Layers	
Layers	Approximate Techniques
Program	Loop Perforation, Code Perforation, Memoization, ThreadFusion, Pattern Reduction,etc
Architecture	Approximate Storage, ISA Extension, Approximate Accelerator, etc
Circuit	Imprecise Logic, Voltage Overscaling, Analog Computation, Precision Scaling etc

Every application operates with two segments that is, Hardware part and Software part. In both these segments some parts are "error-sensitive" and some parts are "error-tolerant". The principle of AxC is only applicable to error-tolerant section of either hardware or software segment. Hence identification of the error-tolerant section of an application is very crucial activity in approximate computing framework. Many techniques are proposed by different researchers to identify the error-tolerant section. Some of the techniques are listed below.

1. Sensitivity Analysis.
2. Error injection and output monitoring.
3. Annotation of the application with approximate data types.

In sensitivity analysis an application is divided into several segments and sensitivity checking is done on each of them and then characterised as sensitive or tolerant.

In error-injection and output monitoring technique, errors are injected in an application, it is run on a simulator and the output is compared with the actual output. If the difference of the outputs is not acceptable then it is treated as non-tolerant. In another method dedicated approximate data types are used to differentiate between resilient and sensitive part in an application [115].

After identification of the resilient parts, the approximation techniques are applied to these parts and the quality of the final result is checked by running these through simulators. As simulated output and the actual output differs, the quality management techniques are applied for the final output for quality checking.

Monitoring of output quality

Many techniques are proposed by the researchers for checking the output quality. Some techniques check the intermediary computations. Some techniques uses simple error prediction models. Another technique is to use fuzzy-memoization where the previous input and output are checked to predict the current output for the similar input.

Scope and Application of AxC

As the computation demands high performance, now a days most applications run on clouds or on mobile devices. It is predicted that if the energy requirement is not minimized or the energy is not conserved properly then, by the end of 2040 the computing platforms
require more than the total world's energy production[16]. Several techniques like optimization in algorithms, adaptation of dynamic run-time, uses of hardware accelerators etc are used
for energy conservation. Another emerging technology is, the use of "Approximation". Approximate Computing plays a vital role in the process of energy gain by computing approximate results for some applications.
The scope of application of AxC can be classified into 4 categories[142].

1. Application with imprecise input types.
2. Applications with imprecise output.
3. Applications of the output that are computed using heuristics algorithm. Mainly these outputs are ambiguous in nature.
4. Applications that compute convergent outcomes produced by optimization or iterative methodologies.

The computations on the input data do not need to be highly precise in the first classification of application. Voice recognition, motion detection, and sensor data processing are examples of such applications.
The acceptable quality of the output in the second classification is primarily determined by human perception. Audio, picture, and video processing in multimedia applications are some examples.
The third category of application includes machine learning applications and big data analysis and mining that uses stochastic algorithms and produces inexact results.
The fourth category of the scope implies that the variation in the output depends on the precision of the computation.

Challenges in AxC

There are certain challenges associated with AxC. The following are some challenges that need to be taken care by the developers.
Restricted application area of AxC.

- Correctness Issue for the application of level of Approximation.
- Difficult to find a proper strategy for an application.
- Overhead and Scalability Issue.
- Close Monitoring of Quality of Result (QoR).

CONCLUSION

The thesis was inspired by the Approximate Computing Technique. The Approximate Computing paradigm is emerging as a new architecture to deliver better solutions in terms of energy gain, improved performance by trading off the accuracy of the result [138]. Approximate computing technique is driven by several error-resilient applications like multimedia, signal processing, machine learning, and robotics. Based on the input data and nature of computations, these applications do not demand exactness in the output. The energy efficient software design is the major concern in today's scenario. Program analysis and optimization techniques are the backbone of design for such a software system. In this work, we have tried to explore the software approximation technique in program analysis and optimization domain Now, we present the main contributions of the thesis and also discuss some possible future research directions. We summarize the contributions of the thesis as follows.

• We introduced the concept of Approximate Computing with the detail differentiation between Traditional Design Process(TDP) and Approximate Design Process(ADP) along with the Scope, Application and Research direction of Approximate Computing.

• We discussed various Approximation strategies for different computation stack and explored various software approximation strategies in program optimization and analysis.

• We proposed one metthod for transforming a program into an approximated program

REFERENCE

[1] Afraz, M., D. Saha, and A. Kanade (2015). P3: partitioned path profiling. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 485–495.
[2] Agosta, G., M. Bessi, E. Capra, and C. Francalanci (2012). Automatic memoizationfor energy efficiency in financial applications. Sustainable Computing: Informatics and Systems 2 (2), 105–115.
[3] Aho, A. V., R. Sethi, and J. D. Ullman (1986). Compilers, principles, techniques. Addison wesley 7 (8), 9.
[4] Aiken, A. (1999). Introduction to set constraint-based program analysis. Science of Computer Programming 35 (2-3), 79–111.
[5] Akturk, I., K. Khatamifard, and U. R. Karpuzcu (2015). On quantification of accuracy loss in approximate computing. In Workshop on duplicating, deconstructing and debunking (WDDD), Volume 15, pp. 28.
[6] Alaghi, A. and J. P. Hayes (2013). Survey of stochastic computing. ACM Transactions on Embedded computing systems (TECS) 12 (2s), 1–19.
[7] Alouani, I., H. Ahangari, O. Ozturk, and S. Niar (2017). A novel heterogeneous approximate multiplier for low power and high performance. IEEE Embedded Systems Letters 10 (2), 45–48.
[8] Alvarez, C., J. Corbal, and M. Valero (2005). Fuzzy memoization for floating-point multimedia applications. IEEE Transactions on Computers 54 (7), 922–927.
[9] Baek, W. and T. M. Chilimbi (2010). Green: A framework for supporting energy-conscious programming using controlled approximation. In Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 198–209.
[10] Ball, T. (1994). Efficiently counting program events with support for on-line queries. ACM Transactions on Programming Languages and Systems (TOPLAS) 16 (5), 1399–1410.
[11] Ball, T. (1999). The concept of dynamic analysis. In Software Engineering—ESEC/FSE'99, pp. 216–234. Springer.
[12] Ball, T. and J. R. Larus (1994). Optimally profiling and tracing programs. ACM Transactions on Programming Languages and Systems (TOPLAS) 16 (4), 1319–1360.
[13] Ball, T. and J. R. Larus (1996). Efficient path profiling. In Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29, pp. 46–57. IEEE.
[14] Ball, T., P. Mataga, and M. Sagiv (1998). Edge profiling versus path profiling: The showdown. In Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 134–148.
[15] Barua, H. B. and K. C. Mondal (2018). Green data mining using approximate computing: An experimental analysis with rule mining. In 2018 International Conference on Computing, Power and Communication Technologies (GUCON), pp. 115–120. IEEE.
[16] Barua, H. B. and K. C. Mondal (2019). Approximate computing: A survey of recent trends—bringing greenness to computing and communication. Journal of The Institution of Engineers (India): Series B 100 (6), 619–626.
[17] Bessi, M. (2014). A methodology to improve the energy efficiency of software.
[18] Betzel, F., K. Khatamifard, H. Suresh, D. J. Lilja, J. Sartori, and U. Karpuzcu (2018, January). Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems. ACM Comput. Surv. 51 (1).
[19] Bienia, C., S. Kumar, J. P. Singh, and K. Li (2008). The parsec benchmark suite: Characterization and architectural implications. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 72–81.