Approximate Associative Memristive Memory for Energy-Efficient GPUs

Abbas Rahimi[‡], Amirali Ghofrani[*], Kwang-Ting Cheng[*], Luca Benini[†§], and Rajesh K. Gupta[‡]
[‡]CSE, UC San Diego, La Jolla, CA 92093, USA
[*]ECE, UC Santa Barbara, Santa Barbara, CA 93111, USA
[†]DEI, University of Bologna, 40136 Bologna, Italy
[§]IIS, Swiss Federal Institute of Technology, 8092 Zurich, Switzerland
{abbas, gupta}@cs.ucsd.edu, {ghofrani, timcheng}@ece.ucsb.edu, luca.benini@iis.ee.ethz.ch

***Abstract*—Multimedia applications running on thousands of deep and wide pipelines working concurrently in GPUs have been an important target for power minimization both at the architectural and algorithmic levels. At the hardware level, energy-efficiency techniques that employ voltage overscaling face a barrier so-called "path walls": reducing operating voltage beyond a certain point generates massive number of timing errors that are impractical to tolerate. We propose an architectural innovation, called A^2M^2 module (approximate associative memristive memory) that exhibits few tolerable timing errors suitable for GPU applications under voltage overscaling. A^2M^2 is integrated with every floating point unit (FPU), and performs partial functionality of the associated FPU by pre-storing high frequency patterns for computational reuse that avoids overhead due to re-execution. Voltage overscaled A^2M^2 is designed to match an input search pattern with any of the stored patterns within a Hamming distance range of 0–2. This matching behavior under voltage overscaling leads to a *controllable* approximate computing for multimedia applications. Our experimental results for the AMD Southern Islands GPU show that four image processing kernels tolerate the mismatches during pattern matching resulting in a PSNR $\geq$ 30dB. The A^2M^2 module with 8-row enables 28% voltage overscaling in 45nm technology resulting in 32% average energy saving for the kernels, while delivering an acceptable quality of service.**

I. Introduction

There is an ever-increasing demand for multimedia information processing. A graphical processing unit or GPU provides a programmable fabric that orchestrates over 2,000 stream cores to meet the required performance demanded by multimedia applications. Given a limited thermal envelope, powering up over 4 billion transistors makes energy efficiency a primary concern for GPUs. Earlier work has pointed to supply voltage overscaling (VOS) [1], [2] and computational reuse [3] as promising approaches to reduce energy consumption. For a core, there is a voltage and clock frequency operating point at which the core is efficiently functional, but reducing the operating voltage beyond a critical point leads to so-called "path walls" [4], [5]. The path walls effect is highly pronounced in well-optimized circuits [4]. Hitting the path walls results either in a complete core failure, or massive number of *timing errors* that are very expensive to correct, and wipe out the energy benefits of VOS.

Multimedia applications provide ability to exploit the varying degrees of tolerance to error that an application has due to its programming or inherent application needs [6]. To use this flexibility, "approximate programs", programs that produce results that may be an approximation to the specified results, have an application-dependent fidelity metric to characterize the quality of the output result. For instance, peak signal to noise ratio (PSNR) of greater than 30dB is generally considered acceptable from users perspective in image processing applications. Therefore if program execution is not 100% numerically correct due to few errors during computations, the program can still "appear" to execute correctly. However, recent experiment on an ARM Cortex-M0 core shows that VOS after the critical operating point increases the number of timing errors dramatically [7]. In a similar vein, SRAM-based cache counterpart displays useless behaviour under VOS: operating at the nominal voltage is error-free; reducing the voltage down by $\sim$25% generates few errors in data array; below that point there is a massive number of errors in every row and column [8]. This massive number of errors is beyond the capability of the approximate applications to tolerate. Efforts have been done to enable VOS in traditional CMOS-based synthesis by generating approximate hardware blocks for coarse-grained meta-function [9].

In contrast, non-volatile memories such as resistive RAM (ReRAM/memristor) offers low energy operation with 270mV–1.0V [10]. Their downside is limited durability beyond billion write operations that limits their lifetime [11]. Li et al. [12] demonstrate a 1-Mb ternary content addressable memory (TCAM) test chip using 2-transistor/2-resistive-phase-change-storage (2T-2R) cell that achieves $>10\times$ smaller cell size than SRAM-based TCAMs, and ensures reliable low voltage search operation. To build energy-efficient GPUs using the CMOS-compatible memristor parts, Rahimi et al. [13] recently integrate the TCAMs with the floating point units (FPUs) for computation reuse. These FPUs consume higher energy-per-instruction than their integer counterparts, and the overall arithmetic operations contribute to more than 70% of the total GPU power consumption in compute-intensive kernels [14].

Parallel execution in the GPU architectures provides an important ability to combine computational reuse and approximation for reducing energy. This paper exploits this opportunity to make three main contributions. **I)** We propose approximate associative memristive memory (A^2M^2) microarchitectural design to enable simultaneous VOS and computational reuse. A^2M^2 is a programmable module accessible by software to store computations that appear frequently, and is tightly integrated to every FPU in the GPU. A^2M^2 is composed of a TCAM and a crossbar-based memristor memory block that together represent the pre-stored computations as partial functionality of the associated FPU. Under VOS, A^2M^2 exhibits a *controllable* error behaviour: when we reduce the voltage from 1.0V down to 725mV, A^2M^2 still matches an input search pattern with any of the stored computations within a Hamming distance of 0, 1, or 2. **II)** We present a framework, compatible with OpenCL as an industry-standard programming for heterogeneous computing, to profile GPU kernels to identify frequent redundant computations. It applies

a fine-grained value partitioning for every FP operation, and extracts a set of values that are occurred frequently through searching the space of possible inputs provided by training samples. The framework carefully pre-stores these key computations in appropriate A^2M^2 modules for reusing them to avoid re-executions. **III)** We demonstrate the effectiveness of our approach on the Southern Islands GPUs with four image processing kernels adopted from AMD APP SDK v2.5 [15]. We use 10% of Caltech 101 computer vision dataset [16] for the training, and the full dataset for the testing. Our experimental results show that the image processing kernels for all the test images: 1) tolerate the Hamming distance mismatches during pattern matching by displaying a PSNR $\geq$ 30dB; 2) save on average 32% energy on A^2M^2 modules of size 8 made possible by approximate reuse under 28% VOS.

The rest of the paper is organized as follows. Section II surveys prior work in this specific topic area. Section III describes design of A^2M^2 for energy-efficient GPU architectures. A framework and kernel execution flow to support A^2M^2 is presented in Section IV. In Section V, we explain our methodology and present experimental results followed by conclusions in Section VI

II. Related Work

Memory-based computing has been shown significant energy efficiency using emerging non-CMOS memories which are particularly well-suited for dense non-volatile memory design [12], [17], [18], [19]. For example, spin-torque transfer RAM (STTRAM) has been used for reconfigurable frameworks which partition the entire input application into smaller representable partitions using lookup tables [17], or use co-design approach for a better application mapping [18]. Compiler support further optimizes the lookup table resource allocation among functions within a program [19]. However, these frameworks map the entire application [17], [18] or hot functions [19] to the non-volatile memory, hence limit their applicability to a subset of applications amenable to full memory-based computing. Beyond non-volatile memory-based computing, profiling driven techniques increase energy efficiency by computation reuse that avoids redundant executions [3].

Moving to parallel architectures, a memristor-based multi-threading processor is proposed that enables continuous flow multi-threading by inserting a multi-state pipeline register using memristor [20]. This register is capable of holding micro-architectural state of different active threads, therefore eliminating the thread switch penalty that improves performance and energy. However, this architecture imposes a great deal of write stress on the memristors, as high as typical registers, hence suffers from durability of the memristors. In GPUs, a temporal memoization technique reduces the energy overhead of timing error recovery by exploiting locality [21]. The technique recalls recent contexts of error-free executions on a FPU and utilize them to correct the timing errors [21]. Zhang et al. [14] propose to use imprecise FPUs in GPUs for approximate computing. However, SRAM-based lookup tables [21] and imprecise hardware blocks [14] share the common drawback in VOS. Another memoization-based technique utilizes the memristors to increase computational reuse in GPUs [13]. This technique cannot fully exploit the capability of approximate programs running in GPUs leaving an untapped energy-saving potential. We design A^2M^2 module that enables simultaneous approximate computational reuse and operation under VOS, while delivering required accuracy. Further, our framework leverages the memristor technology in the right angle by limiting the stress of write to finite number of write operations

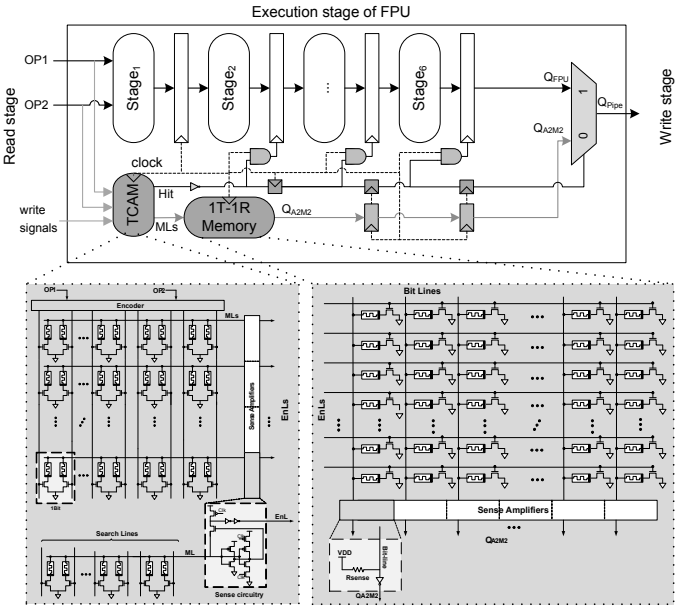


Fig. 1. Execution stage of FPU with A^2M^2 module.

only at the start of kernel execution, therefore extending the lifetime of A^2M^2 modules.

III. Energy-Efficient GPU Architecture

A. Southern Islands Architecture

We focus on one of the most recent GPUs from the AMD, the Southern Islands family (Radeon HD 7000-series). The Southern Islands is based on AMD's Graphics Core Next which is a RISC single instruction, multiple data (SIMD) architecture; it replaces the elder VLIW SIMD architecture from the Evergreen. We target Radeon HD 7970 device which has 32 compute units. Every compute unit contains a scheduler and a set of four SIMD execution units, aka vector units. Each SIMD execution unit has 16 stream cores, or parallel lanes, constituting a total number of 64 stream cores per compute unit.

An OpenCL application is formed of a host program and one or more device kernels that can be run on a GPU device. An instance of the OpenCL kernel is called a work-item. Each stream core is devoted to the execution of one work-item using the integer or FP units. Most arithmetic operations on a GPU are performed by vector instructions. A vector instruction is fetched once and executed in a SIMD fashion by all its comprising work-items. After the fetch and decode stages, the source operands for each instruction are read from vector registers or local memory. The core stage of a GPU is the execute stage, where arithmetic instructions are carried out in each stream core. When the source operands are ready in the vector unit, the execution stage starts to issue the operations into the integer units or FPUs. The execution stage of every FPU has a latency of six cycles and a throughput of one instruction per cycle [22]. Finally, the result of the computation is written back to the destination operands.

B. Approximate Associative Memristive Memory Module

In order to fully exploit the energy saving potentials of both partial memory-based computing and approximate computing, in this section we propose an approximate associative memristive memory (A^2M^2) which is tightly integrated to

each FPU. The proposed A^2M^2 microarchitecture demonstrates controllable approximate computing capabilities under VOS.

For each type of FPU, we first identify the sets of frequent input operands and store them along with their corresponding pre-calculated outputs in an A^2M^2 module. Section IV-A describes this flow in details. During the execution, in case of a match between the input values of the FPU and the input patterns stored in A^2M^2, the pre-stored results are provided by A^2M^2, and FPU re-execution is avoided for frequent operands. A^2M^2 module performs the match operation and returns the output at extremely lower energy costs compared to the FPU, thanks to the ultra-low power characteristics of memristive memories. This energy cost is further reduced by VOS that relaxes the matching criterion, from the exact to approximate, described in the following.

A^2M^2 module consists of two pipelined stages as shown in Fig. 1: (I) a memristive TCAM which stores and searches for the high frequent sets of input operands, and (II) a 1T-1R memristive memory which maintains the pre-calculated FPU output results for each set of such frequent operands. For each operation, in the first stage, the TCAM searches to determine whether there is a match between the input operands and the stored operand patterns. In case of a match, the result of the operation is read in the second stage from the corresponding line in the 1T-1R memory.

Each TCAM row stores one set of highly-frequent input operands. We use a 2T-2R cell structure for the TCAM design [12]. In this structure each bit of data is stored in a cell that consists of two memristive elements to store the pattern and two access transistors that decouple the memristors from a corresponding match line (ML), as shown in Fig. 1. To program the TCAM, the write voltages are applied on the match lines (ML), and access-transistors of select devices are connected via the search line (SL) to perform the write operation.

A memristive TCAM operation is based on the fact that a low-resistance path to the ground discharges a precharged line faster than a high-resistance path. Each row in the TCAM has a match line which is precharged during a precharge phase: SLs are deactivated to disconnect the access transistors. During the evaluation phase, one of the access transistors in each bit-cell is ON and connects the ML to the ground via a high- (or low-) resistance path if the pattern-under-search matches (mismatches) the stored pattern. In case of an exact match, i.e. bit-by-bit, the ML stays charged for an extended period of time due to the high-resistance of the memristive device that connects the ML to the ground. If the pattern-under-search and the stored pattern mismatch by even a single bit, the ML will be discharged quickly because of the existence of low-resistive path(s) between the ML and ground, providing a clear margin between an exact match and mismatches. As the number of bit-mismatches increases, the ML will be discharged even faster. A clocked self-referenced sensing circuitry and a 2-bit data encoding scheme is applied [12] to further increase the noise margin and provide a digital match/mismatch output signal. Fig. 2 illustrates the evolution of the digital "match" signal during the evaluation phase for different number of bit-mismatches based on SPICE simulation results. As it is expected, this signal drops faster when a larger number of bit-mismatches exist. The digital match signals are sampled (i.e. latched) at the end of the evaluation phase. A logic '1' means that the line is not discharged yet, indicating a match. The latched match signals are then fed to the 1T-1R memory stage as enable lines (EnL), to read the previously-computed results that are stored in the 1T-1R memory. The logical OR of the EnLs represents a "hit signal" which indicates that the result is provided by A^2M^2 module.

In case of a match, the pre-computed result ($Q_{A^2M^2}$) is read from the memristive memory at negligible energy cost and is propagated toward the end of the FPU pipeline along with the hit signal. The propagated hit signal is used as a clock-gating signal for the remaining stages of the FPU to avoid the redundant computation. Given that only the first stage of the FPU is concurrently working with TCAM, other FPU stages are clock-gated in case of a match which results in considerable amount of energy saving. In case of a TCAM miss, the FPU works normally, and its result (Q_{FPU}) is selected as the pipeline output. The hit signal selects whether the Q_{FPU} or $Q_{A^2M^2}$ should be reported as the output.

Fig 1 shows the structure of such 1T-1R memory that is used to store the output patterns. To program the memory, a write voltage is applied on the bit-lines, while the enable lines are used to select the target cell. For read operation, the enable lines are derived by the EnL values of TCAM. Assuming an exact matching, either none or only one of EnLs are active at any given clock cycle, connecting the bit-line to the ground through a high-/low-resistance memristive cell, depending on the stored data. The read circuitry works as a voltage divider and is consisted of a sense resistor R_{Sense} and a NOT gate. If the memristor is in the high-resistance state, which represents logic '0', $R_{Memristor} >> R_{Sense}$ and thus the voltage drop on R_{Sense} is negligible and the output of the NOT gate will be a logic '0'. In case of a low-resistance memristor, $R_{Sense} >> R_{Memristor}$, thus most of the voltage is dropped on R_{Sense} and the output of the read circuitry is a logic '1'. It can be observed in Fig. 2 that for few bit-mismatches (e.g. 1 or 2), the drop time of the match signals differ with clear margins. Hence, by shortening the evaluation period (i.e. faster sampling), or similarly reducing the supply voltage while preserving the same evaluation period, a "controllable" approximate matching can be realized in which a pattern with a Hamming distance of 1 or 2 (i.e., the number of bit-mismatches) is considered as a "match". Operating at the nominal voltage of 1V guarantees an exact matching with 0 number of bit-mismatch. If we reduce the voltage to 775mV, TCAM also matches the input pattern with any of the stored patterns if there is a Hamming distance of 1 between them (1-HD approximate matching). VOS down to 720mV matches the input patterns with 2 bit-mismatches (2-HD approximate matching). Further lowering the supply voltages results in an abrupt increase in the number of bit-mismatches.

However, the approximate matching has two downsides: (I) possibility of a false match, and reporting a wrong output as the result of the computation, and (II) having several matches, which would enable several word-lines in the 1T-1R memory,

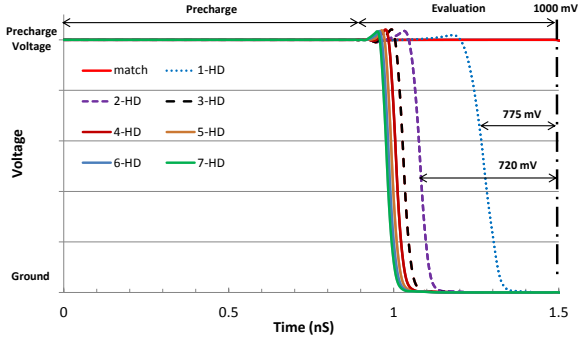


Fig. 2. TCAM match operation under VOS.

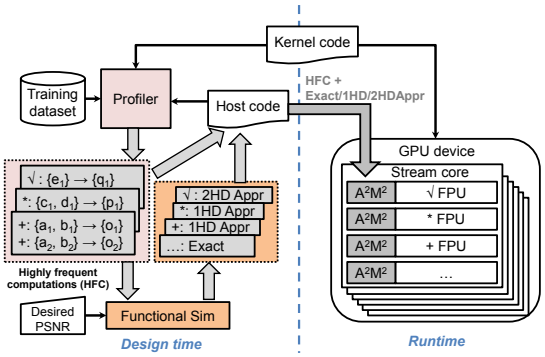


Fig. 3. Execution flow using A^2M^2: design time profiling + runtime reuse.

resulting in the logical OR of the corresponding outputs being reported as the output of A^2M^2 module ($Q_{A^2M^2}$). Possibility of several matches can be avoided if the stored patterns in the TCAM have a minimum Hamming distance (e.g. 3 for 1-HD approximate matching respectively); this is practical given the typical TCAM word-size (i.e., 32, 64, or 96), and the small number of TCAM rows. As for the case of a false match, its likelihood is reduced by a proper sizing of A^2M^2 module described in Section V-B. We limit the match set such that it decreases the likelihood of a false matching and of the introduced error at the same time. In Section V-B, we show the application of this approximate matching for different image processing kernels that can tolerate the introduced errors and display a high PSNR while benefiting from the lower energy consumptions. Moreover, A^2M^2 module could be designed in a *hybrid* fashion to always exclude the error in a few critical bits (e.g., the sign and exponent bits); for instance, by applying a high voltage to those bits to perform a robust and exact matching, lowering the significance effect of such error.

IV. Framework to Support A^2M^2

In this section, we briefly describe our approach to programming A^2M^2 and evaluation of A^2M^2 effectiveness in improving energy efficiency of GPUs.

A. Execution Flow

Execution flow using A^2M^2 has two main stages: (I) design time profiling, and (II) runtime computational reuse. Fig. 3 illustrates this execution flow. The goal of profiling stage is to identify redundant computations with a high frequency of occurrence. In the profiling stage, we have an OpenCL kernel, a host code with a training input dataset. We focus on the individual FPUs to observe the dispersion of the input operands at the finest granularity. To expose highly frequent set of operands for each FP operation, we individually profile every type of FP operation and keep the distinct sets of the input operands with the related output result. The output of this stage for every FP operation is highly frequent computations (HFC): a sorted list of sets of values, each set has the input operand(s) and the related result, and the sets are sorted based on their frequency of occurrence. After extracting HFC, we need to determine how much approximation can be tolerated during the reuse of these key computations. To do so, we leverage the Southern Islands functional simulator to apply different matching constraints for determining the degree of approximation applicable to each A^2M^2 module. The simulator starts with the exact matching and then increases the degree of approximation step-by-step by applying 1-HD and 2-HD approximate matching. For every step, the output image is compared with a *golden* output image to measure PSNR. Finally, the maximum degree of approximation is determined for each A^2M^2 module such that the introduced errors result in a PSNR higher than the desired PSNR (e.g., 30dB). This profiling stage is a *one-off* activity whose cost is amortized across all future usage of the kernel.

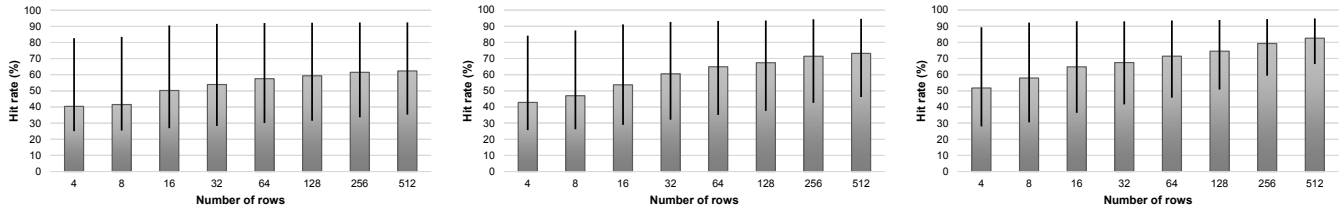
In the next step, the framework transfers the output of the profiling stage to A^2M^2 modules for runtime reuse. The AMD compute abstraction layer (CAL) provides a runtime device driver library that supports code generation, kernel loading, and allows the host program to interact with the stream cores at the lowest-level. A^2M^2 module are designed to be addressable by software therefore the host code can program them using CAL. Right before launching the kernel execution, the host code programs A^2M^2 modules: for every type of FP operation activated during the kernel, a subset of HFC (up to few hundred bytes depending up on the size of A^2M^2) in conjunction with the degree of applicable approximation is set for the corresponding A^2M^2 modules accordingly. In this way, the framework concurrently programs all the A^2M^2 modules integrated to a type of FPU across all the available compute units in the GPU, since their content is equivalent.

B. Design Space for A^2M^2

Here, we explain the design space for utilizing A^2M^2 modules as a case study for Roberts filter, one of our edge detection kernels. We evaluate the trade-off between the size of A^2M^2 module, i.e., the number of rows that store different patterns, with its hit rate. A higher hit rate means higher number of operands are matched with the stored computations in A^2M^2 module, therefore there is no need for re-executing the results for those values, leading to higher energy saving. We quantify the hit rate of A^2M^2 module for multiply-accumulator (MAC) FPU for 100 test input images. Fig 4 summarises the minimum, the maximum, and the average (shown in bars) hit rates of A^2M^2 module with a wider range of sizes. The experiment is repeated for the three matching constraints.

Fig 4(a) shows the hit rates for the exact matching. A^2M^2 module with 4-row displays the hit rates in the range of 25%–83%. Increasing the size of A^2M^2 from 4-row to 8-row, and to 16-row improves the average hit rate from 40% to 42%, and to 50%. Overall, the average hit rates increases less than 12% when the number of rows is increased from 16 to 512. A similar trend of the hit rates versus A^2M^2 sizes is observed for the approximate matching, as shown in Fig 4(b)-(c). Once the number of rows is increased from 16 to 512, the average hit rates improves less than 19% and 18% for 1-HD and 2-HD approximate matching, respectively. Fig 4 also illustrates that an A^2M^2 with a fixed size experiences higher hit rates by switching from the exact matching to any of the approximate matching. For instance, the hit rate of A^2M^2 with 4-row increases 12% on average (from 40% to 52%) by using 2-HD approximate matching instead of the exact matching. This increased hit rate is because A^2M^2 relaxes the matching constraint therefore more number of input patterns are approximately matched with one of the stored patterns.

In a nutshell, choosing large A^2M^2 size has two disadvantage. I) It diminishes the gain of energy saving, because after a certain size the average hit rates almost saturates, while the energy consumption of the A^2M^2 increases for larger sizes. For example, increasing A^2M^2 size from 8-row by 64× only brings 25% higher hit rates with 2-HD approximate matching. This significantly lowers the hit rate per unit of power consumed by A^2M^2. In Section V-B, we show that enlarging A^2M^2 beyond



(a) Exact matching (b) 1-HD approximate matching (c) 2-HD approximate matching

Fig. 4. Hit rate versus size of A^2M^2 for MAC during Roberts filter executions.

a certain size will not bring any energy saving. II) It increases the likelihood of false matches that might quickly drop PSNR below the desired threshold. Our profiling results indicate that Roberts filter is able to tolerate the errors in computations (an average PSNR of 34dB) with A^2M^2 modules of maximum 512-row using 2-HD approximate matching. Increasing A^2M^2 size after 512-row drops the PSNR below 30dB. Visual depiction and the corresponding PSNR of different matchings for one of the test images are shown in Fig 5.

V. Experimental Results

A. Experimental Setup

We focus on the AMD Southern Islands GPU, Radeon HD 7970 device, but our method can be applied to other GPUs as well. We have adopted image processing applications from AMD APP SDK v2.5 [15] a software ecosystem suitable for stream applications written in OpenCL. We have examined four image processing filters: Roberts, Sobel, Sharpen, and Shift. Multi2Sim [22], a cycle-accurate CPU-GPU simulation framework, is used for profiling and simulations. These kernels typically apply a 2D convolution; we extract frequently activated FPUs during the kernel executions: adder (ADD), multiply (MUL), multiply-accumulator (MAC), and SQRT. Accordingly, the 6-stage balanced FPUs are generated and optimized using FloPoCo [23]. These FPUs are synthesized and mapped using a 45-nm ASIC flow. The front-end flow has been performed using *Synopsys Design Compiler*, while *Synopsys IC Compiler* has been used for the back-end. The FPUs have been optimized for power and a signoff clock period of 1.5ns. Finally, *Synopsys PrimeTime* is used to report power at the nominal operating voltage of 1.0V. The second column of Table I shows the energy per operation for each FPU.

Considering the single precision FPUs, we design A^2M^2 module with different word-sizes based on the type of FPU. TCAM has a word-size of 32-bit for SQRT, 64-bit for ADD, MUL, and 96-bit for MAC; while the crossbar-based memory has a fixed word-size of 32-bit for any FPU to maintain the outputs. To estimate power and delay of A^2M^2 module, transistor-level SPICE simulations are done using *Cadence Virtuoso*. For the memristor parts, we integrate 50K R_{on} and 50M R_{off} models based on the measurements of fabricated memristors [24]. For the line resistances and capacitances, we use the same model and numbers reported in [25]. Energy operation of A^2M^2 modules is shown in Table I. Given the clock period of 1.5ns, A^2M^2 modules can reliably work under the designated VOS points (see Section III-B). FPUs face massive errors, in this range of VOS, which is simply too high to be useful. We integrate a functional model of A^2M^2 module into Multi2Sim that computes the Hamming distance for every FP operation to quantify the hit rates and PSNR drops.

B. Energy Saving with Corresponding PSNR

Table I summarizes the energy consumption per operation for individual FPUs, and different sizes of A^2M^2 modules in the cases of exact matching, 1-HD, and 2-HD approximate matching. The energy numbers show the potential of A^2M^2 modules to reduce the energy consumption per operation. For example for SQRT operation, an exact-matcher A^2M^2 module with 8 rows provides $\approx 8\times$ higher energy efficiency compared to FPU counterpart. Although both A^2M^2 (exact) and FPU work at the nominal voltage of 1.0V, this energy saving is accomplished through the ultra-low power memristive-based computing. The energy saving is further improved by allowing the approximate matching, which improves the energy efficiency by factors of $16\times$ and $22\times$, for 1-HD and 2-HD approximate matching respectively. Such saving trend is consistent for different types of FPUs, and different sizes of A^2M^2 modules.

Table I also demonstrates that increasing the size of the A^2M^2 beyond a limit sacrifices the energy efficiency. For instance in case of ADD operation, an exact-matcher A^2M^2 module with 64-row roughly consumes as much energy as FPU itself. Any larger A^2M^2 module can incur energy penalty rather than improving the energy consumption; since the aggregate energy of integrating FPU with A^2M^2 module cannot be paid off by the power saving offered by even an ideal hit rate. In the following, we present energy saving of the kernels using A^2M^2 modules with different sizes.

For the four image processing kernels, our framework uses 10% of Caltech 101 computer vision dataset [16] for the training to extract HFC as explained in Section IV-A. Depending on the size of A^2M^2 modules, the framework loads 4, 8, 16, 32, and 64 top pairs of HFC to A^2M^2 modules before the kernel execution. We quantify the average energy saving and the corresponding average PSNR degradation over the full dataset [16] as the test cases. Fig. 6 shows the normalized energy compared to FPUs for each kernel. For all the kernels, the exact-matcher A^2M^2 modules with 64-row exhibit poor energy efficiency, for instance Sobel (or Sharpen) faces 20% (17%) higher energy consumption compared to using the normal FPUs. A^2M^2 modules with sizes smaller than 64-row provide a significant range of energy saving (16%–

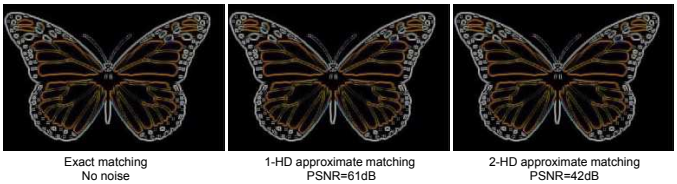
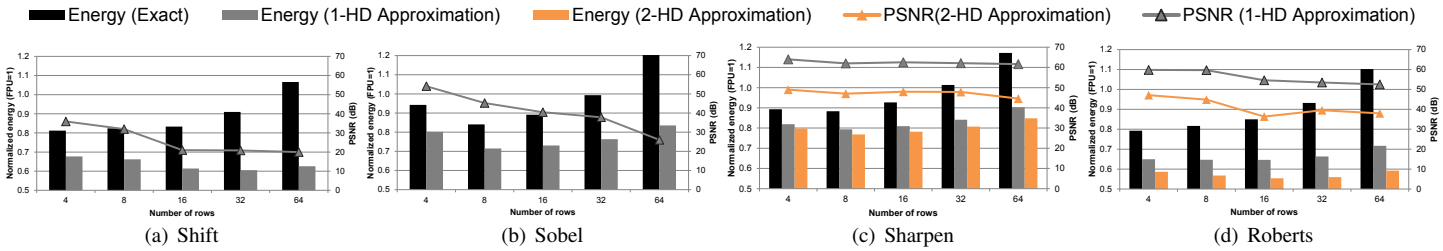


Fig. 5. Visual depiction of the output quality degradation with exact, 1-HD, 2-HD approximate matching for Roberts filter.

TABLE I. ENERGY CONSUMPTION (FJ) PER OPERATION IN 45NM TECHNOLOGY FOR FPUS AND A^2M^2.

Module	FPU (1.0V)	A^2M^2 (Exact Matching)					A^2M^2 (1-HD Approximate Matching)					A^2M^2 (2-HD Approximate Matching)				
		4-row	8-row	16-row	32-row	64-row	4-row	8-row	16-row	32-row	64-row	4-row	8-row	16-row	32-row	64-row
ADD	4742	1176	1403	1858	2740	4568	644	732	906	1262	1953	505	555	709	999	1479
MUL	9891	1176	1403	1858	2740	4568	644	732	906	1262	1953	505	555	709	999	1479
SQRT	9983	934	1137	1528	2322	3901	514	594	756	1084	1738	397	441	593	864	1332
MAC	12051	1410	1653	2122	3096	5071	774	867	1052	1422	2151	612	667	832	1124	1627



(a) Shift (b) Sobel (c) Sharpen (d) Roberts

Fig. 6. A^2M^2 normalized energy and PSNR: for different sizes, matching criteria, and kernels – values are averaged over the full dataset [16].

45%) depending on the size and the degree of approximation. As shown in Fig. 6(b), A^2M^2 modules with 4-row reduce the average energy of Sobel by 20% using 1-HD approximate matching. Increasing the size to 8-row leads to a higher average energy saving of 28% because of the higher hit rate. However, increasing the size beyond 8-row is not optimum because the amount of energy saving offered by the extra hit events is less than the energy overhead due to the increased A^2M^2 sizes. We should note that once we reduce the voltage of FPUs down to 775mV, they face massive number of errors making them impractical to use for low power computations.

Sobel and Shift kernels cannot tolerate the errors using 2-HD approximate matching, as opposed to Sharpen and Roberts filters. For all the kernels, PSNR is degraded with larger A^2M^2 sizes. Increasing the number of stored patterns beyond 32 (or 8) for Sobel (or Shift) abruptly increases the likelihood of a false match that introduces more computational errors resulting in a dropped PSNR of 30dB or lower. Considering the acceptable PSNR of 30dB or higher, A^2M^2 modules with 8-row provide the best average energy saving for Sobel (28%), Sharpen (23%), and Shift (34%); Robert exhibits the best energy saving of 45% with A^2M^2 modules of size 16-row. Choosing 8-row as the size of A^2M^2 modules brings an average energy saving of 32% across all four kernels, while guaranteeing the acceptable PSNR.

VI. CONCLUSION

We propose A^2M^2 as an associative memory module that mixes emerging memristor technology benefits with the application needs to deliver higher energy efficiency. A^2M^2 modules are tightly integrated to every FPU to save energy by: I) recalling the frequent computations therefore avoiding re-executions, and II) operating at VOS by accepting the approximate matches. Using the memristor parts in designing A^2M^2 enables 28% VOS while incurring up to 2 bits mismatch during the operand matching. We observe that this introduced error into the computations is tolerable by the image processing kernels delivering an acceptable PSNR. Experimental results on the Southern Islands GPU show the integrated A^2M^2 modules with 8-row reduce the average kernel energy by 32%. Our continuing work will explore methods to integrate A^2M^2 in a programming environment that enables accuracy- and reliability-aware optimizations of approximate kernels.

VII. ACKNOWLEDGMENTS

This work was supported by the NSF's Variability Expedition (1029783), ERC-AdG MultiTherman (291125), FP7 Virtical (288574), and AFOSR-MURI (FA9550-12-1-0038).

REFERENCES

[1] D. Jeon, et al., "Design methodology for voltage-overscaled ultra-low-power systems," *IEEE TCAS II*, vol. 59, pp. 952–956, Dec 2012.

[2] K. He, et al., "Circuit-level timing-error acceptance for design of energy-efficient dct/idct-based systems," *IEEE TCSVT*, vol. 23, pp. 961–974, June 2013.

[3] W. Wang, et al., "Profiling driven computation reuse: an embedded software synthesis technique for energy and performance optimization," *IEEE ICVD*, 2004.

[4] S. Ramasubramanian, et al., "Relax-and-retime: A methodology for energy-efficient recovery based design," *IEEE DAC*, pp. 1–6, May 2013.

[5] J. Patel, "Cmos process variations: A critical operation point hypothesis," Online Presentation, June 2008.

[6] M. A. Breuer, "Multi-media applications and imprecise computation," *IEEE DSD*, pp. 2–7, 2005.

[7] L. Lai et al., "A case study of logic delay fault behaviors on general-purpose embedded processor under voltage overscaling," Tech. Rep., EE-UCLA, 2014.

[8] M. Gottscho, et al., "Power / capacity scaling: Energy savings with simple fault-tolerant caches," *IEEE DAC*, pp. 1–6, 2014.

[9] D. Mohapatra, et al., "Design of voltage-scalable meta-functions for approximate computing," *IEEE DATE*, pp. 1–6, March 2011.

[10] M.-F. Chang, et al., "19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," *IEEE ISSCC*, pp. 332–333, Feb 2014.

[11] H. Lee, et al., "Evidence and solution of over-RESET problem for HfOX based resistive memory with sub-ns switching speed and high endurance," *IEDM*, 2010.

[12] J. Li, et al., "1 mb 0.41 μm^2 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing," *IEEE JSSC*, pp. 896–907, April 2014.

[13] A. Rahimi, et al., "Energy-efficient gpgpu architectures via collaborative compilation and memristive memory-based computing," *IEEE DAC*, pp. 1–6, 2014.

[14] H. Zhang, et al., "Low power gpgpu computation with imprecise hardware," *IEEE DAC*, pp. 1–6, 2014.

[15] "AMD APP SDK v2.5 [online]. Available: http://www.amd.com/stream"

[16] "Caltech 101 [online]. http://www.vision.caltech.edu/Image_Datasets/Caltech101/"

[17] S. Paul, et al., "Nanoscale reconfigurable computing using non-volatile 2-d sttram array," *IEEE Nanotechnology*, pp. 880–883, July 2009.

[18] S. Paul, et al., "Energy-efficient reconfigurable computing using a circuit-architecture-software co-design approach," *IEEE JETCAS*, pp. 369–380, 2011.

[19] J. Cong, et al., "Energy-efficient computing using adaptive table lookup based on nonvolatile memories," *IEEE ISLPED*, pp. 280–285, Sept 2013.

[20] S. Kvatinsky, et al.,"Memristor-based multithreading," *IEEE CAL*, 2013.

[21] A. Rahimi, et al., "Temporal memoization for energy-efficient timing error recovery in gpgpus," *IEEE DATE*, pp. 1–6, March 2014.

[22] "Multi2sim [online]. Available: https://www.multi2sim.org/"

[23] "Flopoco [online]. Available:http://flopoco.gforge.inria.fr/"

[24] K.-H. Kim, et al.,"A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano Letters*, pp. 389–395, 2012.

[25] A. Ghofrani, et al., "Towards data reliable crossbar-based memristive memories," *IEEE ITC*, pp. 1–10, Sept 2013.