ACAM: Approximate Computing Based on Adaptive Associative Memory with Online Learning

Mohsen Imani†, Yeseong Kim†, Abbas Rahimi‡, Tajana Rosing†

†Computer Science and Engineering, UC San Diego, La Jolla, CA 92093, USA

‡Electrical Engineering and Computer Science, UC Berkeley, Berkeley, CA 94720, USA

{moimani, yek048, tajana}@ucsd.edu, abbas@eecs.berkeley.edu

ABSTRACT

The *Internet of Things* (IoT) dramatically increases the amount of data to be processed for many applications including multimedia. Unlike traditional computing environment, the workload of IoT significantly varies overtime. Thus, an efficient runtime profiling is required to extract highly frequent computations and pre-store them for memory-based computing. In this paper, we propose an approximate computing technique using a low-cost adaptive associative memory, named ACAM, which utilizes runtime learning and profiling. To recognize the temporal locality of data in real-world applications, our design exploits a reinforcement learning algorithm with a least recently use (LRU) strategy to select images to be profiled; the profiler is implemented using an approximate concurrent state machine. The profiling results are then stored into ACAM for computation reuse. Since the selected images represent the observed input dataset, we can avoid redundant computations thanks to high hit rates displayed in the associative memory. We evaluate ACAM on the recent AMD Southern Island GPU architecture, and the experimental results shows that the proposed design achieves by 34.7% energy saving for image processing applications with an acceptable quality of service (i.e., PSNR>30dB).

Keywords

Approximate computing, Associative memory, Online learning, Non-volatile memory

1. INTRODUCTION

Going toward the *Internet of Things* (IoT) and the big data computation significantly increases the size of input data on the recent processors. In this era, many IoT workloads are going to be run on the GPUs in either mobiles or the clouds such as data centers. In particular, multimedia processing as an instance of IoT workload have rapidly proliferated, and to achieve timely performance demand, they require to be accelerated using efficient massive parallel processors [1, 2]. In addition, due to locality of dataset, similar computations repeatedly happen, thus giving an opportunity to significantly reduce the amount of computations based on memory-based computations [3]. To this end, an associative memory in the form of a lookup table has been exploited to reduce the number of redundant computations. A software implementation pre-stores frequent patterns on a hash table and retrieves them using a set of keys that replace original computations. In order to enhance the performance of the lookup table, associative memories can be implemented in hardware using ternary content addressable memory (TCAM).

However, to utilize TCAMs in *computation-with-memory* [4], there are two technical challenges. First, the system design has to consider the actual workloads which keep changing rapidly over different contexts such as time, place, and applications. Market research shows significant growth on interactions with external environment using sensor employments. Therefore, it is obvious that filling associative memories with offline data, on design time, cannot provide desirable hit rates [5]. Since with today's interactive IoT workloads, we need to have a context-aware associative memory which should adapt to the environment. Therefore, runtime profiling is one the essential components of the associative memories for their practical deployment on parallel processors. Second, CMOS-based TCAMs consume very high energy for the search operation. This limits the applicability of these memories to classification and IP routing [6]. Non-volatile memories (NVMs) open a new field to have an efficient memory-based computation [7]. Resistive random access memory (ReRAM) and spin-transfer torque RAM (STT-RAM) are two kinds of low leakage and dense NVMs which are based on memristive and magnetic tunneling junction (MTJ) devices respectively. Moreover, NVM-based TCAMs can further reduce energy consumption by applying *voltage over scaling* (VOS) [8] or reducing the search switching activity [9].

In this paper, we propose a novel approximate computing framework using an adaptive associative memory, called ACAM, with a capability of learning-based runtime profiling. The proposed design also addresses the endurance and cost issues of associative memories for online learning, thus providing a robust and practical solution for a wide range of dynamic workloads on parallel processor architectures. Our design goal is to find the best input data with higher hit rate to adaptively fill the rows of an associative memory and improve overall energy. The learning-based profiling runs in the following steps: (i) Machine learning algorithm finds the image of interest from input dataset based on pixel similarities. The algorithm identifies the most represented data, which is likely to be used in the near future, for profiling based on the proposed TD-LRU policy. (ii) We profile the selected images of interest based on a low-cost approximate concurrent state machine to keep track of the number of repeated computations. The approximate profiling is implemented using hash functions and a bloom filter, thus enhancing energy efficiency at the expense of minimal acceptable errors. In the circuit-level design, to address the endurance and the lifetime issues caused by frequent runtime updates, ACAM exploits high endurance and robust MTJ-based TCAM and memory block. In addition, we apply approximation for a selected part of associative memory to balance the tradeoff between energy and accuracy. Thanks to the proposed method with an efficient runtime profiling, parallel processors can efficiently process a large and active dataset with a support of the adaptive associative memory. Our evaluation shows that the proposed ACAM improves the energy efficiency of GPGPU by 34.7% with acceptable PSNR (peak signal-to-noise ratio) of more than 30dB for image processing applications.

2. RELATED WORK

Non-volatile memories such as ReRAM and STT-RAM are good candidate to design an efficient and low leakage power associative memories [7] [10] [11]. Earlier efforts have used these ReRAM and STT-RAM technologies to design a stable and efficient TCAM.

ISLPED '16, August 08-10, 2016, San Francisco Airport, CA, USA

DOI: http://dx.doi.org/10.1145/2934583.2934595

However the endurance of ReRAMs limits to 10^6-10^7 write operations while the endurance of STT-RAMs is very high (>10^{15}) [12]. Thus, efficient MTJ-based TCAM cell was proposed to address the ReRAM endurance problem [13].

With the development of the efficient TCAM designs, the associative memory exploits the high number of pattern similarities to decrease the amount of computations [5, 14-17]. In such *computation-with-memory* techniques, the associative memory consists of two main blocks: a TCAM for an input lookup table and a memory for output data. Since a streaming core processes the workload while executing an instruction with *operands* (e.g., $r1 and $r2 for "add $r1, $r2" instruction,) the TCAM pre-stores some operands for frequent input patterns while the corresponding outputs is stored in the memory. In the actual computation, input operands of an executed instruction is compared to all TCAM rows in parallel, and in a case of a hit (if any), the system stops the processor computation by using the clock-gating technique. Then, a signal on the matched line activates the corresponding line of the memory to retrieve the output of the computation. Low energy consumption of the NVM-based associative memory motivates many use cases of the memory-based computation, including query processing [18], search engine, text processing [19], image processing [20], pattern recognition and data mining [21].

Recent work mainly target to use the memory-based computation to improve the computation efficiency. In GPUs, Zhang, *et al.* [22] leverage VOS for "imprecise" Floating Point Units (FPUs). However, imprecise blocks such as the SRAM-based look-up tables suffer from high error rate when applying VOS [22]. Rahimi, *et al.* in [17] used a memristor-based look-up table to increase the computational reuse in GPGPUs. Imani *et al.* proposed a configurable approximate associative memory architecture to reduce the TCAM energy consumption by applying selective VOS [8]. Multi-access single charge TCAM has been proposed to reduce the search energy consumption by reducing the number of updates on the CAM [23, 24]. Multi-stage search associative memory is another design which exploits selective row activation and in-advance precharging to respectively improve energy consumption and mitigating the overhead of sequential access. These previous NVM-based associative memories initialize the TCAM block with the offline profiling data without the update capability because of limited endurance of resistive associative memories. However, since running dataset can be significantly changed over different contexts in reality, the design-time offline profiling, which sticks to a limited sample size, is not suitable to choose represented dataset. It limits the applicability of the use of the associative memory on the parallel processors.

In this paper, we proposed an associative memory with online learning to address the main limitation in enabling approximate computation on the real GPU architectures. In the proposed design, learning algorithm finds the best input dataset of interest for profiling; then, an approximate concurrent state machine profiles selected inputs at runtime. Our evaluation shows that the proposed design achieves significantly high energy efficiency based on the online learning especially for running applications with large dataset.

3. DESIGN OF PROPOSED ACAM

In this section, we describe our proposed approximate computing framework, ACAM, in detail. In order to fill the TCAM with highly reusable data, the proposed technique selects representative inputs as profiling candidates using an online learning at runtime. This strategy enables two main advantages. First, filling the TCAM adaptively with the consideration of data locality increases the TCAM hit rate. Thus, by increasing the average time that the FPU is in the clock gating mode, it improves system energy efficiency as well. Second, the runtime profiling allows us the memory-based computation on the real systems. For example, the runtime profiling selects representative dataset while handling not only multiple applications and but also newly installed applications which cannot be considered in the design time. In addition, we can change the TCAM pre-stored data based on the contexts.

Figure 1 illustrates the architectural overview of the proposed ACAM. The execution flow of the system has three main steps. First, we use an efficient online learning algorithm to find proper dataset for profiling by intelligently identifying input image patterns. After that, the approximation profiler, which uses the concurrent state machine [25, 26], profiles operands for instructions, and identify frequent patterns approximately. Based on the profiled result, we fill the proposed associative memories with frequently observed operands which will be likely to appear in future computations. Then, the GPU device can process executed instructions using the pre-stored data. We next describe the key component of our approximate computing design, GPU processing with the proposed adaptive associative memory.

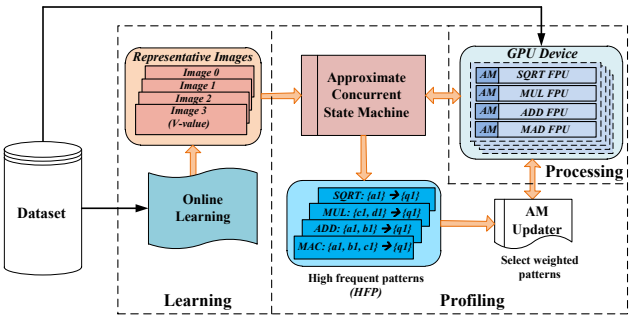


Figure 1. An architectural overview of the proposed ACAM

3.1 Associative Memory Design for GPU

In the proposed GPGPU architecture, we consider four key instructions: adder (ADD), multiplier (MUL), square root (SQRT) and multiply accumulator (MAC). Each instruction is computed on four FPUs (floating-point units) of each streaming core. Beside each stream core we deploy four associative memories corresponding to each computation. Two 32-bit input operands are given for ADD and MUL, one 32-bit operand for SQRT and three 32-bit operands for MAC. Each associative memory keeps profiled operand signatures in each row. Since the profiler is designed to give the signatures for frequent operands, the GPU processing can have higher chances to reuse the pre-stored output in the associative memory.

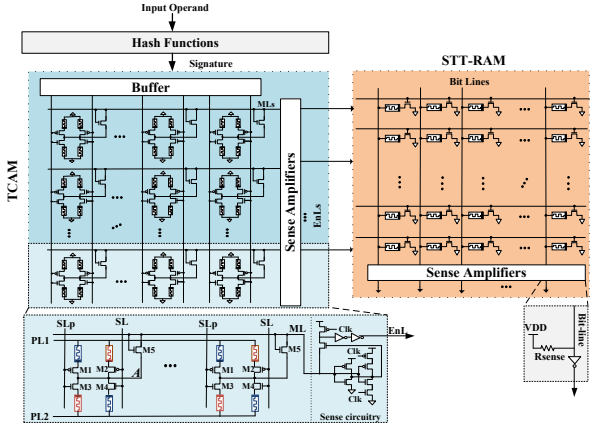


Figure 2. Associative memory structure of ACAM

Figure 2 shows the associative memory structure of the proposed associative memory. This memory consists of three blocks: hash functions, a TCAM and a STT-RAM. First, once an instruction is executed with operands on each FPU, the hash functions produce signatures of the operands, and the signature is searched through all TCAM rows. Using the TCAM technology, this search operation can be carried out in a single cycle. The hit in any TCAM rows stops the processor computation by activating the clock gating mode. Finally, TCAM hit activated the corresponding line of the STT-RAM memory to read the precomputed output data.

In this memory design, there is a key technical challenge which can be caused from working with the runtime updating. Most of the high speed and stable TCAMs are designed with resistive memory. The lifetime of these memories limit to 10^6-10^7 number of writes. The limited endurance avoids using TCAMs with high number of TCAM updates. In order to design an efficient associative memory for frequent run-time updating, we exploit the MTJ-based TCAM which is characterized by high endurance. Most of MTJ-based TCAMs [27] (e.g. 6T-2MTJ) have low sense margin and a long search delay (~2.1ns). Thus, to provide better performance for search operations, we use the 5T-4MTJ TCAM structure [16] which also has the capability of complementary search operations. 5T-4MTJ improves the sense margin of the cell to 240mV and increases the large ML (match line) swing which results in a sub-1.3ns search operation. In addition, the 5T-4MTJ provides high energy efficiency by doing a search operation with 0.68fJ/search/bit, which is suitable for the adaptive updates. As shown in Figure 2, the data is saved on the TCAM complementary using the values of MTJ1 and MTJ2. In the search operation, the drivers pre-charge the MLs. In turn, SL and SL-bar signals activate the access transistors. At the same time, the PL drives put the input search signal on PL1 and PL2 lines. In case of a mismatch on an input signature, the middle node of the cell (node *A*) turns to zero, and the leakage current through M5 discharges the ML.

The outputs of profiled operand signatures are stored in the STT-RAM as shown in the right side of Figure 2. The STT-RAM memory consists of a transistor and an MTJ device. To write in STT-RAM, EnL lines are selecting target cells and a write voltage is applied on the bitline. In the read mode, the EnL value is activated by the TCAM block to read a corresponding line of the STT-RAM. The sense amplifier of the STT-RAM is resistive-based and works with a sense resistor (R_{Sense}).

In most cases, a large size of TCAMs in a view of the word-size and the number of rows can provide higher hit rate. However, for the large size of TCAMs, high search energy consumption degrades total processing energy efficiency, and the long word size exhibits low cell stability due to the cell leakage. Moreover, the TCAM with many rows results in high energy consumption and slow performance for search operations due to the slow input buffer. Thus, it is required to fill the TCAM rows with reusable data, i.e., that have high probability for hit rate, while compensating the search energy of the associative memory. Next, we describe our learning and profiling methodology how to fill the TCAM rows with highly reusable data.

3.2 Online Learning-based Image Selection

In order to efficiently utilize the memory-based computation on the proposed GPU, the AM must keep the best precomputed data which will appear in the future. However, the design-time decision for pre-stored data cannot adapt workload changes. Moreover, even though we have whole large input dataset in the design time, due to the variety of applications and users, it is impossible to maintain all useful data in the limited memory area. Therefore, we utilize a runtime profiling methodology. The main challenge here is how to select the representative input dataset which are good to be profiled for future computations. We design our selection strategy based on the temporal locality of dataset. In reality, input multimedia images usually exhibit two important characteristics: (i) a similar image group is processed repeatedly and (ii) the groups can be changed over time mainly due to the context changes such as time of day and place [28]. For example, if the user takes pictures outside at night, the pictures would be characterized as a group whose background is mostly dark. In this case, since the computations on stream cores would be also similar, they are good candidates to be reused the computed data. As time goes by, the set of groups is likely to be changed due to the change of contexts, thus affecting the set of representative images. Our image selection policy, called TD-LRU, is designed to address these two characteristics as shown in Figure 3.

The first step is to characterize images groups while defining their future usability. We exploit temporal difference (TD) learning [29], which is an algorithm of the reinforcement learning class. The TD-LRU maintains k states which represent k image groups of interest. When an input image is given for every iteration, each state gets a reward value based on its pixel similarity to the input image. For example, more similar image groups gets higher reward values. More formally, for M states, $S_1, S_2, \dots S_M$, the algorithm manages a vector, $V_t =< V_t^{S_1}, V_t^{S_2}, \dots, V_t^{S_M} >$ where each vector element has an accumulated reward value of each tracked state for each iteration t. Once an input image is given at t-th iteration, each element $V_t^{S_i}$ is updated as follows.

$$V_t^{S_i} = R_t^{S_i} + \gamma \cdot V_{t-1}^{S_i}$$

In this equation, $R_t^{S_i}$ is the reword value for the pixel similarity between the input image and the tracked image of the state S_i (where $0 \le R_t^{S_i} \le 1$), and the parameter γ balances the impact of the obtained reward on the previous V-values. A large γ updates the V-values with more consideration of the accumulated rewards of previous stages, while a small γ prioritizes the effect of the current reward in each state. In this paper we set the value of γ by 0.95. Thus, once an input is given, each state is updated by their pixel similarities while giving higher priority for the previous rewards. Therefore, frequently observed image groups will have higher values in the vector over time, and they are considered as the good candidates to be profiled. We select top N images in the vector as the images of interest (where $N \le M$).

In order to consider the second characteristic, i.e., the temporal locality of image groups, we maintain another vector, $Q_t =< Q_t^{S_1}, Q_t^{S_2}, \dots, Q_t^{S_M} >$, which tracks the temporal access pattern of the image groups based on a LRU (Least Recently Used) policy. TD-LRU considers the images with less than a threshold reward, T_{acc}, as non-matched states. According to this criteria, $Q_t^{S_i}$ increases by 1 if non-matched, otherwise it is reset to 0. Thus, the image groups which have not been accessed for a long time will have a large value in the vector. In each iteration, if the rewards with all current states are less than a threshold, T_{new}, we consider this input as a new learning state. Until the k states are all set, we register the new state into the two vectors. Once the vector have all k states, the new state triggers a replacement policy, thus we select the image with the highest $Q_t^{S_i}$ as the victim to be replaced. To give an enough chance for the new image to be profiled, we put the V-value of the new state by the median value of V_t, thus the system can quickly react the changes of image sets, e.g., due to context changes.

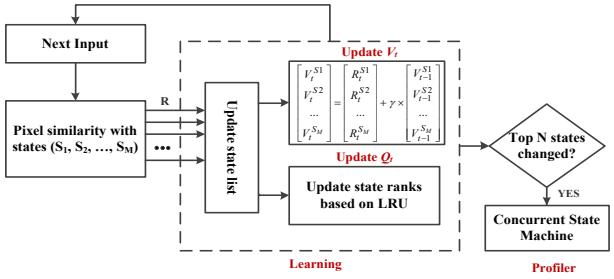


Figure 3. Online learning algorithm to identify images of interest

The two reward threshold values, T_{acc} and T_{new} give the system runtime controllability for running applications and workload. For example, if processor runs an interactive workload, i.e., when the image groups are frequently changed, higher T_{acc} and T_{new} values encourage the number of replacements with new states. If the system wants to decrease the cost of profiling, the lower thresholds can be chosen to avoid frequent profiled image set changes.

3.3 Approximate Profiler

The proposed profiler is running in parallel with main GPU computations to keep track of frequent operand patterns for the images selected by the learning. It would require a large memory

space to keep every operand with their counts to determine the ranking of the operands. In addition, it would also need high computation cost of both performance and energy to search if operands are already counted or not. The high cost of runtime profiling might be as much as it hides the advantage of using the memory-based computation. Thus, the proposed profiler identifies frequent operand patterns in an approximated manner to minimize the profiling overhead at the expense of the accuracy. We use an approximate concurrent state machine [25, 26]. The concurrent state machine exploits a bloom filter, which is implemented with *k* hash functions to generate an input signature. A signature is saved on an *m*-bits vector. The system error is defined with the size of vector and a degree of memberships which means how many operands might have a same signature value. In case of having *n* memberships the false positive error is given by:

$$f = (1\text{-}e^{-nk/m})^k.$$

In our evaluation with three hash functions which generate 60K different signatures of 128-bit length, we can profile the frequency pattern of operands with 5x lower memory space than the exact profiling case. Since we only need to approximately find a list of frequent operands rather than their exact ranking, we can still identify signatures of frequent operands with an acceptable error-rate of 5.3%. Thus, we can significantly speed up the profiling with a small impact on the result of final processor computations.

Based on the profiled result, we update the TCAM rows with *l*-top input operands for each FPU operation. Since we fill the rows for multiple representative images selected by TD-LRU, every image has different probabilities for future occurrences. We also consider this fact by allocating different portions of TCAM rows for each image. We utilize the vector V_t produced by TD-LRU so that more frequently observed image groups have higher proportions of the rows. For an image state S_i of *N* profiled states, the allocated proportion P^{S_i} is computed as follows:

$$P^{S_i} = \frac{V_t^{S_i}}{\sum_{j=1}^{N} V_t^{S_j}}.$$

For example, the frequently profiled operands of an image state with $V_t^{S_a} = 0.4$ will take 2x more number of rows than those of another image state with $V_t^{S_b} = 0.2$. This allocation strategy gives hit rate improvement by giving higher proportion in the TCAM rows for frequently observed image groups.

4. EXPRIMENTAL RESUTLS

4.1 Experimental Setup

In order to evaluate the proposed ACAM framework, we implement our technique based on Multi2Sim cycle-accurate processor simulator [30] for the recent AMD Southern Island GPU architecture. For example, Radeon HD 7000-series have been designed based on this architecture. Note that our proposed design can be implemented in most recent GPU architectures. For circuit level simulations, we have used HSPICE tool to design the TCAM array and the STT-RAM memory. We use NV*sim* tool [31] to estimate the energy consumption of memory accesses. The 6-stage balanced FPUs were designed using *Synopsys Design Compiler* in 45-nm ASIC flow. FPUs are optimized for power based on the corresponding TCAM delay in each size. To compute the energy consumption of the proposed associative memory, we utilized detail simulation parameters (e.g., sizing, resistors and capacitors, etc.) as reported in [13]. In turn, the total energy of the designed streaming processors can be computed using the number of computations extracted from the Multi2Sim simulator. We maintain 10 image states for online learning and select top five images to perform runtime profiling (i.e., *M=10* and *N=5*). We empirically set two thresholds by $T_{acc} = 0.1$ and $T_{new} = 0.3$ for used input data. In order to further decrease the running overhead of the pixel similarity computation, we scaled down images by half (i.e., $1/2$ of widths and heights) in the learning step. Even with the down sampling, we can still obtain enough accurate pixel similarities. In order to recognize when a program start executing the workload on the stream processor, we modified AMD compute abstraction layer (CAL). The AMD CAL provides a runtime device driver library that allows a host program to execute kernels, which is a program instance of the host program, on stream processors. Thus, we can profile and learn the images when the kernels are initiated and update the TCAM of all compute units in parallel.

We use four OpenCL image processing applications of AMD APP SDK v2.5 [32], *Sobel, Robert, Sharpen* and *Shift*. We used Caltech 101 computer vision [33] as input image dataset. In the experiment we compare our proposed ACAM to an offline profiling strategy. The offline profiling strategy utilizes pre-stored data computed for fixed image sets which are randomly chosen by 5% of whole dataset. In order to evaluate how the proposed technique works for different dataset sizes including a large number of images, we test our technique on variety of input dataset sizes from 100 to 4000 images. We also used two types of dataset to verify the advantage of our design: (i) locality dataset and (ii) random dataset. For the locality dataset, we first made an initial sequence so that similar images (e.g., cat and butterfly) appear together, and then swapped two images which were randomly selected within a window of 5% of the dataset size. We repeated the swap iteration by the number of images in the dataset. In contrast, for the random data we chose the sequence of images randomly. In order to evaluate the accuracy of the results of the image processing applications, we compare with the golden picture of the exact computation.

4.2 Impact on Accuracy and Overhead

As explained in Section 3.3, we can control the expected amount of the false positive error of the profiler by changing the number of hash functions (*k*) and bloom vector size (*m*). In our configuration, the false positive error is 5.3%. In order to understand how the false positive error affects the results of actual applications, we evaluate PSNR values over different TCAM row sizes. Table 1 shows the evaluation results. Since smaller than 30db in PSNR is assumed to be not acceptable [34], for the *Sobel* and *shift* applications, we assumed that the number of TCAM rows should be limited to 64-row. For example, Figure 4 shows the visual results of *Robert* application using the original computation (i.e., the golden image case) and approximate computation, resulting in no perceivable change.

Table 1. PSNR comparison for different applications and TCAM sizes

PSNR	*Number of rows*					
	4-row	*8-row*	*16-row*	*32-row*	*64-row*	*128-row*
Robert	62	60	58	52	47	41
Sobel	59	49	46	41	39	28
Sharpen	64	62	58	55	51	46
Shift	47	41	38	35	31	26

Original Image | Exact computing | Approximate computing PSNR=41dB

Figure 4. Output quality comparison in Robert application

The proposed design does not sacrifice the performance compared to the original GPU execution. The additional procedures for profiling and online learning are executed with the GPU computations in parallel, and the proposed associative memory is designed such that the memory-based computation can perform with the same clock frequency of the FPUs. In addition, the area overhead is negligible, since the number of TCAM rows is very small, i.e., less than 64. However, the proposed ACAM may add energy overhead due to learning, profiling and additional associative memories. Thus, we thoroughly evaluate the energy overhead in the following section.

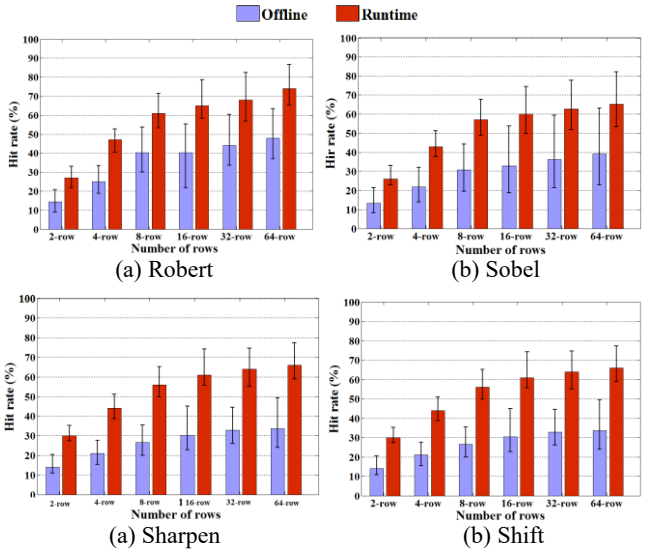


Figure 5. Hit rate comparison for offline profiling and online ACAM

4.3 Hit Rate and Energy Saving Comparison

Figure 5 compares the hit rate of ACAM using the offline and runtime profiling over different TCAM sizes. In the experiment we use the locality dataset containing 1000 images. As shown in the result, the proposed method always outperforms the offline profiling in terms of the hit rate, showing that the GPU can get higher chance to apply the clock gating. This result presents that the ACAM can adaptively fill the TCAM with better input patterns by considering all the data set given to the system so far. In addition, with the larger TCAM size, the hit rate difference also becomes higher. It is because the larger TCAM spaces allows to keep more enough operands, which are used again, from the selected images.

To evaluate how the hit rate affects the energy consumption, we compare the energy saving of two profiling strategies to the exact computation case, called *Exact-FPU*. Figure 6 shows the normalized energy consumption of the GPGPU to that of Exact-FPU over different associative memory sizes. The results show that the proposed ACAM improves the GPGPU energy saving by 34.7% on average. In addition, we observed that the level of hit rate significantly influences the GPU energy saving. For example, since a small size TCAM (e.g., 2 rows) would not provide a high hit rate, the approximation does not change the energy consumption dramatically. However, because of higher hit rate for the larger sizes, the proposed ACAM always outperforms the offline profiling strategy in terms of energy consumption. The ACAM saves 2.9x more energy on average than the offline profiling strategy.

In order to better understand how the ACAM consumes energy, we breakdown the energy consumption into two energy parts, energy computation in FPUs with associative memories (i.e., red-color bar) and energy overhead of online learning and profiling considering memory accesses (i.e., black-color bar). For both offline and runtime profiling, the minimum energy point is observed in the middle of TCAM sizes. The offline profiling strategy shows the minimal energy point for around 8-row case, while the proposed ACAM for around 16-row case. It is because there is an energy trade-off between the FPU and the associative memory. In the ACAM case, the learning and profiling energy is always consumed at a same level since the dataset is fixed. Although a larger TCAM size provides higher hit rate, it also requires a larger amount of energy cost. However, because of the higher hit rate, the FPU can be in the clock gating mode in a longer time, resulting in overall energy saving. For example, using the TCAM size of 64-row does not improve the FPU hit rate as much

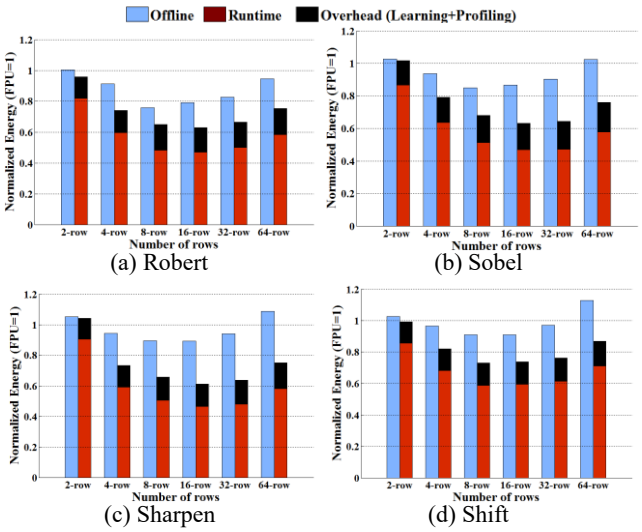


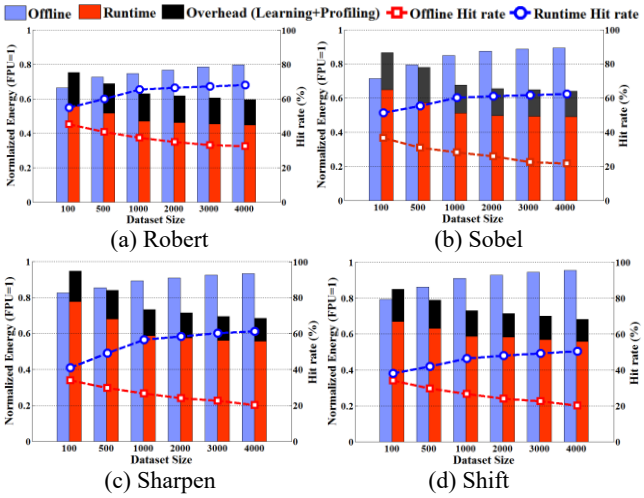
Figure 6. GPGPU energy saving over different ACAM sizes for offline strategy and online ACAM

as it could compensate the TCAM energy cost. Thus, in our case, 16-row is the best setting to achieve the highest energy saving in general. Moreover, the minimum energy point of the offline profiling is shifted to a larger TCAM size in the online profiling, i.e., from 8-row to 16-row. The online profiling achieves higher hit rate improvement in the larger TCAM size, thus showing that the online profiling is an important design strategy to achieve higher energy efficiency.

4.4 Comparison for Different Datasets

Figure 7 shows the impact of increasing the dataset size from 100 to 4000, on GPGPU energy efficiency and ACAM hit rate improvement for the offline profiling and the online ACAM. For a fair comparison for the energy saving, we use 8-row TCAM for the offline profiling, while 16-row TCAM for the ACAM case. Figure 7 shows that the proposed runtime profiling shows higher-energy saving on the large-size input dataset, which is more general for the recent IoT workload. For example, for the largest dataset, i.e., 4000 images, the online ACAM method can adaptively update pre-stored the TCAM values in time by considering the data locality. Thus, we observed that the ACAM presents 3.3x more energy saving improvement, on average for four applications, than the offline case. In addition, we observed that the energy overhead for profiling becomes lower for larger dataset sizes. Since the ACAM profiles workload when selected representative image states are changed, we need relatively frequent profiling until the selected images are good-enough to represent the dataset. Then, once the online learning has sufficient chances to cover the dataset, the number of profiling starts being saturated. Thus, we conclude that the proposed ACAM technique performs better with practical applications which have to handle the substantial amount of workloads.

However, the naïve sampling method of the offline profiling cannot identify the proper images to be profiled, resulting in low hit rate. Therefore, the low hit rate degrades the GPGPU energy efficiency, compared to the ACAM case. In contrast, for a small size dataset, e.g., 100, the hit rate difference between the two strategies are small. Thus, the offline technique has better energy efficiency due to its zero profiling energy. However, we observed that the energy saving of the offline method decreases as the dataset size increases, even though the offline method can profile more images by profiling 5% of dataset to get enough knowledge. Since the TCAM size must be limited in the design time (e.g., 64-row in this case), the offline profiling cannot put enough high-frequent patterns, preventing from the TCAM hit rate improvement. In contrast, our online profiling method can adaptive update the TCAM based on the temporal locality, resulting in high hit rate and energy saving.



(a) Robert (b) Sobel

(c) Sharpen (d) Shift

Figure 7. Normalized energy consumption of GPGPU system using different dataset size

In order to understand how the ACAM works with different levels of data locality, we also evaluate the ACAM framework with the random dataset, which has no data locality. In general, multimedia applications have high temporal locality, but few applications may handle datasets with low data locality. In this experiment, we define $\Delta E = E_{Random} - E_{Locality}$ as a GPGPU energy difference metric for the random and locality dataset cases, where each energy consumption is normalized to the *Exact-FPU* case. Table 2 lists ΔE for all applications in different dataset sizes. Intuitively, the ACAM always performs better with the locality dataset since the proposed TD-LRU is optimized to find the locality of the data. We also observed that the energy difference grows as the data size increases, since the online learning algorithm can make better decisions for image selections based on the locality. Therefore, in this case, the best energy saving occurs in the largest dataset where learning algorithm has better chance to find the relevant image for profiling. However, we also observed that, even with the random dataset, the ACAM saves the energy of GPGPU and outperforms the offline profiling strategy. It is because our learning algorithm can also identify distinct images in different image states, thus selecting operands which covers more representative images in the whole dataset.

Table 2. Normalized GPGPU energy saving difference (ΔE) using locality and random dataset

	Dataset size					
	100	***500***	***1000***	***2000***	***3000***	***4000***
Robert	0.04	0.04	0.07	0.09	0.11	0.13
Sobel	0.02	0.06	0.13	0.15	0.16	0.18
Sharpen	0.05	0.08	0.14	0.17	0.19	0.21
Shift	0.04	0.05	0.09	0.11	0.12	0.14

5. CONCLUSION

In this paper we propose an approximate computing framework using an adaptive associative memory (ACAM) with the capability of the runtime profiling based on online learning. To support runtime updates for the associative memory, the proposed technique exploits MTJ-based TCAM memory which solves the low endurance problem of the resistive TCAM. In the proposed technique, the machine learning algorithm is also used to find the best images of interest based on the proposed TD-LRU algorithm. The approximate concurrent state machine is also implemented for profiling selected input images, decreasing the number of profiling overhead by ~5X with a low error-rate of 5.3%. Our experimental results on AMD southern Island GPU show that the proposed design achieves on average 34.7% energy saving which is 2.1X more than the offline profiling case, while providing acceptable quality of service.

6. ACKNOWLEDGMENT

We would like to thank Mr. Mahdi Imani in Genomic Signal Processing Laboratory at Texas A&M University for technical discussion. This work was supported by NSF grant #1527034 and UCSD Powell Fellowship.

7. REFERENCES

[1] S. W. Keckler, et al., "GPUs and the future of parallel computing," *IEEE Micro*, pp. 7-17, 2011.
[2] H. Tabkhi, et al., "Function-Level Processor (FLP): A High Performance, Minimal Bandwidth, Low Power Architecture for Market-Oriented MPSoCs," *IEEE Embedded Systems Letters*, vol. 6, pp. 65-68, 2014.
[3] J. Manyika, et al., "Big data: The next frontier for innovation, competition, and productivity," 2011.
[4] B. K. Mathew, et al., "Design of a parallel vector access unit for SDRAM memory systems," *IEEE HPCA*, pp. 39-48, 2000.
[5] A. Rahimi, et al., "Approximate associative memristive memory for energy-efficient GPUs," *ACM DATE*, pp. 1497-1502, 2015.
[6] K. Lakshminarayanan, et al., "Algorithms for advanced packet classification with ternary CAMs," *ACM SIGCOMM Computer Communication Review,* 2005, pp. 193-204.
[7] J. Li, et al., "1 Mb 0.41 µm² 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing," *IEEE JSSC*, vol. 49, pp. 896-907, 2014.
[8] M. Imani, et al., "Resistive Configurable Associative Memory for Approximate Computing," IEEE/ACM DATE, pp. 1327 – 1332, 2016.
[9] M. Imani, et al., "ReMAM: Low Energy Resistive Multi-Stage Associative Memory for Energy Efficient Computing," *IEEE ISQED*, 2016
[10] S. Paul, et al., "Nanoscale reconfigurable computing using non-volatile 2-d sttram array," *IEEE-NANO*, pp. 880-883, 2009.
[11] J. Cong, et al., "Energy-efficient computing using adaptive table lookup based on nonvolatile memories," *IEEE ISLPED*, pp. 280-285, 2013.
[12] Y. Kim, et al., "CAUSE: critical application usage-aware memory system using non-volatile memory for mobile devices," *IEEE/ACM ICCAD,* pp. 690-696, 2015.
[13] T. Hanyu, et al., "Spintronics-based nonvolatile logic-in-memory architecture towards an ultra-low-power and highly reliable VLSI computing paradigm," *IEEE/ACM DATE*, pp. 1006-1011, 2015.
[14] T. Kohonen, "Associative memory: A system-theoretical approach" *Springer Science & Business Media*, vol. 17, 2012.
[15] K. Pagiamtzis, et al., "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE JSSC*, vol. 41, pp. 712-727, 2006.
[16] L. Chisvin, et al., "Content-addressable and associative memory: Alternatives to the ubiquitous RAM," *IEEE Computer*, pp. 51-64, 1989.
[17] A. Rahimi, et al., "Energy-efficient gpgpu architectures via collaborative compilation and memristive memory-based computing," *ACM/IEEE DAC*, pp. 1-6, 2014.
[18] N. Bandi, et al., "Fast data stream algorithms using associative memories," *ACM SIGMOD*, pp. 247-256, 2007.
[19] C. Ranger, et al., "Evaluating mapreduce for multi-core and multiprocessor systems," *IEEE HPCA*, pp. 13-24, 2007.
[20] R. Agrawal, et al., "Fast algorithms for mining association rules," *IEEE VLDB*, pp. 487-499, 1994.
[21] T. Kohonen, "Content-addressable memories" *Springer Science & Business Media*, vol. 1, 2012.
[22] H. Zhang, et al., "Low power gpgpu computation with imprecise hardware," *ACM/IEEE DAC*, pp. 1-6, 2014.
[23] M. Imani, et al., "MASC: Ultra-Low Energy Multiple-Access Single-Charge TCAM for Approximate Computing," *IEEE/ACM DATE, pp. 373-378,* 2016.
[24] M. Imani, et al., "Approximate Computing using Multiple-Access Single-Charge Associative Memory," *IEEE Transaction on Emerging Topics in Computing (TETC),* 2016.
[25] F. Bonomi, et al., "Beyond bloom filters: from approximate membership checks to approximate state machines," *ACM SIGCOMM Computer Communication Review*, vol. 36, pp. 315-326, 2006.
[26] Y. Hua, et al., "Locality-sensitive bloom filter for approximate membership query," *IEEE Computers*, vol. 61, pp. 817-830, 2012.
[27] S. Matsunaga, et al., "Fully parallel 6T-2MTJ nonvolatile TCAM with single-transistor-based self match-line discharge control," *IEEE VLSIC*, pp. 298-299, 2011.
[28] T. M. T. Do, et al., "Smartphone usage in the wild: a large-scale analysis of applications and context," *ACM international conference on multimodal interfaces*, pp. 353-360, 2011.
[29] G. Tesauro, "Temporal difference learning and TD-Gammon," *ACM Communications*, vol. 38, pp. 58-68, 1995.
[30] R. Ubal, et al., "Multi2Sim: a simulation framework for CPU-GPU computing," *IEEE/ACM PACT*, pp. 335-344, 2012.
[31] X. Dong, et al., "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, pp. 994-1007, 2012.
[32] "AMD APP SDK v2.5" Available at: "http://www.amd.com/stream"
[33] Available at: "http://www.vision.caltech.edu/Image_Datasets/Caltech101/"
[34] Q. Huynh-Thu, et al., "Scope of validity of PSNR in image/video quality assessment," *IEEE Electronics letters*, vol. 44, pp. 800-801, 2008.