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

Harshitha Menon, James Diffenderfer, Zane Fink, Giorgis Georgakoudis, Ignacio Laguna, Daniel Osei-Kuffuor, Konstantinos Parasyris, Jackson Vanover

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, where the computation accuracy is traded for better performance and energy efficiency. The concept of approximate computing is based on the idea that many applications can tolerate some degree of error in their output results. This allows the system to use less computational resources and consume less energy, thereby improving performance and reducing power consumption.

Approximate computing techniques have mostly been applied in non-High Performance Computing (HPC) domains, such as image processing, machine learning, and visualization. These workloads are often designed for human eyes, which can tolerate some error without significantly affecting the accuracy of the output. However, the impact of even small errors in scientific computing applications can lead to unstable algorithms and incorrect results. Scientific computing often requires high levels of accuracy and stability, making it much more challenging to apply approximate computing techniques in this domain.

Despite these challenges, the concept of trading accuracy for performance is not new and has been applied in the field of HPC for some time. Some of these techniques include spatial and temporal discretization schemes, approximate numerical algorithms, and mixed-precision computing. Approximate Computing has been used in HPC to serve two main purposes. Firstly, they are utilized to reduce computational complexity. For instance, polynomial approximations of functional forms, as described in Trefethen and Bau (1997), are commonly used. Secondly, these strategies help to make better use of computing resources. For example, iterative refinement on NVIDIA Tensor Cores, as detailed in Haidar et al. (2020), is frequently employed. In all of these strategies, the primary goal is to maintain the accuracy of the desired quantity of interest while improving performance. However, they require expert knowledge about the application as well as the nuances of the techniques to give performance gains while not affecting the accuracy. As a result, the impact of approximate computing in scientific computing has been limited, and more research is needed to find ways to apply these techniques effectively in this field.

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

There are various strategies for approximation, including hardware and software techniques. Hardware-based techniques include approximate floating-point multipliers Froehlich et al. (2018); Rehman et al. (2016), heterogeneous architectures with neural-network-based approximate accelerators Grigorian et al. (2015), and dropping a fraction of load requests that miss the cache Yazdanbakhsh et al. (2016). Some proposals have been made for hardware support for approximate computing, such as ISA extensions providing support for approximate arithmetic Esmaeilzadeh et al. (2012).

Various software techniques have also been proposed to reduce computational complexity and computing only when necessary. For instance, loop perforation Mittal (2016) skips specific iterations of a loop in a computational kernel in order to reduce the cost, while function memoization Michie (1968) 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 Kotipalli et al. (2019); Menon et al. (2018); Laguna et al. (2019). As HPC systems become more heterogeneous, mixed precision is expected to become more widespread in scientific applications.

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 Diffenderfer et al. (2022). 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.

There is a definite need to have a benchmark suite to evaluate different approximate computing methods catering to HPC. Our HPC-MixPBench benchmark suite Parasyris et al. (2020), 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. To demonstrate the capability of the benchmark suite, we evaluated them using *mixed-precision*, one of the most popular approximate computing techniques, and reported several insights. 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 t evaluate the specified quality metric. The findings we report in the paper can help programmers choose the appropriate mixed-precision method for their application or workload.

Methods to assist in the design of AC 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 AC 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. We list several tools that we developed to analyze the impact of using approximations on the output of an application.

There is a need for analysis tools that provide mechanisms to express arbitrary approximations and analyze the response of the application to these approximations. We proposed *Puppeteer* Diffenderfer* et al. (2022), a novel method to rank code regions based on amenability to approximation. *Puppeteer* 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 AC techniques on these regions. *Puppeteer*

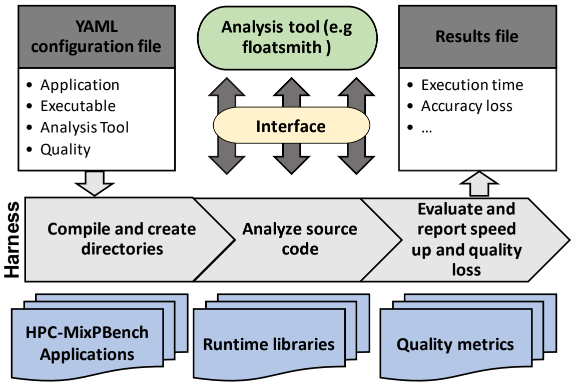

Figure 1: **High level overview of the HPC-MixPBench framework.** HPC-MixPBench includes a set of benchmark applications, a runtime library for profiling, and a verification library to evaluate the specified quality metric. HPC-MixPBench has a harness to execute the benchmarks based on the information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

successfully identifies insensitive regions on different benchmarks. For example, we obtain speedups of 1.18×, 1.8×, and 1.3× for HPCCG, DCT, and BlackScholes, respectively. Figure 2 shows the operation of *Puppeteer*.

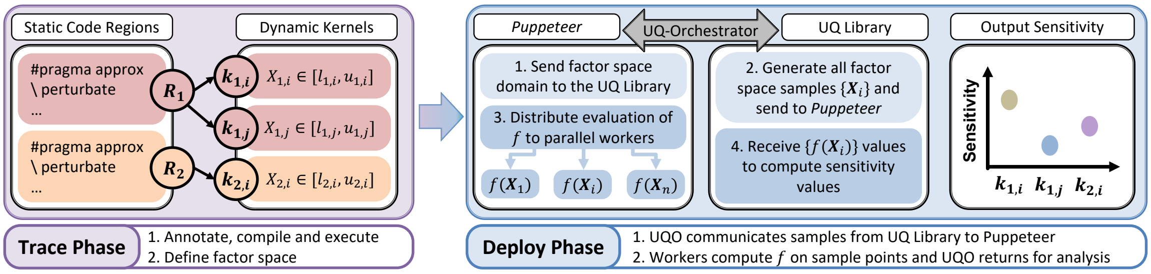
Static Code Regions
Dynamic Kernels
#pragma approx \ perturbate …
#pragma approx \ perturbate …
R_1
R_2
$k_{1,i}$
$k_{1,j}$
$k_{2,i}$
$X_{1,i} \in [l_{1,i}, u_{1,i}]$
$X_{1,j} \in [l_{1,j}, u_{1,j}]$
$X_{2,i} \in [l_{2,i}, u_{2,i}]$
Trace Phase
1. Annotate, compile and execute
2. Define factor space
Puppeteer
UQ-Orchestrator
UQ Library
Output Sensitivity
1. Send factor space domain to the UQ Library
3. Distribute evaluation of f to parallel workers
$f(X_1)$
$f(X_i)$
$f(X_n)$
2. Generate all factor space samples $\{X_i\}$ and send to Puppeteer
4. Receive $\{f(X_i)\}$ values to compute sensitivity values
Sensitivity
$k_{1,i}$
$k_{1,j}$
$k_{2,i}$
Deploy Phase
1. UQO communicates samples from UQ Library to Puppeteer
2. Workers compute f on sample points and UQO returns for analysis

Figure 2: *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.

Using Automatic Differentiation for Approximate Computing error estimation

Building on our previous work of using Automatic Differentiation (AD) for mixed-precision analysis (Menon et al. (2018)), we developed an automatic and efficient method to perform error estimation. We used Clad, an efficient automatic-differentiation tool built as a plugin to the Clang compiler and based on the LLVM compiler infrastructure, to evaluate approximation errors in C++ code. We generated and injected error estimation code directly into the derivative source, resulting in a significant speedup compared to other tools performing similar floating-point error analysis. The generated code also becomes a candidate for better compiler optimizations contributing to lesser runtime performance overhead.

Programming Language Support for Approximate HPC

We designed and developed two new programming frameworks to support the use of Approximate Computing techniques in HPC applications.

HPAC is a new, pragma-based approximate computing framework that includes state-of-the-art approximate computing techniques (loop perforation, input/output memoization) and is composable with OpenMP. HPAC 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 3 shows the overall design of HPAC. HPAC enabled us to understand how approximation composes with OpenMP-level parallelism. We presented an exhaustive evaluation, studying the effectiveness of approximate algorithms on eight different OpenMP HPC benchmarks and characterized their effect on the application accuracy and performance. Also, it is extensible by design and allows for seamless integration of new approximation techniques. With the support of HPAC, developers can effortlessly evaluate several approximate computing methods for different regions of code and select the approximation technique that meets their performance and accuracy criteria Parasyris et al. (2021)).

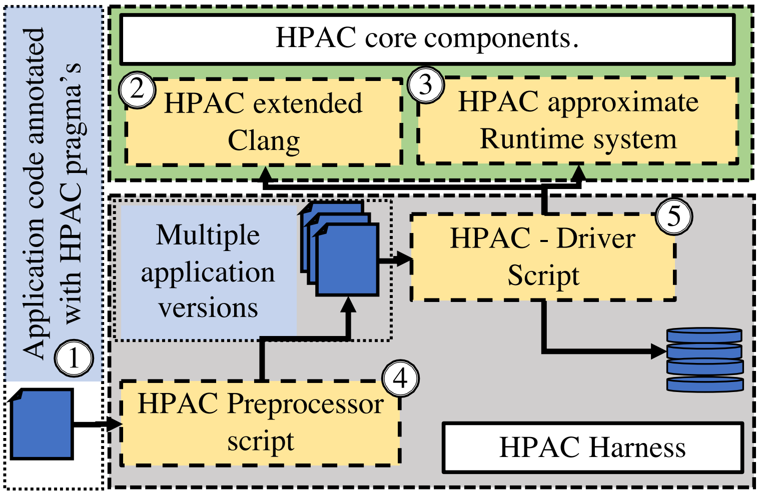
Application code annotated with HPAC pragma's
1
HPAC core components.
2
HPAC extended Clang
3
HPAC approximate Runtime system
Multiple application versions
5
HPAC - Driver Script
4
HPAC Preprocessor script
HPAC Harness

Figure 3: 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.

Conclusion

References

Diffenderfer, J., D. Osei-Kuffuor, and H. Menon (2022). A framework for error-bounded approximate computing, with an application to dot products. *SIAM Journal on Scientific Computing 44*(3), A1290–A1314. LLNL-JRNL-820357.

Diffenderfer*, J., K. Parasyris*, H. Menon, I. Laguna, J. Vanover, R. Vogt, and D. Osei-Kuffuor (2022). Approximate computing through the lens of uncertainty quantification. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '22, New York, NY, USA. Association for Computing Machinery. LLNL-CONF-829328.

Esmaeilzadeh, H., A. Sampson, L. Ceze, and D. Burger (2012, March). Architecture support for disciplined approximate programming. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, London, England, UK, pp. 301–312. Association for Computing Machinery.

Froehlich, S., D. Große, and R. Drechsler (2018, August). Towards Reversed Approximate Hardware Design. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pp. 665–671.

Grigorian, B., N. Farahpour, and G. Reinman (2015, February). BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 615–626.

Haidar, A., H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham (2020). Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proc. Roy. Soc. London A 476*(2243), 20200110.

Kotipalli, P. V., R. Singh, P. Wood, I. Laguna, and S. Bagchi (2019). Ampt-ga: automatic mixed precision floating point tuning for gpu applications. In *Proceedings of the ACM International Conference on Supercomputing*, pp. 160–170.

Laguna, I., P. C. Wood, R. Singh, and S. Bagchi (2019). GPUMixer: Performance-Driven Floating-Point Tuning for GPU Scientific Applications. In M. Weiland, G. Juckeland, C. Trinitis, and P. Sadayappan (Eds.), *High Performance Computing*, Cham, pp. 227–246. Springer International Publishing.

Menon, H., M. O. Lam, D. Osei-kuffuor, M. Schordan, S. Lloyd, K. Mohror, and J. Hittinger (2018). ADAPT : Algorithmic Differentiation Applied to Floating-Point Precision Tuning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18)*, Dallas, Texas, pp. 48:1–48:13. IEEE Press.

Menon, H., M. O. Lam, D. Osei-Kuffuor, M. Schordan, S. Lloyd, K. Mohror, and J. Hittinger (2018). Adapt: Algorithmic differentiation applied to floating-point precision tuning. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 614–626.

Michie, D. (1968). "memo" functions and machine learning. *Nature 218*, 19–22.

Mittal, S. (2016). A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR) 48*(4), 1–33.

Parasyris, K., G. Georgakoudis, H. Menon, J. Diffenderfer, I. Laguna, D. Osei-Kuffuor, and M. Schordan (2021). Hpac: Evaluating approximate computing techniques on hpc openmp applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA. Association for Computing Machinery. LLNL-CONF-821216. **Deputy Director for Science and Technology Excellence in Publication Award** and **SC Best Reproducibility Advancement Award**.

Parasyris, K., I. Laguna, H. Menon, M. Schordan, D. Osei-Kuffuor, G. Georgakoudis, M. O. Lam, and T. Vanderbruggen (2020). Hpc-mixpbench: An hpc benchmark suite for mixed-precision analysis. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 25–36. LLNL-CONF-809027.

Rehman, S., W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel (2016, November). Architectural-space exploration of approximate multipliers. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8.

Trefethen, L. N. and D. Bau (1997). *Numerical Linear Algebra.* SIAM.

Yazdanbakhsh, A., G. Pekhimenko, B. Thwaites, H. Esmaeilzadeh, O. Mutlu, and T. C. Mowry (2016). Rfvp: Rollback-free value prediction with safe-to-approximate loads. *ACM Transactions on Architecture and Code Optimization (TACO) 12*(4), 1–26.