AXBENCH: A Multi-Platform Benchmark Suite for Approximate Computing

Amir Yazdanbakhsh Divya Mahajan
Pejman Lotfi-Kamran† Hadi Esmaeilzadeh

Alternative Computing Technologies (ACT) Lab
School of Computer Science, Georgia Institute of Technology

†School of Computer Science, Institute for Research in Fundamental Sciences (IPM)

{a.yazdanbakhsh, divya_mahajan}@gatech.edu plotfi@ipm.ir hadi@cc.gatech.edu

http://axbench.org

***Abstract*— As we enter the dark silicon era, the benefits from classical transistor scaling are diminishing. The current paradigm of microprocessor design falls significantly short of the historical cadence of performance improvements. To address these challenges, there is a need to go beyond traditional approaches and explore unconventional paradigms in the computing landscape. One such paradigm is *approximate computing* that embraces imprecision and relaxes the traditional abstraction of ''near-perfect'' accuracy across the system stack. Approximate computing promises to deliver significant performance and energy efficiency gains when small losses of quality are permissible. As approximate computing attracts more attention, having a general, diverse, and representative set of benchmarks to evaluate different approximation techniques becomes necessary. In this paper, we introduce AXBENCH, a general, diverse and representative set of benchmarks for CPUs, GPUs, and hardware design. We judiciously select and develop each benchmark to cover a diverse set of domains such as financial analysis, machine vision, medical imaging, machine learning, data analytics, scientific computation, signal processing, image processing, robotics, and compression. Furthermore, to enable a wide range of studies, each benchmark is paired with three different input datasets. AXBENCH also provides necessary annotations to mark the approximable regions of code and the application-specific quality metric to assess the output quality of each application.**

***Index Terms*—GPU, CPU, Hardware Design, Benchmark, Approximate Computing.**

I. INTRODUCTION

THE diminishing benefits from Dennard scaling and Moore's law have hampered the historical cadence of performance improvements in microprocessor design. More fundamentally, recent decline in the decades-long power-scaling has led to the *dark silicon* problem [1]. The predicament of dark silicon is hindering the devices to obtaining benefits proportional the increase in available resources. This paradigm shift in the computing landscape is driving both the industry and research community to explore viable solutions and techniques to maintain the traditional scaling of performance and energy efficiency.

Approximate computing presents itself as an approach that promises significant efficiency gains at the cost of some quality degradation for applications that can tolerate inexactness in their output. As approximate computing gains popularity as a viable alternative technique to prolong the traditional scaling of performance and energy efficiency improvements, it has become imperative to have a representative set of benchmarks for a fair evaluation of different approximation techniques.

A benchmark suite for approximate computing has to have several features as we explain in the following paragraphs.

Diverse set of applications. As various applications in different domains like finance, machine learning, image processing, vision, medical imaging, robotics, 3D gaming, numerical analysis, etc. are amenable to approximate computation, a good benchmark suite for approximate computation should be diverse to be representative of all these applications.

Multiple platforms. Approximate computing can be applied to various levels of the computing stack and through different techniques. Approximate computing is applicable to both hardware and software (*e.g.*, [2], [3], [4]). A good benchmark suite for approximate computation should be useful for evaluating all of these possibilities. Being able to evaluate vastly different approximation techniques using a common set of benchmarks enables head-to-head comparison of different approximation techniques.

Different input datasets. With approximate computing, the natural question to ask is whether a specific technique will work across a range of input datasets. Providing different input datasets enables the users to perform a wide range of studies on each benchmark and analyze the effect of their approximation techniques across different input datasets.

Application specific quality metric. Each approximation technique introduces different levels of quality loss in the applications' output. Therefore, it is inevitable for an approximation benchmark suite to include a quality metric to evaluate the applications' output quality loss. Furthermore, as different applications generate different types of output, they require different application quality metrics. For example, *image difference*, which is an appropriate quality metric for image processing applications, is not applicable to a robotic application, which changes the location of a robotic arm.

This paper introduces AXBENCH, a diverse and representative set of benchmarks for evaluating various approximation techniques in CPUs, GPUs, and hardware design. AXBENCH covers diverse application domains such as machine learning,

robotics, arithmetic computation, multimedia, and signal processing. Moreover, AXBENCH comes with approximable region of benchmarks marked to facilitate evaluation of approximation techniques. Each benchmark is accompanied with three different sized input datasets (*e.g.* small, medium, and large) and an application specific quality metric. AXBENCH enables researchers to study, evaluate, and compare a wider range of approximation techniques on a diverse set of benchmarks in a straightforward manner.

We evaluate three previously proposed approximate computation techniques using AXBENCH benchmarks. We apply Loop Perforation [4] and Neural Processing Units (NPUs) to CPU [2] and GPU [5], and Axilog [3] to dedicated hardware. We find that loop perforation results in large output quality degradation and consequently, NPUs offer higher efficiency on both CPUs and GPUs. Moreover, we observe that, on CPU+NPU, significant opportunity remains to be explored by other approximation techniques. On GPUs, however, NPUs leverage most of the potential and leave little opportunity for other approximation techniques. Finally, we find that Axilog is effective at improving efficiency of dedicated hardware, though significant opportunity remains to be explored by other approximation techniques.

II. BENCHMARKS

One of the goals of AXBENCH is to provide a diverse set of applications to further facilitate research and development in approximate computing. Table I shows the benchmarks that are included in AXBENCH, their target platforms, domains, and the application specific quality metrics. In total, AXBENCH consists of 29 applications from diverse set of domains.

A. Common Benchmarks

AXBENCH provides a set of C/C++ benchmarks for CPUs, a set of CUDA benchmarks for GPUs, and a set of Verilog benchmarks for hardware design. Some algorithms are amenable for execution on all platforms. For these algorithms, AXBENCH provides all three implementations for CPUs, GPUs, and hardware design.

Inversek2j is used in robotic and animation applications. It uses the kinematic equation to compute the angles of 2-joint robotic arm. The input dataset is the position of the end-effector of a 2-joint robotic arm and the output is the two angles of the arm.

Sobel takes an RGB image as the input and produces a grayscale image in which the edges are emphasized.

B. Common CPU and GPU Benchmarks

For some algorithms, AXBENCH provides both C/C++ and CUDA implementations for both CPUs and GPUs.

Black-Scholes is a financial analysis workload. It solves partial differential equations to estimate the price for a portfolio of European options. Each option consists of different floating point values and the output is the estimated price of the option. **Jmeint** is a 3D gaming workload. The input is a pair of two triangles' coordinates in the 3D space and the output is a Boolean value which indicates whether the two triangles intersect or not.

C. Common CPU and Hardware-Design Benchmarks

For some algorithms, AXBENCH provides both the C/C++ implementation for execution on CPUs and the Verilog implementation for hardware design.

Forwardk2j is used in robotic and animation applications. It uses kinematic equations to compute the position of a robotic arm in a two-dimensional space. The input dataset consists of a set of 2-tuple angles of a 2-joint robotic arm and the output dataset is the computed (x,y)-position of the end-effector of the arm.

K-means is widely used in machine learning and data mining applications. It aims to partition a number of n-dimensional input points into k different clusters. Each point belongs to the cluster with the nearest mean. We use an RGB image as the input. The output is an image that is clustered in different color regions.

D. CPU Specific Benchmarks

Canneal is an optimization algorithm for minimizing the routing cost of a chip. Canneal employs the simulated annealing (SA) technique to find the optimum design point. At each iteration of the algorithm, Canneal pseudo-randomly swaps the netlist elements and re-evaluates the routing cost of the new placement. If the cost is reduced, the new placement will be accepted. In order to escape from the local minimum, the algorithm also randomly accepts a placement with a higher routing cost. This process continues until the number of possible swaps is below a certain threshold. The input to this benchmark is a set of netlist element and the output is the routing cost.

FFT is an algorithm that is used in many signal processing applications. FFT computes the discrete Fourier transform of a sequence, or its inverse. The input is a sequence of signals in time domain and the output is the signal values in frequency domain.

JPEG is a lossy compression technique for color images. The input is an uncompressed image (RGB). The JPEG algorithm performs a lossy compression and produces a similar image with reduced file size.

E. GPU Specific Benchmarks

Binarization is an image processing workload, which is frequently used as a pre-processor in optical character recognition (OCR) algorithms. It converts a 256-level grayscale image to a black and white image. The image Binarization algorithm uses a pre-defined threshold to decide whether a pixel should be converted to black or white.

Convolution operator can be used in a variety of domains such as machine learning and image processing. One of the application of convolution operator is to extract the feature map of an image in deep neural networks. In the image processing domain, it is used for image smoothing and edge detection. Convolution takes an image as the input. The output of the convolution is the transformed form of the input image.

FastWalsh is widely used in a variety of domains including signal processing and video compression. It is an efficient

algorithm to compute the Walsh-Hadamard transform. The input is an image and the output is the transformed form of the input image.

Laplacian is used in image processing for edge detection. The output image is a grayscale image in which all the edges are emphasized.

Meanfilter is used as a filter for smoothing (and removing the noises from) an image. The meanfilter replaces all the image pixels with the average value of their 3×3 window of neighbors. Meanfilter takes as input a noisy image. The output is the same image in which the noises are reduced.

Newton-Raphson is an iterative numerical analysis method. This method is widely used in scientific applications to find an approximation to the roots of a real-valued function. The Newton-Raphson method starts with an initial guess of the root value. Then, the method finds a better approximation of the root value after each iteration.

SRAD is a method that is widely used in medical image processing domain and is based on partial differential equations. SRAD is used to remove the correlated noise from the image without destroying the important image features. We evaluate this benchmark with a grayscale and noisy image of a heart. The output is the same image with reduced noise.

F. Hardware-Design Specific Benchmarks

Brent-Kung is one of the parallel prefix form of carry look-ahead adder. Brent-Kung is an efficient design in terms of area for an adder. The input dataset for this benchmark is a set of two random 32-bit integer numbers and the output is a 32-bit integer sum.

FIR filter is widely used in signal processing domain. One of the applications of FIR filter is to select the desired frequency of a finite-length digital input. We use a set of random values as the input dataset.

Kogge-Stone is one of the parallel prefix form of carry look-ahead adder. Kogge-Stone adder is one of the fastest adder design and is widely used for high performance computing in industry. We use a set of random two 32-bit integer values as input. The output is the summation of the corresponding values.

Neural-Network is an implementation of a feedforward neural network that approximates the Sobel filter. Such neural networks are used in a wide variety of applications including pattern recognition and function approximation. The benchmark takes as input an RGB image and the output is a grayscale image whose edges are emphasized.

Wallace-Tree is an efficient design for multiplying two integer values. The input is random 32-bit integer numbers and the output is the product of the corresponding numbers.

G. Input Datasets

As mentioned before, one of the main concern in any approximation technique is to find out whether the proposed technique work across different input datasets. Moreover, having multiple input datasets enable the users to thoroughly analyze the effect of their approximation techniques on the applications' output quality. To address these concerns, each benchmark in AXBENCH is accompanied with three different sized (*e.g.* small, medium, and large) input datasets. The small sized input dataset is provided to test the functionality of the program. The medium sized input dataset is included to explore and study different microarchitectural parameters for CPU and GPU applications with MARSS×86 and GPGPU-Sim, respectively. Finally, we include large dataset suitable for execution on the actual hardware. For the image applications, the small input dataset consists of *ten different 512×512* pixel images. The medium and large sized input dataset include *ten different 1024×1024* and *ten different 2048×2048* pixel images, respectively. In all the other applications, the small, medium, and large sized input dataset include 2^{12}, 2^{18}, and 2^{24} data points. Having different sized and diverse input datasets for each benchmark facilitate the evaluation of approximation techniques, enable the users to perform a wide range of studies, and help to better realize the effect of approximation on the applications.

TABLE I: The evaluated Benchmarks and their platforms, domains, and quality metrics.

benchmark	platform	domain	Quality Metric
binarization	GPU	Image Processing	Image Diff
blackscholes	CPU, GPU	Finance	Avg. Relative Error
brent-kung	ASIC	Arithmetic Computation	Avg. Relative Error
canneal	CPU	Optimization	Avg. Relative Error
convolution	GPU	Machine Learning	Avg. Relative Error
fastwalsh	GPU	Signal Processing	Image Diff
fft	CPU	Signal Processing	Avg. Relative Error
fir	ASIC	Signal Processing	Avg. Relative Error
forwardk2j	CPU, ASIC	Robotics	Avg. Relative Error
inversek2j	CPU, GPU, ASIC	Robotics	Avg. Relative Error
jmeint	CPU, GPU	3D Gaming	Miss Rate
jpeg	CPU	Image Processing	Image Diff
kmeans	CPU, ASIC	Machine Learning	Image Diff
kogge-stone	ASIC	Arithmetic Computation	Avg. Relative Error
laplacian	GPU	Image Processing	Image Diff
meanfilter	GPU	Machine Vision	Image Diff
neural network	ASIC	Machine Learning	Avg. Relative Error
newton-raph	GPU	Numerical Analysis	Avg. Relative Error
sobel	CPU, GPU, ASIC	Image Processing	Image Diff
srad	GPU	Medical Imaging	Image Diff
wallace-tree	ASIC	Arithmetic Computation	Avg. Relative Error

H. Application Specific Quality Metric

In AXBENCH, we augment each application with a proper application-specific quality metric. The application-specific quality metric compares the output generated by the precise and approximate version of an application and report the application output quality loss. In total, we introduce three different quality metrics: (1) average relative error, (2) miss rate, and (3) image difference. We use image difference for applications that produce image output. Image difference calculates the average root-mean-square of the pixel differences of the precise and approximate outputs. For applications that produce Boolean outputs, we use miss rate to measure the fraction of correct outputs. For applications that produce numeric outputs, we use average relative error to measure the discrepancy between the original and approximate outputs.

I. Approximable Region Identification

AXBENCH comes with the initial annotations, which mark the approximable region of code. The annotations only provide high-level guidance about where the approximable regions are and not how to approximate those regions. We introduce two criteria to identify the approximable regions in AXBENCH. An approximable region of code in AXBENCH must satisfy these criteria: (1) it must be hot spot; and (2) it must tolerate imprecision in its operations and data;

Hot spot. The intention of approximation techniques is to trade off accuracy for higher gains in performance and energy. Therefore the obvious target for approximation is the region of code which either takes the most execution time or consumes the highest energy of an application. We call this region hot spot. The hot spot of an application contains the highest potential for approximation. Note that, this region may contain function calls, loops, complex control flows, and memory operations.

Tolerance to imprecision. The identified approximable region will undergo approximation during the execution. Therefore, the AXBENCH benchmarks must have some application-level tolerance to imprecision. For example, in jpeg any imprecision on region of code that stores the meta-data in the output image totally corrupts the output image. Whereas, imprecision in region of code that compresses the image (i.e., quantization) has tolerance to imprecision and may only leads to some degree of quality loss in the output image. In AXBENCH, we perform the similar study for each benchmark to identify the region of code which has tolerance to imprecision. The identified regions commensurate with prior work on approximate computing [2], [4], [6], [7].

J. Safety

Recent work on approximate programming languages [7], [6], [8] introduce practical techniques to provide statistical safety guarantees for approximation. However, as mentioned in the previous section, one of the objective inAXBENCH, is to only provide an abstraction above the approximation techniques. This abstraction only provides guidelines about where the potentials for approximation lies and not about how to apply approximation to these regions. Therefore, we do not provide any guarantees about the safety of the AXBENCH applications when they undergo approximation. It is still the responsibility of the users of AXBENCH to provide safety guarantees for their approximation technique. Due to this reason, as we evaluate various approximation techniques in Section III, we use the safety mechanism that is proposed for that approximation technique to provide safety in the AXBENCH's approximable regions.

III. EXPERIMENTAL RESULTS

This section shows how AXBENCH is effective in evaluating previously proposed approximation techniques. For CPU and GPU platforms, we evaluate loop perforation [4] and Neural Processing Unit (NPU) [2], [5]. For dedicated hardware, we evaluate Axilog [3]. We also include an *Ideal accelerator* that magically eliminates approximable regions (i.e., zero delay, energy, and area for approximable regions).

We use `MARSS×86` cycle-accurate simulator for CPU evaluations. The core is modeled after the Nehalem microarchitecture and operates at 3.4 GHz (Table II). We use `McPAT` to measure the energy usage of the benchmarks. We use version 3.2.2 of the `GPGPU-Sim` cycle-level simulator for GPU evaluations. We use a default `GPGPU-Sim`'s configuration that closely models an Nvidia GTX 480 chipset (Table III). We measure the energy usage of GPU workloads using `GPUWattch`. Finally, we use `Synopsys Design Compiler` version G-2012.06 SP5 to synthesize and measure the energy usage of the Verilog benchmarks. We use `TSMC 45-nm` multi-Vt standard cells libraries.[1]

TABLE II: CPU microarchitectural parameters.

Processor: Fetch/Issue Width: 4/5, INT ALUs/FPUs: 6/6, Load/Store Queue: 48-entry/32-entry, ROB Entries: 128, Issue Queue Entries: 36, INT/FP Physical Registers: 256/256, Branch Predictor: Tournament 48 KB, BTB Sets/Ways: 1024/4, RAS Entries: 64, Dependence Predictor: 4096-entry Bloom Filter, ITLB/DTLB Entries: 128/256; **L1 Data Cache** 32 KB, 64B line, 8-Way, Latency: 2 cycles; **L2 Cache** 2 MB, 64B line, 8-Way, Latency: 20 cycles; **Memory Latency:** 200 cycles

TABLE III: GPU microarchitectural parameters.

Processor: 700 MHz, SMs: 16, Warp Size: 32, SIMD Width: 8, Threads per Core: 1024, **L1 Data Cache**: 16KB, 128B line, 4-way, LRU; **Shared Memory**: 48KB, 32 banks; **L2 Unified Cache**: 768KB, 128B line, 8-way, LRU; **Memory**: GDDR5, 924 MHz, FR-FCFS, 4 memory channels, **Bandwidth**: 177.4 GB/sec

A. CPU Platform

Figure 1 compares loop perforation and NPU accelerators for improving speedup and energy efficiency of CPU benchmarks. The maximum quality loss is set to 10%. We restrict the degree of loop perforation and NPU invocations to limit the quality loss to 10%.

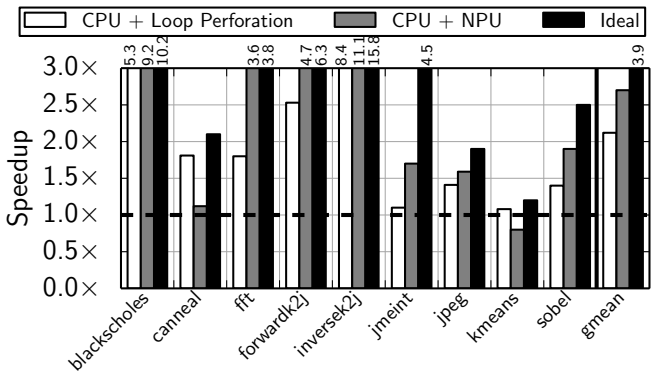
The maximum speedup and energy reduction is registered for inversek2j: loop perforation offers 8.4× speedup and 4.7× energy reduction and NPU offers 11.1× speedup and 21.1× energy reduction. The average speedup (energy reduction) for loop perforation and NPU is 2.1× and 2.7× (1.7× and 3.2×), respectively. Across all benchmarks expect kmeans and canneal, CPU+NPU offers higher speedup and energy reduction as compared to CPU+Loop Perforation. The approximable region in canneal and kmeans consists of few arithmetic operations. Therefore, the communication overhead outweights the potential benefits of NPU acceleration. While NPUs are effective, there is over 30% gap between what they offer and the Ideal, which may be leveraged by other approximation techniques.

B. GPU Platform

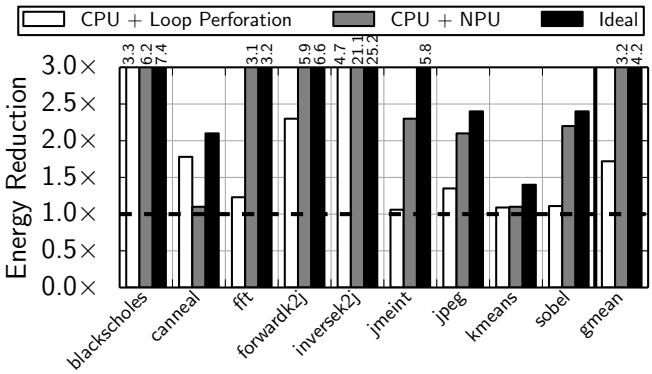
Figure 2 compares loop perforation and NPU accelerators for improving speedup and energy efficiency of GPU benchmarks with 10% maximum quality degradation.

Across all benchmarks, GPU+NPU offers higher speedup and energy reduction as compared to GPU+Loop Perforation.

[1]We use the medium input dataset for all the experiments.

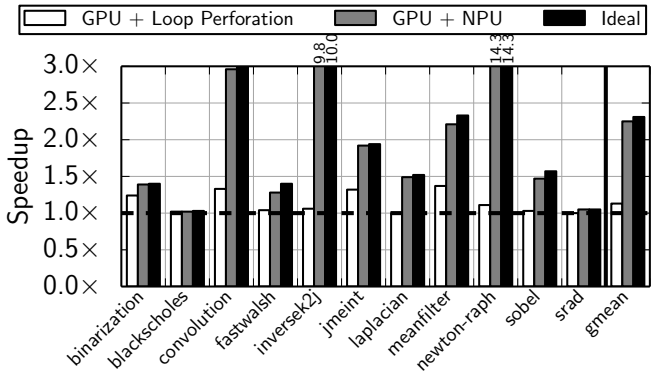


(a) Speedup

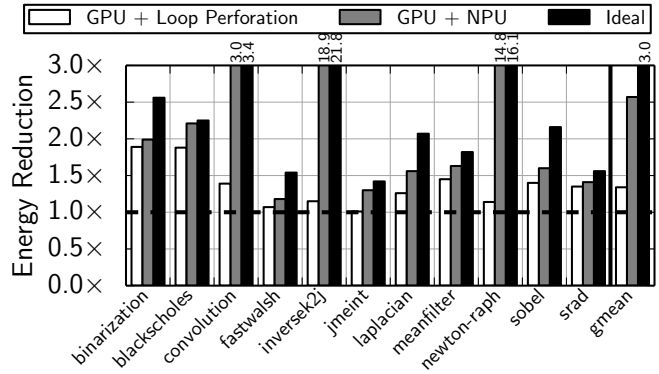


(b) Energy Reduction

Fig. 1: Loop perforation [4] vs. NPU acceleration [2] vs. Ideal on a CPU platform with 10% maximum quality degradation.



(a) Speedup

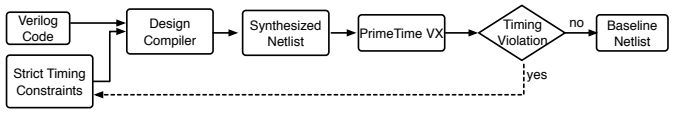


(b) Energy Reduction

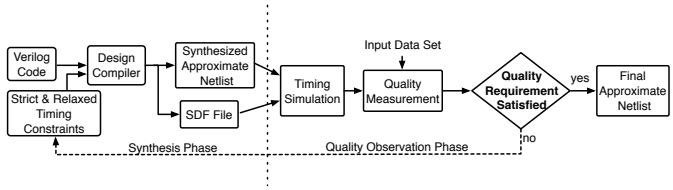
Fig. 2: Loop perforation [4] vs. NPU acceleration [5] vs. Ideal on a GPU platform with 10% maximum quality degradation.

The maximum speedup and energy reduction is registered for newton-raph (14.3×) and inversek2j (18.9×), respectively. The average speedup (energy reduction) for loop perforation and NPU is 1.1× and 2.3× (1.3× and 2.6×), respectively.

Unlike CPU+NPU, which only realizes part of the opportunity, GPU+NPU realizes most of the opportunity of approximate computation. The difference between what NPUs offer and that of an Ideal accelerator is small. As Figure 2 shows, GPU+NPU realizes 97% (85%) of the speedup (energy reduction) opportunity, respectively. The reason is that GPUs execute many threads in parallel to hide data-movement delays. Consequently, massively parallel GPUs augmented with neural accelerators achieve the peak potential of approximate computation.



(a)



(b)

Fig. 3: Synthesis flow for (a) baseline and (b) approximate circuits [3].

C. Dedicated Hardware

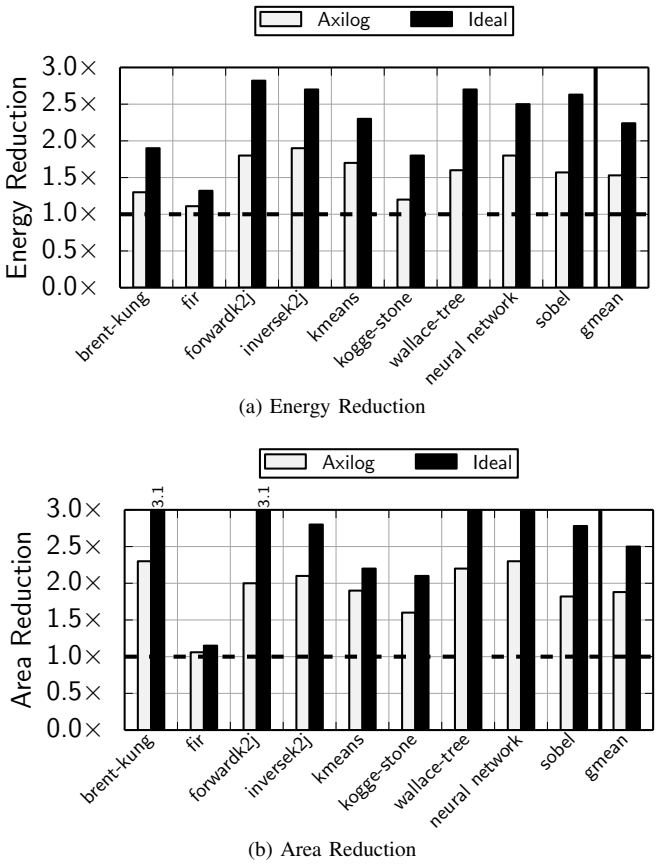
We evaluate Axilog hardware approximation technique [3] using AXBENCH benchmarks. We set the maximum output quality degradation to 10%. We apply Axilog to each benchmark to the extent in which the 10% output quality degradation is preserved. Figure 3 shows hardware synthesis flow for baseline and approximate (Axilog [3]) circuits.

Figure 4 shows the energy and area reduction of applying Axilog to the benchmarks. We do not include a graph for speedup as Axilog does not affect the performance of the benchmarks. Axilog is quite effective at reducing the energy and area needed by the benchmarks. The energy reduction across all benchmarks ranges from 1.1× in fir to 1.9× in inversek2j with a geometric mean of 1.5×. The area reduction ranges from 1.1× in fir to 2.3× in brent-kung with a geometric mean of 1.9×.

Comparing Axilog against Ideal reveals that it only realizes 68% (75%) of the opportunity for reducing the energy (area). While Axilog is effective at reducing the energy and area usage of dedicated hardware, there is still a significant opportunity for innovative approximate computation techniques at the hardware level.

IV. RELATED WORK

Approximate computing. Recent work has explored various approximation techniques across system stack and for different



(a) Energy Reduction

(b) Area Reduction

Fig. 4: Axilog [3] vs. Ideal in reducing (a) energy and (b) area of dedicated hardware design with 10% maximum quality degradation.

frameworks such as CPUs, GPUs, and hardware design that include: (a) programming languages [7], (b) software [4], (c) circuit-level [9], (d) approximate circuit synthesis [3], and (e) neural acceleration [2], [5]. However, prior work does not provide benchmarks for approximate computing. In contrast, this work is an effort to address the needed demand for benchmarking and workload characterization in approximate computing. Distinctively, we introduce AXBENCH, a diverse set of benchmarks for CPUs, GPUs, and hardware design frameworks. AXBENCH may be used by different approximate techniques to study the limits, challenges, and benefits of the techniques.

Benchmarking and workload characterization. There is a growing body of work on benchmarking and workload characterization, which includes: (a) machine learning [10], (b) neural network [11] (c) big data analytics [12], (d) heterogeneous computing [13], (e) scientific computing [14], (f) data mining [15], and (g) computer vision [16]. However, our work contrasts from all the previous work on benchmarking, as we introduce a set of benchmarks that falls into a different category. We introduce AXBENCH, a set of diverse and multi-framework benchmarks for approximate computing. To the best of our knowledge, AXBENCH is the first effort towards providing benchmarks for approximate computing. AXBENCH accelerates the evaluation of new approximation techniques and provides further support for the needed development in this domain.

V. CONCLUSION

As we enter the dark silicon era, it has become more crucial than ever to explore alternative techniques so as to maintain the traditional cadence of performance improvements. One such alternative techniques is approximate computing that promises to deliver significant performance and energy gains at the cost of some quality degradation. As approximate computing gains popularity in different computing platforms, it is important to have a diverse, representative, and multi-platform set of benchmarks. A benchmark suite with these features facilitates fair evaluation of approximation techniques and speeds up progress in the approximate computing domain. This work expounds AXBENCH, a benchmark suite for approximate computing across the system stack. AXBENCH includes benchmarks for CPUs, GPUs, and hardware design design. Benchmarking is of foundational importance to an emerging research direction and AXBENCH provides the initial groundwork.

VI. ACKNOWLEDGMENTS

We thank Jongse Park, Bradley Thwaites, Anandhavel Nagendrakumar, Sindhuja Sethuraman, Abbas Rahimi, and other members of the Alternative Computing Technologies (ACT) Lab for their feedback and contributions. This work was in part supported by a Qualcomm Innovation Fellowship, NSF award CCF #1553192, Semiconductor Research Corporation contract #2014-EP-2577, and gifts from Google and Microsoft.

REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," in *MICRO*, 2012.

[3] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan, "Axilog: Language support for approximate hardware design," in *DATE*, 2015.

[4] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *FSE*, 2011.

[5] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural Acceleration for GPU Throughput Processors," in *MICRO*, 2015.

[6] 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 *PLDI*, 2011.

[7] J. Park, H. Esmaeilzadeh, X. Zhang, M. Naik, and W. Harris, "Flexjava: Language support for safe and modular approximate programming," in *23rd Symposium on Foundations of Software Engineering*, 2015.

[8] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *OOPSLA*, 2013.

[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ASPLOS*, 2012.

[10] O. D. Alcântara, Á. R. Pereira Junior, H. M. d. Almeida, M. A. Gonçalves, C. Middleton, and R. B. Yates, "Wcl2r: a benchmark collection for learning to rank research with clickthrough data," 2010.

[11] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam, "Benchnn: On the broad potential application scope of hardware neural network accelerators?" in *IISWC*, Nov. 2012.

[12] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "Bigdatabench: A big data benchmark suite from internet services," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, Feb 2014, pp. 488–499.

[13] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[14] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, 2012.

[15] R. Narayanan, B. Özisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A benchmark suite for data mining workloads," in *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 2006, pp. 182–188.

[16] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "Sd-vbs: The san diego vision benchmark suite," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 55–64.