Department: Software Technology

Editor: Markus Schordan

Harshitha Menon

Center for Applied Scientific Computing,

Lawrence Livermore National Laboratory

Email: [harshitha@llnl.gov](mailto:harshitha@llnl.gov)

James Diffenderfer

Center for Applied Scientific Computing,

Lawrence Livermore National Laboratory

Email: diffenderfer2@llnl.gov

Giorgis Georgakoudis

Center for Applied Scientific Computing,

Lawrence Livermore National Laboratory

Email: georgakoudis1@llnl.gov

Ignacio Laguna

Center for Applied Scientific Computing,

Lawrence Livermore National Laboratory

Email: ilaguna@llnl.gov

Michael O. Lam

James Madison University

Email: [lam2mo@jmu.edu](mailto:lam2mo@jmu.edu)

Daniel Osei-Kuffuor

Center for Applied Scientific Computing,

Lawrence Livermore National Laboratory

Email: [oseikuffuor1@llnl.gov](mailto:oseikuffuor1@llnl.gov)

Konstantinos Parasyris

Center for Applied Scientific Computing,

Lawrence Livermore National Laboratory

Email: [parasyris1@llnl.gov](mailto:parasyris1@llnl.gov)

Jackson Vanover

University of California, Davis  
Email: [jdvanover@ucdavis.edu](mailto:jdvanover@ucdavis.edu)

Approximate High Performance Computing : A Fast and Energy Efficient Computing Paradigm in the Post-Moore Era

To gain computing performance is to bring the power of computing to bear on unprecedented problems at ever-growing scale and fidelity. However, as we reach the limits of Moore’s law, which states that the number of transistors per square inch will double every 18 months, and the end of Dennard scaling, where power use is proportional to area, new approaches are needed to shape the future of computer architecture and software systems. One promising approach is Approximate Computing. Techniques for Approximate Computing aim to exploit the fact that many applications can tolerate some degree of error in their output results. By giving up some amount of correctness, such techniques allow the system to use less computational resources and consume less energy. Put more succinctly, Approximate Computing trades computation accuracy for improved performance [21, 11, 15] and energy-efficiency [10, 3, 22, 19, 5].

Approximate Computing (AC) techniques have been effectively deployed in domains like image  
processing, visualization, and machine learning [38, 21, 44, 45] – all fields in which the accuracy of a program’s output is often judged using the limited capabilities of human perception. For example, processed images can tolerate significant errors so long as they are imperceptible to the human eye [38]. However, in the domain of scientific High Performance Computing (HPC), even small errors can lead to unstable algorithms and incorrect numerical results. This presents additional challenges that must be addressed in the application of Approximate Computing techniques to HPC.

Despite the challenges presented by this domain, the concept of trading accuracy for performance is not new in HPC and is commonly applied for two main purposes: Firstly, approximation is required to discretize continuous quantities. For instance, polynomial approximations of functional forms, as described in [51], are commonly used in numerical linear algebra to reduce the computational complexity of modeling and reasoning about real-world processes. We also see this in the spatial and temporal discretization schemes common to HPC. Secondly, approximation helps to make better use of computing resources. Consider a fundamental piece of scientific computation: solving a system of linear equations. Computing a solution via mixed-precision iterative refinement on NVIDIA Tensor Cores, as detailed in [18], achieves up to 5-fold increases in performance and energy-efficiency without sacrificing numerical stability. Despite these applications, the impact of Approximate Computing in scientific HPC has been limited; expert knowledge is required about both the approximation technique and the program to be approximated in order to yield performance gains without excessively deteriorating accuracy. As such, more research is needed in order to lower the barrier to entry, allowing these techniques to be deployed effectively and widely in HPC.

For approximate computing to gain widespread acceptance in HPC, three main considerations  
must be taken into account: accuracy, efficiency, and ease of use. Approximate Computing techniques must be able to produce results that are within a defined error threshold, while also maintaining the stability and correctness of the algorithms used. This necessitates a deep understanding of the approximate techniques and applications involved, as well as meticulous design and evaluation of the Approximate Computing methods. Ensuring efficiency is also critical. Approximate Computing techniques must be able to deliver performance gains that are significant enough to make them worthwhile. This requires careful design and optimization of the Approximate Computing techniques as well as support at different levels of the software stack to ensure that they are able to deliver the performance benefits promised. Finally, ease of use is essential for the wider adoption of Approximate Computing. The techniques must be easy for developers and users to understand and use effectively. This includes having tools and frameworks that make it easy to apply Approximate Computing techniques in HPC.

This article highlights some of our recent works to address these challenges and list open challenges in approximation.

Approximate Computing techniques

In the field of Approximate Computing, there are numerous strategies available, ranging from lower-level hardware techniques to higher-level software techniques. Some of the hardware-based techniques include approximate floating-point multipliers [11, 39, 26], inexact adder [23, 17], dropping a fraction of load requests that miss the cache [54]. Other hardware techniques involve the use of voltage scaling to reduce the energy consumption of circuits [5, 38, 10, 20, 48]. Proposals have been put forth for hardware support for Approximate Computing, such as ISA extensions providing support for approximate arithmetic [10]. In addition, heterogeneous architectures with neural-network-based approximate accelerators have been used for function approximation [15, 34, 9, 37].

Various software techniques have also been proposed to reduce computational complexity and improve performance and energy efficiency. For instance, loop perforation [21, 14] skips specific tasks or iterations of a loop in a computational kernel in order to reduce the cost, while function memoization [33, 24, 44] stores computed entries of computationally expensive kernels in a look-up table. Among the different Approximate Computing methods, mixed-precision has recently gained popularity. It involves using multiple levels of precision for floating-point data and arithmetic operations to balance accuracy and performance, and has been shown to significantly enhance the performance of scientific applications in recent studies [25, 32, 27, 43]. Additionally, there exists several algorithmic approaches, such as communication avoiding algorithms [13, 6] and relaxed synchronization [40, 53], to improve performance. Some algorithmic methods exploit single precision operations whenever possible and resort to double precision to iteratively refine the solution to provide the full double precision results [30, 2] while attaining significant speedups.

Many of the Approximate Computing strategies lack a priori error analysis on their impact on  
the quality of the output of the algorithms they approximate, which means that they would have to be deployed as a preprocessing tool to first identify whether the approximation satisfy the required error tolerance. In our recent work, we formalized a general framework for designing error-bounded Approximate Computing kernels and applied this framework to the dot product kernel to design qdot [7]. We theoretically proved and empirically demonstrated that qdot bounds the relative error introduced by the approximation. Our experiments on the Conjugate Gradient (CG) and power method algorithms demonstrate that high levels of approximation can be introduced into these algorithms without degrading their performance. Furthermore, the formalized general framework could be used to design other error-bounded Approximate Computing strategies for kernels other than the dot product.

Given the wide range of available approximate techniques, it is necessary to have a benchmark  
suite to assess their suitability for HPC applications. Our HPC-MixPBench benchmark suite [36], that represent common HPC applications, can be used for evaluating different Approximate Computing analysis. Our set of benchmarks is composed of ten kernel codes and seven application codes that represent common HPC workloads. While we demonstrated the capability of the benchmark suite by evaluating them using mixed-precision and reporting several insights, it can be easily adapted to use other approximate techniques. Figure 1 gives a high-level overview of the HPC-MixPBench framework consisting of a runtime library for profiling, a harness to execute the benchmarks, and a verification library to evaluate the specified quality metric. As HPC systems become more heterogeneous, use of Approximate Computing techniques is expected to become more widespread in scientific applications and these evaluations could programmers choose the appropriate approximate method for their application or workload.

Figure 1: **High level overview of the HPC-MixPBench framework.** HPC-MixPBench includes a set of benchmark applications, a runtime library for profiling, and a verification library to evaluate the specified quality metric. HPC-MixPBench has a harness to execute the benchmarks based on the information provided in the YAML configuration file.

Methods to assist in the design of Approximate Computing applications

Developers wishing to optimize program performance via the use of Approximate Computing techniques must take care that the error induced by such approximations do not deteriorate the output quality beyond some prescribed threshold. Despite the success of Approximate Computing in several domains, there is a significant challenge to be addressed to adopt Approximate Computing techniques in scientific computing applications: the lack of methods to identify error resilient code regions. Developers of error-sensitive high-performance computing (HPC) applications must allocate significant effort toward the identification of approximable kernels within the program.

Several methods have been proposed to help guide the process of applying approximations in  
code. Roy et al. [42] presents ASAC, a software framework designed to automatically identify data that can be approximated within a program. Their approach involves gathering information on the range of a given variable and then perturbing its value to measure the resulting output, which is then used to calculate its sensitivity. Chippa et al. [4] used resiliency characterization to aid in approximation by injecting random errors into the output of the kernel and studying it’s resiliency profile. Dynamic search-based approaches [43, 28] have been used to evaluate mixed-precision configurations and determine the best configuration that satisfies the error threshold. Interval analysis [52] and algorithmic differentiation [12] have also been used to study the impact of approximation and guide the selection of code regions that can be approximated. Finally, in [16, 41], the authors rely on dynamic checks to control the quality of the output and to gain insights into the potential impact of approximation on the output’s quality.

One of the primary concerns with many of the existing tools is their limited scalability, which  
has prevented their effective use in HPC workloads. To address this limitation, we have developed several tools that enable us to analyze the impact of using approximations on the output of an application. One such tool, ADAPT, uses Automatic Differentiation to identify regions of the code that are amenable to approximations [31, 29]. ADAPT provides accurate estimate of the output error due to lowering the precision of variables and produces a floating-point precision sensitivity profile to guide programmers in the development of a mixed-precision version.  Figure 2 shows the call graph at the function level highlighting functions that need to be in higher precision for LULESH, a hydrodynamics proxy application. We used the profile as a guide to develop a mixed-precision version for a CUDA implementation of LULESH. The mixed-precision version resulted in a speedup of 1.2× while maintaining an error threshold of 2.0e−11.

Figure 2: **Output of analysis from ADAPT highlighting precision requirements of different functions in LULESH.** Functions in LULESH highlighted in red require higher precision whereas the ones in green can be in lower precision. ADAPT analysis indicates that the function CalcTimeConstraintsForElems, which pertains to the calculation of time step constraints, can be performed  
in lower precision. Functions CalcElemShapeFunctionDerivatives and CalcElemVelocityGradient, which compute the rate of distortion of the element volume, can also be evaluated in lower precision.

To meet the need for analysis tools that allow for the expression of arbitrary approximations and analysis of the response of the application to these approximations. To this end, we proposed Puppeteer [8]. Puppeteer is a novel method that ranks the code regions based on amenability to approximation and uses uncertainty quantification methods to measure the sensitivity of application outputs to approximation errors. A developer annotates possible application code regions and Puppeteer estimates the sensitivity of each region. One can then utilize Approximate Computing techniques on these regions. Puppeteer successfully identifies insensitive regions on different benchmarks to obtain speedups of 1.8× for DCT, an image compression application, without significantly reducing the PSNR (Peak Signal-to-Noise Ratio) by perforating the least sensitive kernels. Figure 3 shows the operation of Puppeteer.



Figure 3: Puppeteer operates in two phases. During the trace phase, Puppeteer executes the annotated application and generates the description of the error domain. In the deployment phase, the UQ-Orchestrator uses external UQ libraries to sample the error domain, perform distributed evaluation on the samples, and compute the sensitivity values of the kernels.

Programming Language Support for Approximate HPC

Numerous frameworks have been developed to facilitate the use and exploration of the tradeoffs associated with Approximate Computing techniques. GREEN [1] is an API for loop and function level approximations and provides functions that generate application error models. SAGE [46] is a compiler and a runtime system targeting GPUs to allow automatic generation of approximate kernels in machine learning and image processing applications. The SAGE compiler automatically creates multiple approximate kernel versions and selects suitable kernel to approximate at runtime to meet a user provides quality threshold. ACCEPT [47] is a compiler framework built on top of the LLVM compiler infrastructure. It supports programmer annotations and provides a quality aware auto-tuner that automatically chooses the best approximation strategy. ApproxHPVM [49] uses a compiler intermediate representation to enable accuracy aware compiler optimizations for ML applications. The authors in [50] extend ApproxHPVM with an auto-tuning framework for approximate aware optimizations of tensor-based applications.

Majority of Approximate Computing frameworks are not designed to support parallel applications, a critical aspect of HPC applications. As a result, we designed and developed a new programming framework, called HPAC [35], to support the use of Approximate Computing techniques in HPC applications. HPAC is a pragma-based Approximate Computing framework that includes some of the state-of-the-art Approximate Computing techniques (loop perforation, input/output memoization) and is composable with OpenMP. It extends Clang/LLVM compiler and OpenMP runtime system to enable easy integration with existing OpenMP codes to identify approximation opportunities. It is also extensible to support new approximation techniques, providing a convenient and versatile framework to explore various approximation strategies.



Figure 4: HPAC is comprised of two builing blocks, the core, and the harness. The core implements the approximate programming model. It extends the Clang/LLVM compiler and provides runtime support. The harness facilitates easy exploration of the approximate design space, which includes the different code regions, various parameters of the approximate methods, output accuracy, and overall performance gains.

Figure 4 shows the overall design of HPAC. HPAC enabled us to understand how approximation composes with OpenMP-level parallelism. In our study, we conducted a thorough evaluation of the effectiveness of approximate algorithms on eight different OpenMP HPC benchmarks and characterized their effect on the application accuracy and performance. Our findings suggest that comprehending the effects of parameter selection for an approximate method is not always intuitive. For instance, choosing the optimal activation function for memoization techniques is challenging as one needs to find a balance between frequency of activation, error minimization, and performance gains. Similarly, understanding the effects of perforation parameters on the application error can be difficult. With the support of HPAC, developers can easily evaluate several Approximate Computing methods for different regions of code and select the approximation technique and suitable parameters that meets their performance and accuracy criteria.

Conclusion

With the ever-increasing demand for higher computational power, Approximate Computing provides a practical way to meet these needs while also reducing energy consumption and cost. A key challenges in Approximate Computing is the analysis and handling of errors that are introduced by approximate techniques, particularly for scientific applications where accuracy is important. There is significant risk that the errors introduced may accumulate and result in incorrect results. Therefore, effective and scalable error analysis and mitigation strategies are necessary to ensure the use of these techniques in scientific codes. In order to fully realize the potential of Approximate Computing, it is essential to adopt a co-design strategy that combines both hardware and software techniques. This allows for effective integration of approximate techniques while also addressing programmability issues. Despite these challenges, the future outlook for Approximate Computing is promising and it is anticipated to gain further momentum in the coming years. As more applications realize the potential of approximate techniques and more research is conducted, we can expect to see increase in use of Approximate Computing techniques to obtain a more efficient and reliable computing solution.

Acknowledgements

The views and opinions of the authors do not necessarily reflect those of the U.S. government or Lawrence Livermore National Security, LLC neither of whom nor any of their employees make any endorsements, express or implied warranties or representations or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of the information contained herein. This work was prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-MI-845434) and was supported by the LLNL-LDRD Program under Project No. 20-ERD-043.