AxBench: A Benchmark Suite for Approximate Computing Across the System Stack

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

ABSTRACT

As the end of Dennard scaling looms, both the semiconductor industry and the research community are exploring for innovative solutions that allow energy efficiency and performance to continue to scale. Approximation computing has become one of the viable techniques to perpetuate the historical improvements in the computing landscape. As approximate computing attracts more attention in the community, having a general, diverse, and representative set of benchmarks to evaluate different approximation techniques becomes necessary.

In this paper, we develop and introduce AxBench, a general, diverse and representative multi-framework set of benchmarks for CPUs, GPUs, and hardware design with the total number of 29 benchmarks. We judiciously select and develop each benchmark to cover a diverse set of domains such as machine learning, scientific computation, signal processing, image processing, robotics, and compression. AxBench comes with the necessary annotations to mark the approximable region of code and the application-specific quality metric to assess the output quality of each application. AxBench with these set of annotations facilitate the evaluation of different approximation techniques.

To demonstrate its effectiveness, we evaluate three previously proposed approximation techniques using AxBench benchmarks: loop perforation [1] and neural processing units (NPUs) [2–4] on CPUs and GPUs, and Axilog [5] on dedicated hardware. We find that (1) NPUs offer higher performance and energy efficiency as compared to loop perforation on both CPUs and GPUs, (2) while NPUs provide considerable efficiency gains on CPUs, there still remains significant opportunity to be explored by other approximation techniques, (3) Unlike on CPUs, NPUs offer full benefits of approximate computations on GPUs, and (4) considerable opportunity remains to be explored by innovative approximate computation techniques at the hardware level after applying Axilog.

1 Introduction

As the end of Dennard scaling and Moore's law advances loom, the computing landscape confronts the diminishing performance and energy improvements from the traditional scaling paradigm [6–8]. This evolution drives both the industry and the research community to explore viable solutions and techniques to maintain the traditional scaling of performance and energy efficiency. Approximate computing is one of the promising approaches for achieving significant efficiency gains at the cost of some output quality degradation for applications that can tolerate inexactness in their output.

A growing swath of studies have proposed different approximation languages and techniques including software languages EnerJ [9] and Rely [10], hardware language Axilog [5], circuit level techniques [11–26], microarchitecture techniques [27, 28], algorithmic techniques [29, 30] and approximate accelerators [2–4, 31]. As approximate computing gains popularity, it becomes important to have a diverse and representative set of benchmarks for a fair evaluation of approximation techniques. While a bad set of benchmarks makes progress problematic, a good set of benchmarks can help us as a community to rapidly advance the field [32].

A benchmark suite for approximate computing has to have several features. 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 different applications.

Moreover, approximate computing can be applied to various levels of computing stack and through different techniques. Approximate computing is applicable to both hardware and software. At both levels various techniques may be used for approximation. At the hardware level, a dedicated approximate hardware may perform the operations [2–4, 31, 33–35] or an imprecise processor may run the program [12, 34, 36], among other possibilities [5, 13–26, 37, 38]. Likewise, there are many possibilities at the software level [1, 29, 30, 39–41]. A good benchmark suite for approximate computation should be useful for evaluation of all 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.

Finally, approximation not only applies to information processing, but also can be applied to information communication and information retrieval. While many approximation techniques target processing units [2–4, 12, 34, 36], the communication and storage medium are also amenable to approximation [27, 42–46]. This means that a good benchmark suite for approximate computing should be rich enough to be useful for evaluation of approximate communication and storage.

This paper introduces AxBench, a diverse and representative multi-framework set of benchmarks for evaluating approximate computing research in CPUs, GPUs and hardware design. We discuss why AxBench benchmarks have all the necessary features of a good benchmark suite for approximate computing. AxBench covers diverse application domains such as machine learning, robotics, arithmetic computation, multimedia, and signal processing. AxBench enables researchers in the approximate computing to study, evaluate, and compare the state-of-the-art approximation techniques on a diverse set of benchmarks in a straightforward manner.

We perform a detailed characterization of AxBench benchmarks on CPUs, GPUs, and dedicated hardware. The results show that approximable regions of the benchmarks, on average, constitute 74.9% of runtime and 81.8% of energy usage of the whole applications when they run on a CPU. On a GPU, approximable regions constitute 53.4% of runtime and 56.0% of energy usage of the applications. We use an approximation synthesis [5] to gain insights about the potential benefits of using approximation in the hardware design. The results demonstrate, on average, approximate parts constitute 92.4% of runtime, 69.4% of energy usage, and 70.1% of the area of the whole dedicated hardware. These results clearly demonstrate that these benchmarks, which are taken from various domains, are amendable to approximation.

We also evaluate three previously proposed approximate computation techniques using AxBench benchmarks. We apply Loop Perforation [1] and Neural Processing Units (NPUs) [2–4] to CPU and GPU, and Axilog [5] 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 mainly because NPUs do nothing for data misses. On GPUs, however, NPUs leverage all the potentials and leave very little opportunity for other approximation techniques except on workloads that saturate the off-chip bandwidth. Data misses are not a performance bottleneck for GPU+NPU, as massively-parallel GPU can effectively hide data miss latencies. Finally, we find that Axilog is effective at improving efficiency of dedicated hardware but still significant opportunity exists for other approximation techniques to be explored.

The contributions of this work are as follows:

1. We provide a diverse and representative set of **multi-framework** benchmarks for approximate computation (AxBench). AxBench comes in three categories tailored for studying approximate computing on CPUs, GPUs, and dedicated hardware. It includes 9 benchmarks for CPUs, 11 for GPUs and 9 for hardware design.

2. All the benchmarks come with the **initial annotations**, which marks the approximable regions of the code. The annotations *only* marks *where* the approximation is beneficial without specifying how to approximate the region. The annotations ease the evaluation of different approximation techniques with minor changes in the code.

3. We introduce **application-specific quality metrics** for each of the benchmarks. The introduced quality metric for each benchmark helps to understand *how* the exploited approximation technique affects the output quality. The user can easily alter the degree of approximation or the applied approximation technique and compare their effects on the output quality of the application.

Table 1: The evaluated CPU benchmarks, characterization of each approximable region, and the quality metric.

Name	Domain	# of Function Calls	# of Loops	# of ifs / elses	# of x86-64 Insts.	Quality Metric
blackscholes	Financial Analysis	5	0	5	309	Avg. Relative Error
canneal	Optimization	6	2	6	378	Avg. Relative Error
fft	Singal Processing	2	0	0	34	Avg. Relative Error
forwardk2j	Robotics	2	0	0	65	Avg. Relative Error
inversek2j	Robotics	4	0	0	100	Avg. Relative Error
jmeint	3D Gaming	32	0	23	1,079	Miss Rate
jpeg	Compression	3	4	0	1,257	Image Diff
kmeans	Machine Learning	1	0	0	26	Image Diff
sobel	Image Processing	3	2	1	88	Image Diff

Table 2: The evaluated GPU benchmarks, characterization of each approximable region, and the quality metric.

Name	Domain	# of Function Calls	# of Loops	# of ifs / elses	# of PTX Insts.	Quality Metric
binarization	Image Processing	1	0	1	27	Image Diff
blackscholes	Financial Analysis	2	0	0	96	Avg. Relative Error
convolution	Machine Learning	0	2	2	886	Avg. Relative Error
fastwalsh	Signal Processing	0	0	0	50	Avg. Relative Error
inversek2j	Robotics	0	3	5	132	Avg. Relative Error
jmeint	3D Gaming	4	0	37	2,250	Miss Rate
laplacian	Image Processing	0	2	1	51	Image Diff
meanfilter	Machine Vision	0	2	1	35	Image Diff
newton-raph	Numerical Analysis	2	2	1	44	Avg. Relative Error
sobel	Image Processing	0	2	1	86	Image Diff
srad	Medical Imaging	0	0	0	110	Image Diff

Table 3: The evaluated ASIC benchmarks, characterization of each approximable region, and the quality metric.

Name	Domain	# of Lines	Quality Metric
brent-kung (32-bit)	Arithmetic Computation	352	Avg. Relative Error
fir (8-bit)	Signal Processing	113	Avg. Relative Error
forwardk2j	Robotics	18,282	Avg. Relative Error
inversek2j	Robotics	22,407	Avg. Relative Error
kmeans	Machine Learning	10,985	Image Diff
kogge-stone (32-bit)	Arithmetic Computation	353	Avg. Relative Error
wallace tree (32-bit)	Arithmetic Computation	13,928	Avg. Relative Error
neural network	Machine Learning	21,053	Image Diff
sobel	Image Processing	143	Image Diff

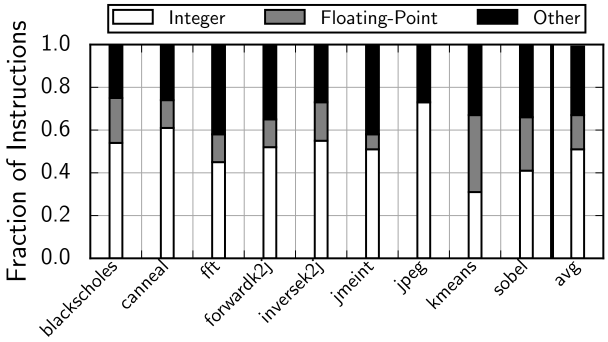


Figure 1: The instruction classification for CPU platform.

4. We evaluate previously proposed approximation techniques using AxBench and find that (1) While NPUs provide considerable efficiency gains on CPUs, there still remains significant opportunity to be explored by other approximation techniques, mainly because NPUs do nothing for data accesses (2) Unlike on CPUs, NPUs offer full benefits of approximate computations on GPUs when off-chip bandwidth is not saturated.

2 Benchmarks

One of the goals of AxBench is to provide a diverse set of applications to further facilitate the research and development in approximate computing. It consists of 29 applications from different domains including machine learning, computer vision, big data analytics, 3D gaming, and robotics. We first introduce the benchmarks and some of their characteristics, and then discuss the benchmark specific quality metric.

2.1 Common Benchmarks

AxBench provides a set of C/C++ benchmarks for execution on CPUs, a set of CUDA benchmarks for execution on GPUs, and a set of Verilog benchmarks for hardware design. Some algorithms are amenable for execution on all platforms. For these algorithms, AxBench provides the C/C++ implementation for execution on CPUs, the CUDA implementation for execution on GPUs, and the Verilog implementation for hardware design. In this part, we briefly introduce these benchmarks.

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 is widely used in image processing and computer vision applications, particularly within edge detection algorithms. The Sobel application takes an RGB image as the input and produces a grayscale image in which the edges are emphasized.

2.2 Common CPU and GPU Benchmarks

For some algorithms, AxBench provides both the C/C++ implementation for execution on CPUs and the CUDA implementation for execution on GPUs. We briefly introduce these benchmarks here.

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 three-dimensional space and the output is a Boolean value which indicates whether the two triangles intersect or not.

2.3 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. To evaluate this benchmark, we use an RGB image as the input. The output is an image that is clustered in different color regions.

2.4 CPU Specific Benchmarks

Table 1 shows the complete list of all CPU benchmarks. In addition to the six benchmarks that are already introduced, there are three dedicated CPU benchmarks as listed in the table. The table also indicates the domain in which these benchmarks are taken from and some characteristics about the benchmarks. In what follows, we briefly introduce the three CPU specific benchmarks.

Canneal is an optimization algorithm which is used to minimize the routing cost of chip design. Canneal employs the simulated annealing (SA) [47] 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. We use a set of sequences of signals as input dataset. The output is the representation of the corresponding signals 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.

Figure 1 shows the instructions breakdown for all CPU benchmarks. On average, 51% of the instructions are integers, 16% of the instructions are floating-points, and the rest are other types of instructions (e.g., load, store, conditional, etc.). The figure shows that the CPU benchmarks have diversity in their instruction breakdown.

2.5 GPU Specific Benchmarks

Table 2 shows the complete list of all GPU benchmarks, the domain that the benchmarks are taken from and some characteristics about the applications. In addition to the four GPU benchmarks that are already introduced, there are seven GPU specific benchmarks in the table. In this part, we introduce the 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 an image processing workload and 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. Similar to other image processing benchmarks, we use an image as the input. The output is the transformed form of the input image.

Laplacian is an image processing application. It is mainly used in image processing for edge detection. The output image is a

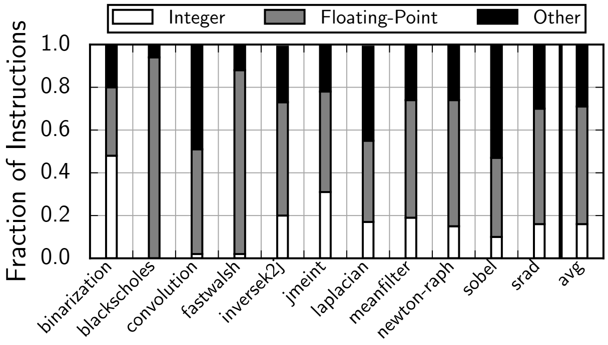


Figure 2: The instruction classification for GPU platform.

grayscale image in which all the edges are emphasized.

Meanfilter is an image processing application. It 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 3x3 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.

Figure 2 shows the instructions breakdown for all GPU benchmarks. On average, 16% of the instructions are integers, 55% of the instructions are floating-points, and the rest are other types of instructions (e.g., load, store, conditional, etc.). As compared to CPU benchmarks, GPU benchmarks have more floating point instructions. Even for common CPU and CPU benchmarks, the CPU benchmarks have more floating-point instructions. The reason is that we implemented many floating-point operations like sin and cos using lookup tables for CPU benchmarks, as processors usually do not have hardware support for such operations. However, due to hardware support in GPUs, we used the floating-point operation for GPUs benchmarks.

2.6 Hardware-Design Specific Benchmarks

Table 3 shows the complete list of all hardware-design benchmarks. In addition to the four benchmarks that are already introduced, there are five dedicated hardware-design benchmarks as listed in Table 3. The table also indicates the domain in which these benchmarks are taken from and some characteristics about the benchmarks. In what follows, we briefly introduce the five 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. To evaluate this benchmark, 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.

Wallace-Tree is an efficient design for multiplying two integer values. Similar to other arithmetic computation benchmarks, we use random 32-bit integer values as input. The output is the product of the corresponding numbers.

Neural-Network is an implementation of a feedforward neural network that approximate the Sobel filter. Such artificial 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.

2.7 Application Specific Quality Metric

Each approximation technique may introduce different levels of quality loss in the applications' output. Therefore, a good benchmark suite for approximation has to include a quality metric to evaluate the applications' output quality loss as they undergoes approximation. However, since the nature of applications' output varies from one application to another, it is required to have a quality metric that takes into account the type of the application's output. For example, the output of sobel is an image, but the output of jmeint is a Boolean. Therefore, it is necessary to have an application specific quality metric to effectively assess the quality loss of each application's output.

To make AxBench useful, we augment each application with an application specific quality metric. In total, we introduce three different quality metrics: (1) average relative error, (2) miss rate, and (3) image difference. We use image difference for all applications that produce image output, including binarization, jpeg, kmeans, laplacian, meanfilter, neural network, sobel, and srad. Image difference calculates the average root-mean-square of the pixel differences of the precise and approximated images. Jmeint algorithm checks whether two 3D triangles intersects and produce a Boolean value (True if the triangles intersect, and false otherwise). Therefore, we use miss rate to measure the number of misclassified triangles. For all the other applications (e.g. blackscholes, canneal, fastwalsh, fft, inversek2j) that produce numeric outputs, we use average relative error to measure the discrepancy between the original and approximate outputs.

3 Methodology

3.1 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. There-

Table 4: Major CPU microarchitectural parameters for the core, caches, memory.

Core		Caches and Memory	
Clock Frequency	3.4 GHz	L1 Cache Size	32 KB instruction, 32 KB data
Architecture	x86-64	L1 Line Width	64 bytes
Fetch/Issue Width	4/6	L1 Associativity	8
INT ALUs/FPUs	3/2	L1 Hit Latency	3 cycles
Load/Store FUs	2/2	ITLB/DTLB Entries	128/256
ROB Entries	96	L2 Cache Size	2 MB
Issue Queue Entries	32	L2 Line Width	64 bytes
INT/FP Physical Registers	256/256	L2 Associativity	8
Branch Predictor	Tournament, 48 KB	L2 Hit Latency	12 cycles
BTB Sets/Ways	1024/4	Memory Latency	104 cycles (30 ns)
Load/Store Queue Entries	48/48	On-Chip Bandwidth	33.77 GB/sec
Dependence Predictor	4096-entry Bloom Filter	Off-Chip Bandwidth	8 GB/sec

fore 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. Recent approximate programming work [1–4, 9] has shown that identifying these regions of code with tolerance to imprecision is not difficult. 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 [1, 2, 9, 48].

3.2 Safety

Recent work on approximate programming languages [9, 10, 48] introduce practical techniques to provide statistical safety guarantees for approximation. However, as mentioned in the previous section, one of the objective in AxBench, 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, in Section 4.2 in which we evaluate AxBench with prior approximation techniques, we use the similar safety mechanism techniques as what they proposed to provide safety in the AxBench's approximable regions.

3.3 Experimental Setup for CPU

Cycle-accurate simulation and energy modeling. We use the MARSSx86 cycle-accurate x86-64 simulator [50] to characterize AxBench benchmarks on a CPU platform. Table 4 summarizes the major microarchitectural parameters of the core, caches, and the memory subsystem. The core is modeled after the Nehalem microarchitecture and operates at 3.4GHz. We made several minor changes in the simulator to enable the simulation and characterization of the approximable and non-approximable regions. We use C assembly inlining in the source code to identify the beginning and the end of the approximable region. We compile all the benchmarks using GCC/G++ version 4.8.4 with the -O3 flag to enable aggressive compiler optimizations. We simulate all the benchmarks to completion to capture the distinct phases of each workload.

We use McPAT [51] to measure the energy consumption of the benchmarks. MARSSx86 generates an event log during the cycle-accurate simulation of a program. We pass this event log to a modified version of the McPAT. At the end, McPAT provides the energy consumption of the approximable and non-approximable regions.

Table 5: Major GPU microarchitectural parameters for the streaming multiprocess (SM), shader processor (SP), caches, and memory.

System Overview	15 SMs, 32 thread/warps 6 memory channel
Shader Core Config	1.4 GHz, GTO scheduler [49] 2 scheduler per SM
Resources per SM	48 warps/SM, 32768 registers
L1 Instruction Cache	2KB, 128B line, 4-way, LRU
L1 Data Cache	16KB, 128B line, 4-way, LRU
L1 Constant Cache	8KB, 64B line, 2-way, LRU
L1 Texture Cache	12KB, 128B line, 24-way, LRU
Shared Memory	48KB, 32 banks
Interconnect	1 crossbar/direction (15 SMs, 6 MCs) 1.4 GHz
L2 Shared Cache	768KB, 128B line, 16-way, LRU
Memory	924 MHz, 6 GDDR5 Memory Controllers FR-FCFS scheduling, 16 banks/SM
On-Chip Bandwidth	443.5 GB/sec
Off-Chip Bandwidth	177.4 GB/sec

3.4 Experimental Setup for GPU

Cycle-accurate simulation and energy modeling. We use version 3.2.2 of the GPGPU-Sim cycle-accurate simulator [52] to characterize AxBench workloads on a GPU platform. We simulate all the benchmarks with a default GPGPU-Sim's configuration that closely models an Nvidia GTX 480 chipset. Table 5 summarizes the major microarchitectural parameters of the simulated system. We made several changes in the simulator to simulate and characterize the approximable and non-approximable regions. We use the PTX[1] inlining in the source code to iden-

[1] Parallel Thread Execution is a pseudo-assembly language in Nvidia's CUDA programming model

tify the beginning and the end of the approximable region. We compile all the benchmarks using NVCC version 4.2 with -O3 flag to enable aggressive compiler optimizations. Furthermore, we accordingly optimize the number of thread blocks and the number of threads per block of each kernel for our simulated hardware. We simulate all the benchmarks to completion to capture the distinct phases of each workload.

We measure the energy consumption of GPU workloads by using GPGPU-Wattch [53], which is integrated with GPGPU-Sim.

3.5 Experimental Setup for ASIC

Gate-level simulation and energy modeling. We use Synopsys Design Compiler version G-2012.06SP5 to synthesize and measure the energy consumption of the Verilog benchmarks. We use TSMC 45-nm multi-V_t standard cells libraries for the synthesis. We report the results for the slowest PVT corner (SS, 0.81 V, 0°C). We use Cadence NC-Verilog version 11.10-s062 for timing simulation with SDF back annotations extracted from the output of the synthesis.

3.6 Prior Approximation Techniques

To show the advantage of AxBench, we evaluate some of the prior approximation techniques with the introduced benchmarks. For CPU and GPU benchmarks, we study the benefits of using neural processing units (NPUs) [2–4] and loop perforation [1]. For dedicated hardware benchmarks, we use the proposed approximation synthesis technique in Axilog [5].

Neural processing units (NPUs). Hadi Esmaeilzadeh et al. [2] introduced an automated workflow for neural transformations. Furthermore, they designed a tightly-coupled neural acceleration for general-purpose approximate programs. First, the automated framework observes the annotated region of code that transforms a region of code to a neural representation. This transformation consists only of simple arithmetic operations such as addition and multiplication and sigmoid. Then, the compiler annotates the source code which allow the CPU to offload the annotated region(s) to an efficient implementation of feedforward neural networks.

Loop perforation. This software-level technique provides higher efficiency by reducing the amount of required computation to produce the output. The Loop perforation technique first identifies the loops whose perforation still leads to acceptable output. Then, they transform these loops to execute a subset of their iterations.

Axilog. Amir Yazdanbakhsh et al. [5] introduces a set of language annotations that enables the hardware designers to incorporate the approximation in hardware design. They also proposed an approximate synthesis workflow in which they relax the timing constraints on some paths. This allows the synthesis tools to use slower gates (less leaky and smaller) on these paths which lead to reduction in both area and energy consumption.

4 Experimental Results

We first characterize AxBench's benchmarks on CPU, GPU and dedicated hardware to demonstrate their usefulness for approximate computing. We then use the benchmarks to evaluate three previously proposed approximation techniques, namely loop perforation [1], neural processing units (NPU) [2–4], and Axilog [5].

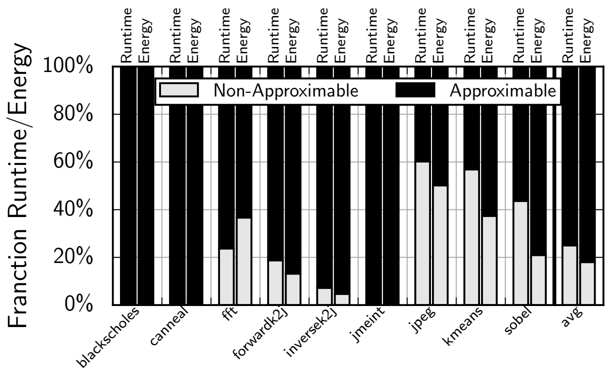


Figure 3: Total application runtime and energy breakdown between non-approximable and approximable region in CPU platform.

4.1 Benchmark Characterization

4.1.1 CPU Platform

Runtime and energy breakdown. Figure 3 shows the runtime and energy breakdown of approximable and non-approximable parts of applications when applications execute on a CPU. Some applications such as blackscholes, canneal, and jmeint are completely approximable. Among the rest, while many applications spend more than 70% of their runtime and energy in approximable regions (e.g., inversek2j), the number goes down to less that 40% in some applications (e.g., jpeg). On average, applications spend 74.9% of their runtime and 81.8% of their energy in approximable regions. While the time-energy breakdown differs from one application to another, this experiment clearly shows that there is a strong opportunity for approximate computing across a large number of applications from scientific computing, signal processing, image processing, etc.

L1-I and L1-D misses per kilo instructions (MPKI). Figure 4 and 5 present the number of L1-I and L1-D misses per kilo instructions of the approximable regions of our benchmarks, respectively. For some applications (e.g., blackscholes) the instruction footprint is relatively small and fits in the L1-I of the processor. For these benchmarks, the number of L1-I misses per kilo instructions is almost zero. The rest of the benchmarks have instruction footprint that is larger than the L1-I cache, and consequently, suffer from L1-I misses (e.g., jmeint). On the other hand, the data working sets of AxBench benchmarks are large, exceeding the capacity of L1-D caches, and as a result, the number of L1-D misses per kilo instructions of many applications is relatively large. Across all benchmarks, the L1-D MPKI ranges from 0.5 in blackscholes to 27.6 in canneal. These graphs clearly show that instruction and data delivery is a bottleneck across many applications. Any approximation technique that can unblock the bottleneck has a potential to significantly improve performance and energy efficiency.

On-chip bandwidth utilization Figure 6 shows the on-chip bandwidth utilization across all the benchmarks. The maximum bandwidth utilization is registered for fft and is less than 10%. The rest of the benchmarks have even lower bandwidth utilization. While many applications suffer from large instructions and data misses, the on-chip bandwidth is almost underutilized. This is due to (1) low performance per core due to high number of misses and (2) few number of cores (in this case: one) per processor. The results suggest that approximate computation techniques

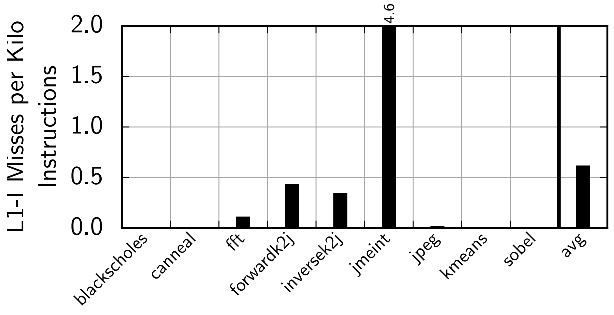


Figure 4: Approximable region L1 instruction cache misses per kilo instructions in CPU platform. L1 instruction cache size is 32 KBs.

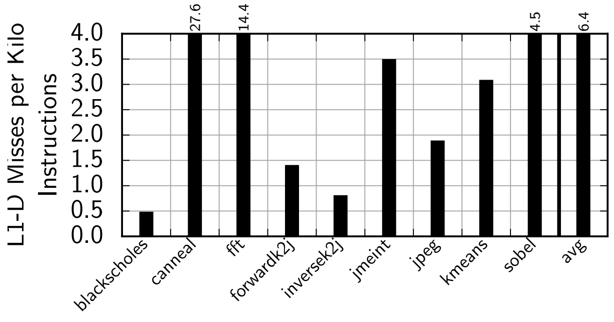


Figure 5: Approximable region L1 data cache misses per kilo instructions in CPU platform. L1 data cache size is 32 KBs.

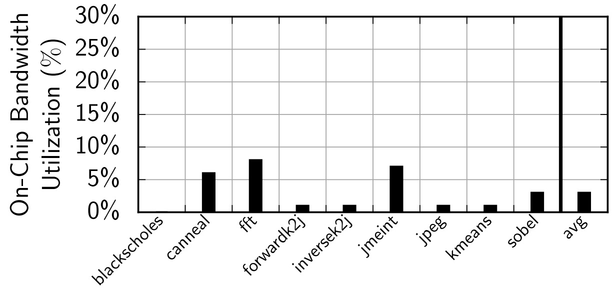


Figure 6: On-chip bandwidth utilization in CPU platform. The baseline on-chip bandwidth is 33.77 GB/sec.

that increase the performance of benchmarks are unlikely to hit the on-chip bandwidth bottleneck.

LLC misses per kilo instructions (MPKI). Figure 7 shows the number of LLC misses per kilo instructions of the approximable regions of AxBench's benchmarks. We find that many benchmarks in our suite have data working sets larger than the LLC capacity (i.e., 2 MBs) and consequently observe many LLC misses (the instruction footprint of the benchmarks fits in the LLC, so the LLC misses are mainly due to data accesses). The LLC MPKI ranges from almost 0 in blackscholes to 15 in canneal. Large number of LLC misses per kilo instructions results in significant performance and energy loss. Consequently, the results suggest usefulness of any approximate computing technique that reduces the LLC pressure.

Off-chip bandwidth utilization. Figure 8 shows the off-chip utilization of our CPU benchmarks. Just like on-chip bandwidth utilization, the off-chip bandwidth of almost all benchmarks is underutilized. This is due to (1) low performance per core due to large number of misses and (2) few number of cores in the processor. The only exception is canneal that experiences large number of LLC misses (almost every access to LLC is a miss). The

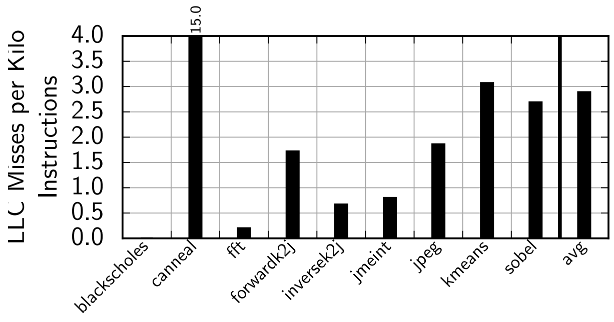


Figure 7: Approximable region LLC cache misses per kilo instructions in CPU platform. LLC cache size is 2 MBs.

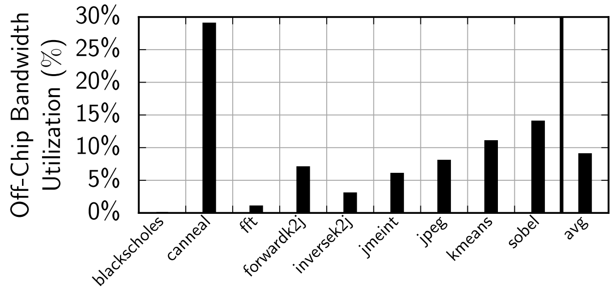


Figure 8: Off-chip bandwidth utilization in CPU platform. The baseline off-chip bandwidth is 8 GB/sec.

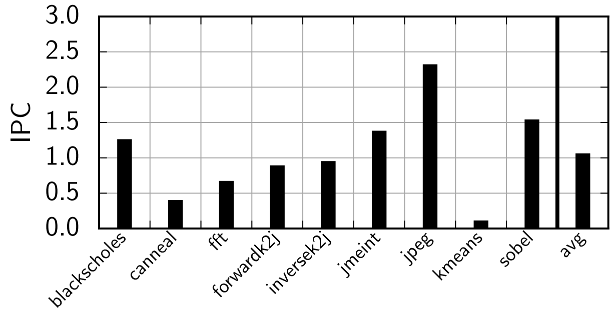


Figure 9: The approximable region instruction per cycle (IPC) in CPU platform.

results show that an approximate computation that improves the performance of almost all of these benchmarks is unlikely to hit the off-chip bandwidth bottleneck. For canneal that suffers from large number of LLC misses, a successful approximate computation technique should reduce the number of off-chip accesses.

Instruction per cycle (IPC). Figure 9 shows the average number of instructions committed per cycle when approximable parts of the applications are executing on the CPU. While the CPU is capable to execute up to 3 instructions per cycle, due to inefficiencies in instructions and data delivery, the average number of instructions committed per cycle for majority of the benchmarks do not reach 1.5. In many applications, the average IPC is less than 1 (e.g., canneal and fft). Any approximate computing that unblocks the bottleneck on instructions and data delivery has a significant potential to improve performance.

4.1.2 GPU Platform

Runtime and energy breakdown. Figure 10 shows the time and energy breakdown of approximable and non-approximable parts of applications when applications run on a GPU. On average, applications spend 53.4% of their runtime and 56.0% of their energy usage in approximable regions. Some applications

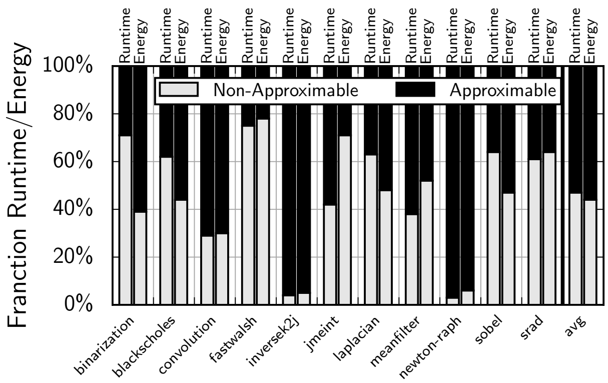


Figure 10: Total application runtime and energy breakdown between non-approximable and approximable region in GPU platform.

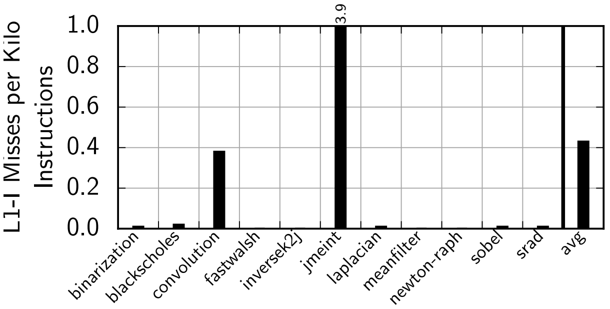


Figure 11: Approximable region application L1 instruction cache misses per kilo instructions in GPU platform. L1 instruction cache size is 2 KBs.

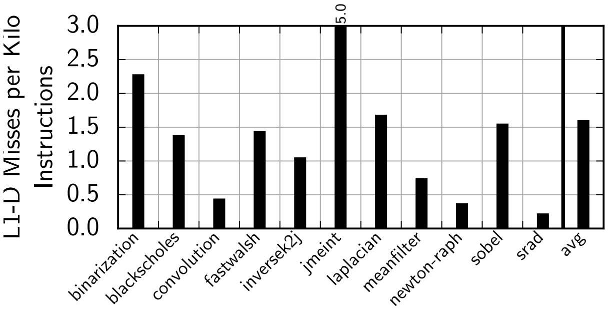


Figure 12: Approximable region application L1 data cache misses per kilo instructions in GPU platform. L1 data cache size is 16 KBs.

such as inversek2j and newton-raph spend more than 90% of their runtime and energy in approximable regions. This experiment clearly shows that approximate computing has a strong opportunity to improve execution time and energy efficiency of GPUs across a large number of domains such as signal processing, scientific computing, and multimedia.

L1-I and L1-D misses per kilo instructions (MPKI). Figure 11 and 12 present the number of L1-I and L1-D misses per kilo instructions of the approximable regions of our benchmarks. We find that the instruction footprint of GPU benchmarks tend to be smaller than that of CPU benchmarks. Consequently, the number of L1-I misses per kilo instructions of GPU benchmarks is smaller than that of CPU benchmarks. On the other hand, the number of L1-D misses per kilo instructions for GPU benchmarks is significant. Across all benchmarks, the number of L1-D misses per kilo instructions ranges from less than 0.25 in srad to 5 in jmeint.

On-chip bandwidth utilization. Figure 13 shows the on-chip

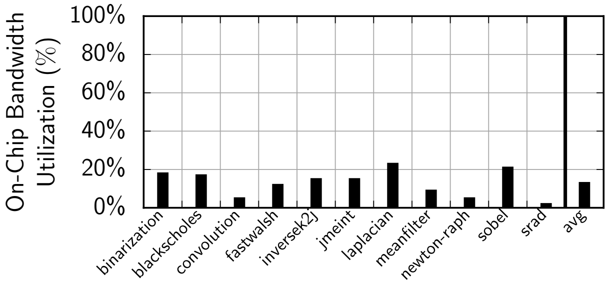


Figure 13: Off-chip bandwidth utilization in GPU platform. The baseline on-chip bandwidth is 443.5 GB/s.

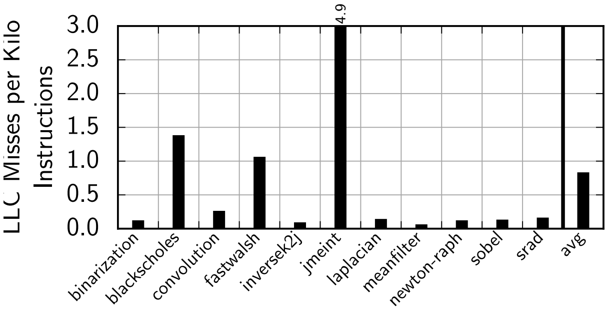


Figure 14: Total application L2 data cache misses per kilo instructions in GPU platform. L2 data cache size is 786 K-Byte.

bandwidth utilization of the GPU across all benchmarks. Similar to the results for the CPU, on-chip bandwidth is underutilized for all the benchmarks. The on-chip bandwidth utilization is under 25% for all the benchmarks. As compared to a CPU, a GPU benefits from thread-level parallelism and executes many threads in parallel (our CPU executes one thread while each SM can execute 64 threads in parallel and there are 15 SMs in the GPU). The underutilization of on-chip bandwidth is due to (1) GPU cores (SIMD lanes) are simpler and less capable than CPU cores and (2) GPU has higher on-chip bandwidth as compared to the CPU: our CPU has 33.77 GB/sec of on-chip bandwidth while the GPU benefits from 443.5 GB/sec (a factor of 13× higher bandwidth). These results suggest that an approximation technique that improves the performance of GPU workloads is unlikely to hit the on-chip bandwidth bottleneck.

LLC misses per kilo instructions (MPKI). Figure 14 shows the number of LLC misses per kilo instructions of the approximable regions of AxBench's benchmarks. We find that many benchmarks in our suite have data working sets larger than the LLC capacity (i.e., 786 KBs), and consequently, observe large number of LLC misses. The number of LLC misses per kilo instructions ranges from 0.05 in meanfilter to 4.9 in jmeint.

Off-chip bandwidth utilization Figure 15 shows the off-chip bandwidth utilization of GPU benchmarks. Some of the GPU benchmarks have high on-chip bandwidth utilization (more than 40%). The high utilization is because GPUs benefit from executing many threads in parallel. Across the benchmarks, the off-chip bandwidth utilization ranges from less than 5% in meanfilter and newton-raph to about 60% in blackscholes and srad. The results suggest that off-chip bandwidth is a bottleneck for some GPU benchmarks and a successful approximation technique should reduce the off-chip traffic to be useful for these applications.

Instruction per cycle (IPC). Figure 16 shows the average number of instructions committed per cycle when approximable parts

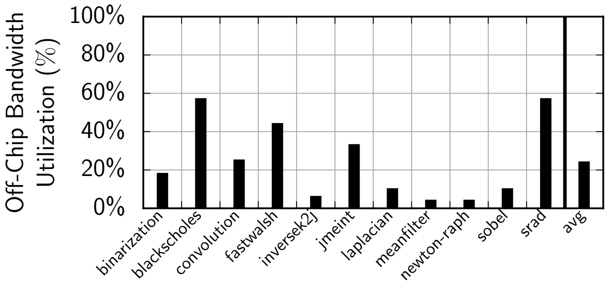


Figure 15: Off-chip bandwidth utilization in GPU platform. The baseline off-chip bandwidth is 177.4 GB/s.

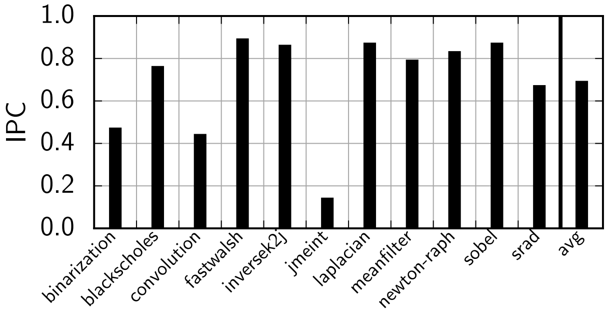


Figure 16: Approximable region instruction per cycle (IPC) for each thread in GPU platform. The maximum IPC for each thread is one.

of the applications are executing on the GPU. Each GPU thread can execute 1 instruction per cycle. Due to inefficiencies in instructions and data delivery, the average number of instructions committed per cycle is less than 1. The IPC ranges from less than 0.2 in jmeint to near 0.9 in fastwalsh. While the IPC of each thread is less than 1, as a GPU's SM executes 64 threads in parallel, GPUs execute significantly higher number instructions per cycle as compared to CPUs. Nevertheless, there are opportunities, as manifested by Figure 10, for approximate computing techniques to improve the performance and energy efficiency of GPU applications.

4.1.3 Dedicated Hardware

AxBench provides a set of Verilog benchmarks for approximate hardware design. In this section, we characterize these benchmarks to show their effectiveness for approximate computing.

Runtime, energy, and area breakdown. Figure 17 shows the time, energy, and area breakdown of approximable and non-approximable parts of the benchmarks when we synthesized them using Synopsis Design Compiler (DC). On average and across all benchmarks, 92.4% of the time, 69.4% of the energy, and 70.1% of the area go to approximable parts. Some applications such as forwardk2j, inversek2j, kmeans, and neural network spend more than 90% of their runtime, energy, and area in approximable parts. All benchmarks spend 90% of their runtime on approximable parts. For energy and area, some benchmarks spend less than 50% on approximable parts. However, the majority of the benchmarks spend more than 50% of the area and energy on approximable parts. This experiment clearly shows that there is a strong opportunity for approximate computing to improve runtime, energy, and area of dedicated hardware across a large number of domains such as machine learning, robotics, and signal processing.

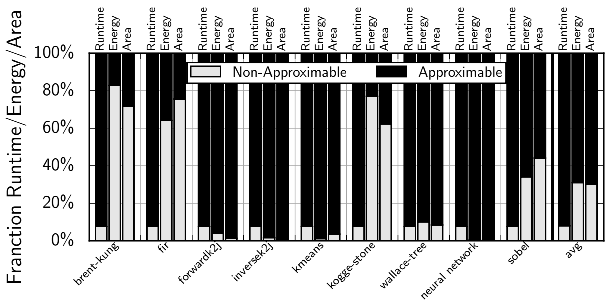


Figure 17: Total application runtime and energy breakdown between non-approximable and approximable region in dedicate hardware design.

4.2 Evaluation of Prior Approximation Techniques

In this part, we benefit from AxBench benchmarks to evaluate some of the previously proposed approximation techniques. For CPU and GPU platforms, we evaluate loop perforation [1] and neural processing unit (NPU) [2–4]. For dedicated hardware, we evaluate Axilog [5].

4.2.1 CPU Platform

Figure 18 compares loop perforation and neural processing units (NPU) accelerators for improving speedup and energy efficiency of CPU benchmarks. The maximum quality degradation is set to 10%. We restrict the degree of loop perforation and NPU invocations to limit the quality degradation to 10%.

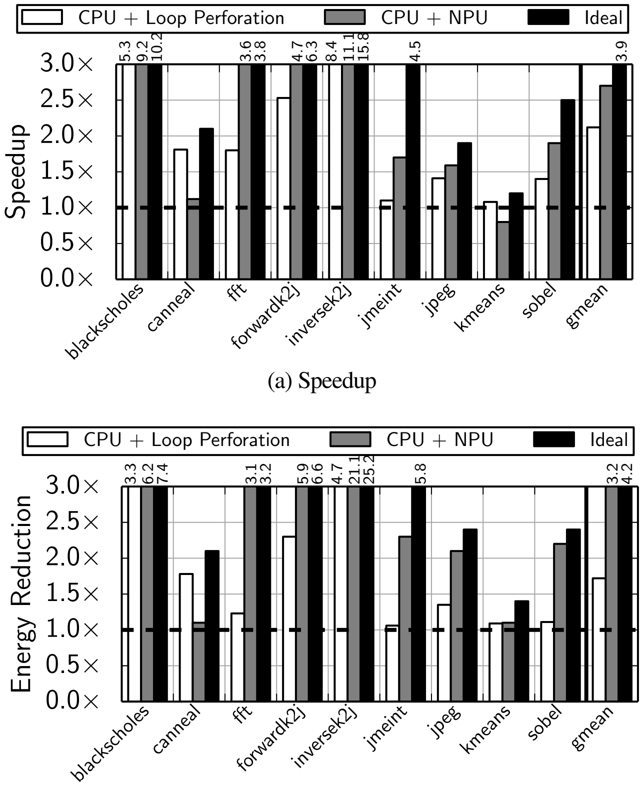
Across all benchmarks expect kmeans and canneal, CPU+NPU offers higher speedup and energy reduction as compared to CPU+Loop Perforation. The approximable region in canneal and kmeans consists of few arithmetic operations. Therefore, the communication overhead kills the potential benefits of NPU acceleration. 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 13.1× energy reduction. The average speedup (energy reduction) for loop perforation and NPU is 2.0× and 2.7× (1.6× and 2.4×), respectively.

While NPU is superior to loop perforation for improving efficiency of CPUs, it cannot reach the peak potential because it does not reduce the number of L1-D misses (it does not approximate loads). Unlike instruction accesses that get eliminated by neural transformation, data accesses remain the same. Figure 3 indicates that about 80% of time and energy of AxBench's benchmarks are spent on approximable parts, which suggests the maximum speedup and energy improvement of an approximate computation technique is 5×. NPUs only realize half of the opportunity, mainly because they do nothing for data misses.

4.2.2 GPU Platform

Figure 19 compares loop perforation and neural processing units (NPU) accelerators for improving speedup and energy efficiency of GPU benchmarks. The maximum quality degradation is set to 10%. We restrict the degree of loop perforation and NPU invocations to limit the quality degradation to 10%.

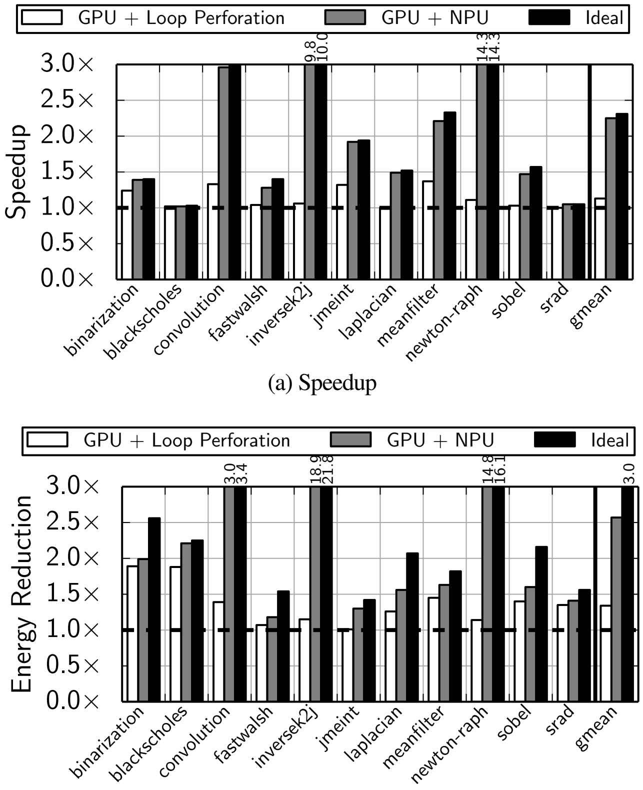
Across all benchmarks, GPU+NPU offers higher speedup and energy reduction as compared to GPU+Loop Perforation. The maximum speedup and energy reduction is registered for newton-raph (14.3×) and inversek2j (18.9×), respectively. The average speedup (energy reduction) for loop perforation and



(a) Speedup

(b) Energy Reduction

Figure 18: The comparison between NPU acceleration [2] and loop perforation [1] approximation techniques in CPU platform with 10% quality degradation.



(a) Speedup

(b) Energy Reduction

Figure 19: The comparison between NPU acceleration [3, 4] and loop perforation [1] approximation techniques in GPU platform with 10% quality degradation.

NPU is 1.1× and 2.3× (1.3× and 2.6×), respectively.

Unlike CPU+NPU, which only realizes half of the opportunity, GPU+NPU realizes all of the opportunity of approximate computation. The numbers in Figure 10 suggest the maximum speedup and energy reduction of an approximation technique to be 3.1× and 3.0×, respectively. As Figure 19 shows, GPU+NPU realizes 72.5% (83.4%) of the speedup (energy reduction) opportunity, respectively. While NPU does nothing for data misses, a GPU executes many threads in parallel to hide data misses. Consequently, massively parallel GPUs augmented with neural accelerators achieve the peak potential of approximate computation. The only exceptions are blackscholes and srad that has high off-chip bandwidth utilization (off-chip bandwidth is saturated). As the off-chip bandwidth is a bottleneck, NPU accelerators cannot offer speedup though they improve the energy efficiency of the GPU.

4.2.3 *Dedicated Hardware*

We evaluate Axilog hardware approximation technique [5] 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.

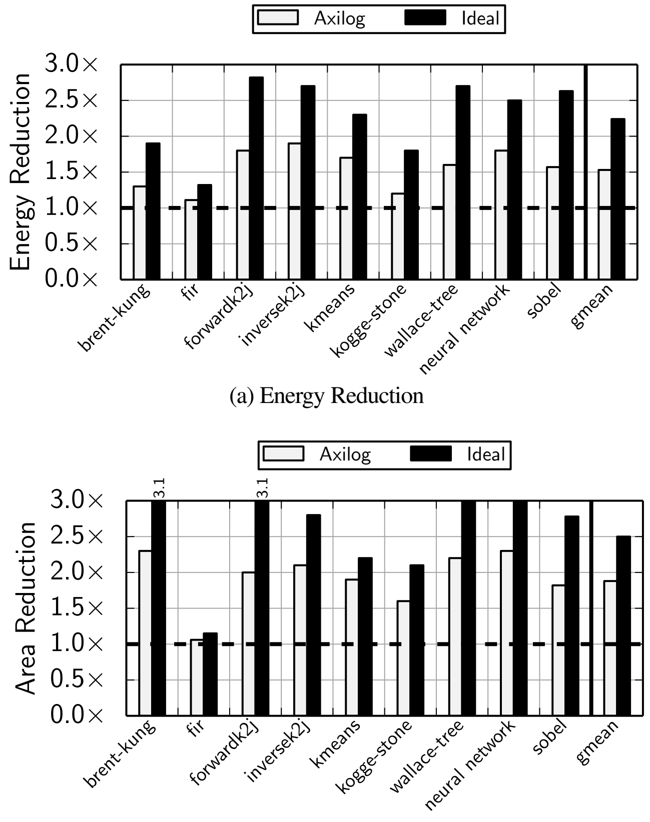
Figure 20 shows the energy and area reduction of applying Axilog to the benchmarks. We do not include a graph for speedup as Axilog does not affect the performance of the benchmarks. Axilog is quite effective at reducing the energy and area needed by the benchmarks. The energy reduction across all benchmarks 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×.

Figure 17 shows that roughly 70% of the energy and area of the benchmarks, on average, is due to approximable parts, which translates to the maximum area and energy reduction of 3.3×. 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.

5 Related Work

Approximate computing. Recent work has explored various approximation techniques across system stack and for different frameworks such as CPUs, GPUs, and hardware design that include: (a) programming languages [9, 10, 48], (b) software [1, 29, 30, 40, 54, 55], (c) memory system [44, 46], (d) circuit-level [11, 12, 34, 36, 56], (e) approximate circuit synthesis [5, 14, 15, 57, 58], (f) memoization [29, 59, 60], (g) limited fault recovery [61–67], and (h) neural acceleration [2–4, 31, 68–73]. 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 [74–76], (b)



(a) Energy Reduction

(b) Area Reduction

Figure 20: Reduction in (a) energy and (b) area for the AxBench ASIC benchmarks by using Axilog [5] hardware approximation technique. The quality degradation is set to 10%.

big data analytics [77–79], (c) heterogeneous computing [80], (d) scientific computing [81], (e) bioinformatics [82], (f) multi-thread programming [83], (g) data mining [84], (h) embedded computing [85], (i) computer vision [86, 87], and (j) general-purpose computing [88, 89]. 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.

6 Conclusion

As the approximate computing gains popularity in different computing platforms such as CPUs and GPUs, and across the system stack, 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 multi-framework benchmark suite for approximate computing across the system stack. AxBench includes 9 benchmarks for CPUs, 11 benchmarks for GPUs, and 9 benchmarks for hardware design. We extensively characterize all the benchmarks and provide guidelines for the potential approximation techniques that can be applied across the system stack. Furthermore, we evaluate loop perforation approximation [1] and neural processing units (NPUs) [2–4] using CPU and GPU benchmarks, and Axilog [5] using hardware design benchmarks to show the effectiveness of AxBench. We find that previously proposed techniques are effective at improving performance or energy efficiency. However, there remains significant opportunity to be explored by future approximation techniques. While all the AxBench benchmarks are ready, we have not released them publicly to preserve anonymity.

7 Acknowledgments

This work was supported by a Qualcomm Innovation Fellowship, NSF award CCF#1553192, Semiconductor Research Corpo. contract #2014-EP-2577, and a gift from Google.

References

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

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Neural acceleration for general-purpose approximate programs,” in *MICRO*, 2012.

[3] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, “Neural acceleration for gpu throughput processors,” in *MICRO*, 2015.

[4] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, “Neural acceleration for gpu throughput processors,” in *GT-CS-15-05*, 2015.

[5] 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,” Mar. 2015.

[6] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, “Conservation cores: Reducing the energy of mature computations,” in *ASPLOS*, 2010.

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

[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, “Toward dark silicon in servers,” *IEEE Micro*, vol. 31, pp. 6–15, July–Aug. 2011.

[9] 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.

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

[11] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, “Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology,” in *DATE*, 2006.

[12] L. Leem, H. Cho, J. Bau, Q. Jacobson, and S. Mitra, "ERSA: error resilient system architecture for probabilistic applications," in *DATE*, 2010.

[13] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pp. 409–414, IEEE Press, 2011.

[14] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *DAC*, 2012.

[15] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pp. 361:1–361:6, 2014.

[16] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *VLSI*, 2011.

[17] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *DATE*, 2010.

[18] S. Ramasubramanian, S. Venkataramani, A. Parandhaman, and A. Raghunathan, "Relax-and-retime: A methodology for energy-efficient recovery based design," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, May 2013.

[19] A. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC*, 2012.

[20] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pp. 1–6, IEEE, 2011.

[21] M. Kamal, A. Ghasemazar, A. Afzali-Kusha, and M. Pedram, "Improving efficiency of extensible processors by using approximate custom instructions," in *DATE*, 2014.

[22] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *ICCAD*, 2013.

[23] Y. Liu, R. Ye, F. Yuan, R. Kumar, and Q. Xu, "On logic synthesis for timing speculation," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pp. 591–596, Nov 2012.

[24] A. Lingamneni, K. K. Muntimadugu, C. Enz, R. M. Karp, K. V. Palem, and C. Piguet, "Algorithmic methodologies for ultra-efficient inexact architectures for sustaining technology scaling," in *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, (New York, NY, USA), pp. 3–12, ACM, 2012.

[25] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *ICCAD*, 2013.

[26] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, 2004.

[27] J. S. Miguel, M. Badr, and N. E. Jerger, "Load value approximation," in *MICRO*, 2014.

[28] B. Thwaites, G. Pekhimenko, H. Esmaeilzadeh, A. Yazdanbakhsh, O. Mutlu, J. Park, G. Mururu, and T. Mowry, "Rollback-free value prediction with approximate loads," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pp. 493–494, ACM, 2014.

[29] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: pattern-based approximation for data parallel applications," in *ASPLOS*, 2014.

[30] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: self-tuning approximation for graphics engines," in *MICRO*, 2013.

[31] R. S. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014.

[32] D. Patterson, "For better or worse, benchmarks shape a field: Technical perspective," *Communications of the ACM*, vol. 55, pp. 104–104, July 2012.

[33] P. D. Düben, J. Joven, A. Lingamneni, H. McNamara, G. De Micheli, K. V. Palem, and T. Palmer, "On the use of inexact, pruned hardware in atmospheric modelling," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 372, no. 2018, p. 20130276, 2014.

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

[35] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar, "A 1.45 ghz 52-to-162gflops/w variable-precision floating-point fused multiply-add unit with certainty tracking in 32nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 182–184, IEEE, 2012.

[36] S. Narayanan, J. Sartori, R. Kumar, and D. L. Jones, "Scalable stochastic processors," in *DATE*, 2010.

[37] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, ACM, 2013.

[38] I. Akturk, N. S. Kim, and U. R. Karpuzcu, "Decoupled control and data processing for approximate near-threshold voltage computing," *Micro, IEEE*, vol. 35, no. 4, pp. 70–78, 2015.

[39] M. C. Rinard, "Using early phase termination to eliminate load imbalances at barrier synchronization points," in *OOPSLA*, 2007.

[40] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *PLDI*, 2010.

[41] Z. A. Zhu, S. Misailovic, J. A. Kelner, and M. Rinard, "Randomized accuracy-aware program transformations for efficient approximate computations," in *ACM SIGPLAN Notices*, vol. 47, pp. 441–454, ACM, 2012.

[42] B. Thwaites, G. Pekhimenko, H. Esmaeilzadeh, A. Yazdanbakhsh, O. Mutlu, J. Park, G. Mururu, and T. Mowry, "Rollback-free value prediction with approximate loads," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pp. 493–494, 2014.

[43] P. Stanley-Marbell and M. Rinard, "Approximating outside the processor," 2015.

[44] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *MICRO*, 2013.

[45] M. Shoushtari, A. Banaiyan, and N. Dutt, "Relaxing manufacturing guard-bands in memories for energy savings," *Center for Embedded Computer Systems, University of California, Irvine, Tech. Rep. CECS-TR-14-04*, 2014.

[46] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving refresh-power in mobile devices through critical data partitioning," in *ASPLOS*, 2011.

[47] S. Kirkpatrick, C. Gelatt Jr, M. Vecchi, and A. McCoy, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–679, 1983.

[48] 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.

[49] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, 2012.

[50] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A full system simulator for x86 CPUs," in *DAC*, 2011.

[51] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.

[52] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *ISPASS*, 2009.

[53] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pp. 487–498, ACM, 2013.

[54] M. Rinard, H. Hoffmann, S. Misailovic, and S. Sidiroglou, "Patterns and statistical analysis for understanding reduced resource computing," in *Onward!*, 2010.

[55] J. Sartori and R. Kumar, "Branch and data herding: Reducing control and memory divergence for error-tolerant gpu applications," *Multimedia, IEEE Transactions on*, vol. 15, no. 2, 2013.

[56] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *ISLPED*, 1999.

[57] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *DATE*, 2014.

[58] A. Lingamneni, K. K. Muntimadugu, C. Enz, R. M. Karp, K. V. Palem, and C. Piguet, "Algorithmic methodologies for ultra-efficient inexact architectures for sustaining technology scaling," in *CF*, 2012.

[59] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Trans. Comput.*, vol. 54, no. 7, 2005.

[60] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis, "Eliminating redundant fragment shader executions on a mobile gpu via hardware memoization," in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ISCA '14, (Piscataway, NJ, USA), pp. 529–540, IEEE Press, 2014.

[61] M. de Kruijf, S. Nomura, and K. Sankaralingam, "Relax: An architectural framework for software recovery of hardware faults," in *ISCA*, 2010.

[62] X. Li and D. Yeung, "Application-level correctness and its impact on fault tolerance," in *HPCA*, 2007.

[63] X. Li and D. Yeung, "Exploiting application-level correctness for low-cost fault tolerance," *J. Instruction-Level Parallelism*, 2008.

[64] M. de Kruijf and K. Sankaralingam, "Exploring the synergy of emerging workloads and silicon reliability trends," in *SELSE*, 2009.

[65] S. Misailovic, S. Sidiroglou, H. Hoffman, and M. Rinard, "Quality of service profiling," in *ICSE*, 2010.

[66] Y. Fang, H. Li, and X. Li, "A fault criticality evaluation framework of digital systems for error tolerant video applications," in *ATS*, 2011.

[67] V. Wong and M. Horowitz, "Soft error resilience of probabilistic inference applications," in *SELSE*, 2006.

[68] Z. Du, A. Lingamneni, Y. Chen, K. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, January 2014.

[69] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, "Continuous real-world inputs can open up alternative accelerator designs," in *ISCA*, 2013.

[70] B. Grigorian and G. Reinman, "Accelerating divergent applications on simd architectures using neural networks," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pp. 317–323, IEEE, 2014.

[71] B. Grigorian, N. Farahpour, and G. Reinman, "BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing," in *HPCA*, 2015.

[72] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "SNNAP: Approximate computing on programmable socs via neural acceleration," in *HPCA*, 2015.

[73] L. McAfee and K. Olukotun, "Emeuro: A framework for generating multi-purpose accelerators via deep learning," in *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '15, pp. 125–135, 2015.

[74] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval,"

[75] N. Tax, S. Bockting, and D. Hiemstra, "A cross-benchmark comparison of 87 learning to rank methods," *Information Processing & Management*, vol. 51, no. 6, pp. 757–772, 2015.

[76] 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.

[77] 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*, pp. 488–499, Feb 2014.

[78] A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive analysis of the data analytics workload in cloudsuite," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 202–211, IEEE, 2014.

[79] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pp. 41–51, IEEE, 2010.

[80] 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 *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, IEEE, 2009.

[81] 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.

[82] D. Bader, Y. Li, T. Li, and V. Sachdeva, "Bioperf: a benchmark suite to evaluate high-performance computer architecture on bioinformatics applications," in *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pp. 163–173, Oct 2005.

[83] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, ACM, 2008.

[84] 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*, pp. 182–188, IEEE, 2006.

[85] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pp. 3–14, IEEE, 2001.

[86] 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*, pp. 55–64, IEEE, 2009.

[87] J. Clemons, H. Zhu, S. Savarese, and T. Austin, "Mevbench: A mobile computer vision benchmarking suite," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pp. 91–102, IEEE, 2011.

[88] SPEC, "SPEC CPU2000 and CPU2006 benchmarks," SPEC CPU2000 and CPU2006 Benchmarks.

[89] A. KleinOsowski and D. Lilja, "MinneSPEC: A new spec benchmark workload for simulation-based computer architecture research," *Computer Architecture Letters*, vol. 1, pp. 7–7, January 2002.