Towards More Precision in Approximate Computing *

Radha Venkatagiri Abdulrahman Mahmoud Sarita V. Adve

University of Illinois at Urbana-Champaign

{venktgr2,amahmou2,sadve}@illinois.edu

Imperfect Moore's law scaling has led to recent trends that consider systems that generate incorrect outputs. The emergent field of approximate computing considers deliberate, but controlled, relaxation of correctness for better performance or energy.

While the fast growing popularity of the field indicates the general acceptance, even eagerness, of the idea as a one plausible solution, there are many hurdles to the practical application of approximate computing in today's system that is preventing it from realizing it's full potential. Many of these hurdles will naturally be surmounted by the evolution of the field, but it is imperative that a strong fundamental base be established upon which future research directions can be built.

Standardized Benchmarks and Realistic Evaluations

Key among the obstacles are availability of benchmark suites and standards for their evaluation. While benchmarks for approximate computing are rapidly emerging, standards for evaluating the various approximation techniques are still developing. Approximation techniques are often measured based on how much accuracy is lost in the final output. There are many metrics that may be used to measure this accuracy loss and approximation techniques are evaluated based on the how much error is introduced in the benchmark output. Both the metric and the margin of error (calculated using the chosen metric) tolerable to the end user are highly application specific. Even within a single application, there may be different metrics that can be used to measure the error (or deviation) in the output. For instance, let us consider the PARSEC benchmark Blackscholes which outputs pricing for a portfolio of European options. Depending on the input size the portfolio can consist of anywhere between 4,096 and 65,536 options. The first problem that arises is how to calculate the error (introduced by the approximation technique) in a given option price. Do we calculate the dollar value difference between the approximately calculated output (*Approximate output*) and the precise output(*Golden output*) or maybe the relative error between the two? While relative error may be fine in some cases, it may be unreasonable in cases where even small relative errors lead to tangible losses in dollar amounts. Perhaps a combination of the two maybe the most appropriate and realistic metric.

Once a metric has been chosen to evaluate individual option prices, we can then determine the error in the entire portfolio (the full output of the benchmark). Calculating the average error, while determining the extent of the accuracy loss across the entire output, tends to undermine significant and often unacceptable errors in individual option prices in a large portfolio. The next step is choosing an acceptable threshold to guarantee error bounds in the benchmark outputs. Often an error threshold of around 10% is used [1, 2, 3, 4, 5], but in reality this threshold is too large to be practical in real world scenarios. For example 10% of a $20 option is $2, which is unacceptable error margin for most financial transactions. Even a 3-5% error threshold might too much for these types of applications. While there are works that use lower error bounds [6, 7], we need to be much more aggressive as a community in defining what is an acceptable error.

As can be seen from the above, there is little consensus in what is the right metric and error threshold even for an application like Blackscholes which has simple numerical outputs and whose usage model is straightforward to understand. These questions are harder to answer in other applications that may utilize sensory inputs and outputs such as audio and video. While these applications have a lot of potential for approximate computing, gauging and bounding errors in these are even more difficult without understanding the usage model of the outputs. A pixel by pixel comparison of two images to produce a numerical error percentage may not be useful without understanding how the final image will be used. In some cases, it may not be the quantity of the error but rather the quality that may be important. A good example is images that are used in surveillance, in which where the error is on the image is equally important as understanding how much error has been generated. It is imperative that a consensus is reached in defining standard benchmarks and common-sense metric(s) that can be used as platforms to gauge approximation techniques. We also must as a community embrace tighter and more practical er-

*This work was supported in part by the National Science Foundation under Grant CCF-1320941 and by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

ror bounds which will hasten the acceptance of approximate computing as a legitimate and practical technique to counter the demise of Moore's law.

Programmer Expertise and Automation

Another obstacle in the path of widespread adoption of approximate computing is its lack of application to a wide variety of programs. A significant reason for this is the need for expert programmer knowledge to distinguish locations in the application which may be amenable to approximation vs those that need precise computation [8, 9]. As a result, many potential opportunities for approximation may be overlooked due to insufficient or inaccurate application information. While minimal programmer input such as the end to end program error bound and possibly the higher order metric for error calculation are necessary, it is unreasonable to expect the programmer to provide details at the instruction and data level which can be then be exploited for approximation. Relieving the programmer of this burden will open up many new programs to the potential benefits of approximations. There is a need for tools and techniques that will take the end to end error bound supplied by the programmer and use it to automatically identify lower level program constructs that may be used by the system to apply approximation techniques on.

Towards this goal we developed a tool called *Approxilyzer* which takes a step towards building an automated framework to help identify the first order approximation potential of applications without any programmer expertise. The Approxilyzer framework extends the resiliency tool Relyzer [10] towards application in approximate computing. Relyzer aims to generate a comprehensive resiliency profile of applications in the presence of single bit perturbations. It uses a combination of fault injections and program analysis to determine the outcome of an execution when a single-bit fault is injected in a given bit pertaining to a given dynamic instruction (referred to as a *Fault Site*) in an execution. The outcomes predicted by Relyzer include whether the fault will be masked, detected, or produce a Silent Data Corruption (SDC). Approxilyzer expands upon Relyzer by introducing the notion of *quality* to the SDCs based on how far the corrupted output deviates from the fault free execution's output. Armed with the information about the *quality* of SDCs, Approxilyzer can now automatically identify instructions that may be amenable to approximation based on their behavior in the presence of perturbations. For example, instructions that produce no errors or only produce errors of acceptable *quality* in the presence of faults are considered as possible candidates for approximation. Using Approxilyzer, the exploration space of instructions to consider for different approximation techniques can be significantly reduced and it can help programmers identify code segments for more targeted experiments. The only input needed from the programmer is the metric to measure the *quality* of the output corruptions and possibly an end-to-end error bound (this is also not required if the aim is to do analysis or tuning based on different error thresholds).

We envision a tool like Approxilyzer being an intrinsic part of a larger systematic and automated framework for Approximate computing that will incorporate a single workflow; from analyzing an application for approximation opportunities to exploiting them by using the most optimal technique to satisfy given constraints. There are many questions that will arise in the process of building this workflow: What role will the compiler play in such a framework? What is the best axis of approximation - instruction or data? How do we deal with input dependency? How do we make individual techniques modular so they can be integrated into a single workflow? and so on. We urge the community to strive toward building such systematic end-to-end frameworks while working on individual techniques that we hope will serve as essential building blocks in this larger goal.

Conclusion

One thing is clear - while our techniques to achieve performance and energy efficiency can be approximate, the tools and methodology used to apply and evaluate them cannot. Only by standardizing benchmarks and their evaluation methodologies, using tight and practical error bounds, and building systematic frameworks that make it easy for programmers of all skill levels to automatically apply approximation techniques can we further the field to have widespread and lasting impact. Without these we are doing this much promising field a disservice that will prevent approximate computing from truly achieving it's potential.

1. REFERENCES

[1] 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*, PLDI '11, (New York, NY, USA), pp. 164–174, ACM, 2011.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pp. 449–460, 2012.

[3] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, (New York, NY, USA), pp. 13–24, ACM, 2013.

[4] J. S. Miguel, M. Badr, and N. E. Jerger, "Load value approximation," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, (Washington, DC, USA), 2014.

[5] J. San Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelganger: A cache for approximate computing," in *International Symposium on Microarchitecture*, 2015.

[6] D. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 554–566, June 2015.

[7] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural acceleration for gpu throughput processors," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.

[8] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels," *SIGPLAN Not.*, vol. 49, pp. 309–328, Oct. 2014.

[9] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, (New York, NY, USA), pp. 33–52, ACM, 2013.

[10] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting Application-Level Fault Equivalence to

Analyze Application Resiliency to Transient Faults," in *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.